UNIT - II

1. REGULAR EXPRESSIONS

- Regular expressions are shorthand notations to describe a language. They are used in many programming languages and language tools like lex, vi editor etc. They are used as powerful tools in search engines.
- Regular expressions (RE) are useful for representing certain sets of strings in an algebraic fashion. RE describes the language accepted by finite state automata.

Definition:

Let ∑ be a given alphabet. Then

(i) ϕ, $\in$, and 'a' $\in \sum$ are all Regular expressions. These are called 'Primitive Regular expressions'

(ii) If r1 and r2 are regular expressions, so are r_{1+} r_2, r_1r_2, r_1* and (r_1).

(iii) A string is a Regular expression, if and only if it can be derived from the primitive Regular expressions by a finite number of the rules in (ii).

1.1 Language Associated with Regular Expressions:

Regular expressions can be used to describe some simple languages. If r is a regular expression, we will let L(r) denote the language associated with r The language is defined as follows.

Definition:

The language L(r) denoted by any regular expression r is Defined by following rules.

1) ϕ is a regular expression denoting the empty set.
2) $\in$ is a regular expression denoting the set$\{\in\}$
3) For every a$\in\sum$, 'a' is a regular expression denoting set set $\{a\}$.

   If r_1 and r_2 are regular expressions, then
4) $L(r_1+r_2)=L(r_1)\cup L(r_2)$
5) $L(r_1.r_2)=L(r_1)L(r_2)$
6) $L(r_1^*)=(L(r_1))^*$

Example:

Exhibit the language L $(a^*.(a+b))$ In set notation

$$L(a^*.(a+b))=L(a^*)L(a+b)=(L(a^*))(L(a) \cup L(b))$$

$$=\{\in,a,aa,aaa,.....\}\{a,b\}=\{a,aa,aaa,.....,b,ab,aab,...\}$$

1.2 Precedence of Regular Expression Operators

(1) Kleene closure has higher precedence than concatenation operator.

(2) Concatenation has higher precedence than union operator.

1.3 Equivalence of Regular expressions:

- Two regular expressions are said to be equivalent if they denote the same language

Example: Consider the following regular expressions

$r_1=(1^*011^*)^*(0+\in)+1^*(0+\in)$ and $r_2=(1+01)^*(0+\in)$

Both r_1 and r_2 represent the same language i.e. the language over the alphabet {0,1} with no pair of consecutive zeros. So r_1 and r_2 are said to be equal.

1.4 Algebraic Laws For Regular Expressions:

Let r_1, r2 and r_3 be three regular expressions.

1. Commmmtative law for union:

- The commutative law for union, say that we take the union of two languages in either order. i.e. $r_1+r_2=r_2+r_1$

2 Associative laws for union:

- The association law for union says that we may take the union of three languages either by taking the union of the first two initially or taking the union of the last two initially.

$$(r_1+r_2)+r_3= r_1+(r_2+r_3)$$

3. Associative law for concatenation:

$$(r_1r_2)\ r3= r_1\ (r_2r_3)$$

4. Distributive laws for concatenation:

- Concatenation is left distributive over union

  i.e. $r_1\ (r_2+r_3) = r_1r_2+ r_1r_3$

- concatenation is right distributive over union

  i.e. $(r_1+r_2)\ r3=r_1r_3+ r_2r_3$

5. Identities For union And Concatenation:

- ϕ is the identity for union operator

  i.e. $r_1+\phi=\phi+r_1=r_1$

- $\in$ is the identity for concatenation operator

  i.e. $r_1\in=\in r_1=r_1$

6. Annihilators for Union and Concatenation:

- Annihilator for an operator is a value such that when the operator is applied to the Annihilator and other value, the result is the Annihilator**.**

  ϕ is the Annihilator for concatenation

  i.e. $\phi r_1= r_1\phi=\phi$

  there is no Annihilator for union operator.

7. Idempotent law for Union:

- This law states that if we take the union of two identical expressions, we can replace them by one copy of the expression.

  i.e. $r_1+r_1=r_1$

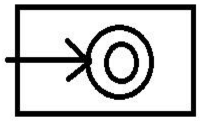
8. Laws involving closure

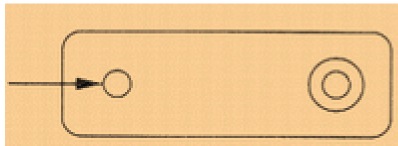
- Let 'r' be a regular expression ,then

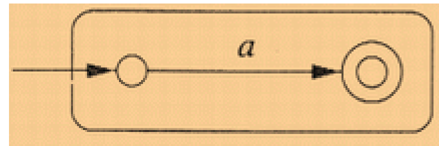
  1. $(r^*) =r^*$
  2. $\phi^*=\in$
  3. $\in^*=\in$
  4. $r^+=r.r^*=r^*.r$ i.e. $r^+=rr^*=r^*r$
  5. $r^*=r^++\in$
  6. $r?=\in+r$ (Unary postfix operator ? means zero or one instance)

2. Construction of ∈ - NFA from a regular expression

Basis: Automata for ∈ , ϕ and 'a' are (a),(b) and (c) respectively.

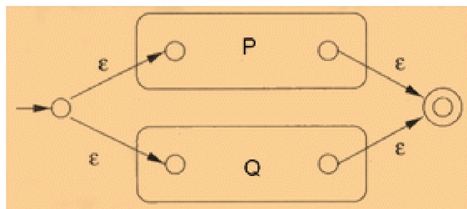




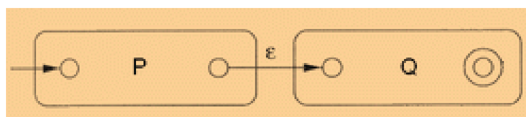


a) Accepting ∈ b) Accepting ϕ c) Accepting a

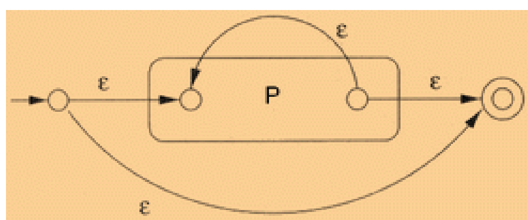
Induction: Automata for P+Q, PQ and P^* are (d), (e) and (f) respectively.



d) P+Q



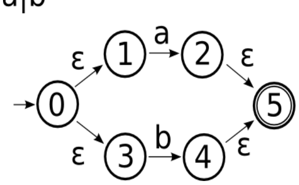
e) PQ



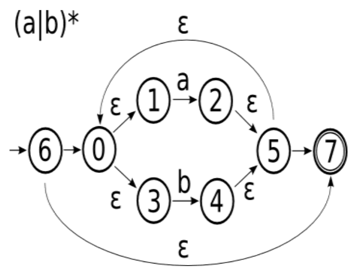
f) P^*

Example: Construct ∈ -NFA for the regular expression $(a|b)^*|c$

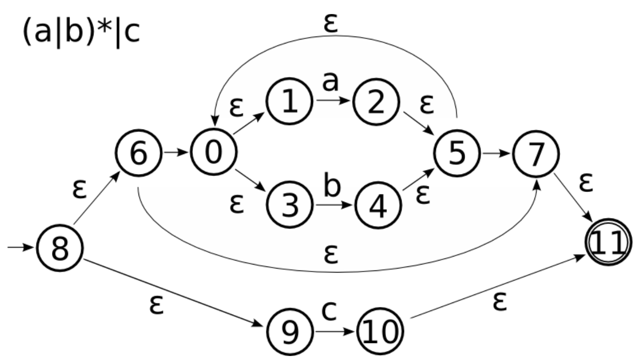
Solution: using Thompson's Construction. First we construct the union of a and b: a|b



Next we construct the Kleene Star of the previous union:



Finally we create the union between this and the next symbol c:



2.1 Construction of DFA from a regular expression:

This procedure can be explained easily with an example.

STEP 1: Augment the given regular expression with the symbol '# `

Ex: if (a|b)*abb is the given regular expression. After augmenting # the regular expression becomes (a|b)*abb#

STEP 2: Give positions to each symbol in the regular expression including # symbol.

Ex: (a | b)*a b b #
↓ ↓ ↓ ↓ ↓ ↓
1 2 3 4 5 6

STEP 3: Find Firstpos of the given regular expression. Firstpos is a set that contains the positions of all the symbols which can be at the beginning of a valid word of the regular expression.

Ex: The Firstpos (a|b)*a b b #) is {1,2,3} which are corresponding to the words given
1 2 3 4 5 6

Below

'1' is included in the Firstpos because of the word a b a b b # or a a b b #
1 2 3 4 5 6 1 3 4 5 6

'2' is included in the Firstpos because of the word b a b b #

2 3 4 5 6

'3' is included in the Firstpos because of the word a b b #

3 4 5 6

STEP 4 :Find following of each symbol. Followpos of a symbol is a set which contains the positions of all the symbols which can follow the current symbol.

Ex: in the regular expression (a | b)* a b b #

1 2 3 4 5 6

Followpos(1)= {1,2,3}

Followpos(2)= {1,2,3}

Followpos(3)= {4}

Followpos(4)= {5}

Followpos(5)= {6}

Followpos(6)= ϕ

STEP 5: construct Dstates the set of states of DFA, and Dtran,the transition table for DFA by the procedure given below .the states in Dstates are sets of positions ;initially each state is "unmarked" and state becomes "marked "just before we consider its outtransitions .the start state of DFA is Firstpos(regular expression) which is computed in step 3 ,and the start state are all those containing the position associated with the marker #.

PROCEDURE:

Initially the only unmarked state in Dstates is start state

While there is an unmarked state T in Dstates *do begin*

Mark T;

For each input symbol a do begin

Let U be the set of positions that are in followpos(P) for some position p in T ,such that the symbol at position p is a ;

If U is not empty and is not in Dstates *then*

add U as an unmarked state to Dstates;

Dtran[T,a]:=U

End end

Example:

Root for DFA of regular expression (a/b)* abb is {1,2,3} from step3 .

Let this set be A and consider input symbol a .positions 1 and 3 are for a ,so let B=followpos(1) U followpos(3) = {1,2,3,4}. Since this set has not yet been seen,we set

Dtran[A.a]:=B and addB to Dstates.

When we consider input b,we note that of the positions in A ,only 2 si associated with b,so we must consider the set followpos(2)={1,2,3}.Since this set has alredy been seen,we do not add it to Dstates but we add the transition Dtran[A,b]:A.

Now consider B on input ‘a’ positions 1 and 3 are for ‘a’ in B so Dtran[B,a]=followpos(1) U followpos(3)={1,2,3,4}=B

On input ‘b’ Dtran[B,b]=follow pos(2) $\cup$ followpos(4)={1,2,3,4,5}

As this is the new state name it C and to Dstates . $\therefore$ Dtran[B,b]=C.

Now we consider state C on input ‘ a’.

Dtran [C,a]=followpos(1) $\cup$ followpos(3)={1,2,3,4}=B

Dtran [C,b]=followpos(2) $\cup$ followpos(5)={1,2,3,6}

As this is the new state name it as D and add to Dstates. $\therefore$ Dtran[C,b]

Dtran [D,a]=followpos(1) $\cup$ followpos(3)={1,2,3}$\cup$ {4}={1,2,3,4}=B

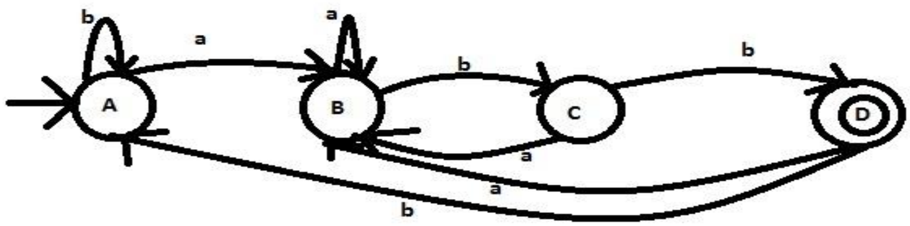
Dtran [D,a]=followpos(2)= {1,2,3}=A

As D only contains positional number of end marker # it is the only final state.

Transition table:

	a	b
A	B	A
B	B	C
C	B	D
*D	B	A

Transition Diagram:



3. Construction of regular expression from Finite Automata:

Arden's theorem: Let P and Q be two regular expression over ∑. If 'P' does not contain $\in$ then the equation in R=Q+RP has unique solution (i.e only one solution) given by $R=QP^*$.

Method for finding regular expression of Finite automata in transition diagram representation using Arden's theorem:

The following assumptions are made regarding the finite automata.

i. The finite automaton does not have $\in$ - moves.

ii. It has only one initial state, say q_0.

iii. It's states are $q_0, q_1 .... q_n$

i) Q_i is the regular expression representing the set of string accepted by the automata even through q_i is a final state.

ii) α_{ij} denotes the regular expression representation the set of labels of edges from v_i to v_j when there is no such edge $\alpha_{ij}=\phi$ .Consequently, we can get the following set of equation in $Q_1, \ldots Q_n$

$Q_1= Q_1\alpha_{11}+ Q_2\alpha_{21}+\ldots+ Q_n\alpha_{n1}+\in$

$Q_2= Q_1\alpha_{12}+ Q_2\alpha_{22}+\ldots+ Q_n\alpha_{n2}$

……………………………………………..

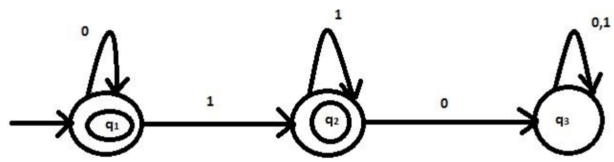
……………………………………………..

$Q_n= Q_1\alpha_{1n}+ Q_2\alpha_{2n}+\ldots+ Q_n\alpha_{nn}$

By repeatedly applying substitutions and Arden's theorem we can express Ri in terms of α_{ij}'s for getting the set of strings recognized by the automata, we have to take union of all Ri's Corresponding to final states.

Example1:

Derive a regular expression from the following given FA?



Sol:

$q_1=\in +q_10$ .... (1)

$q_2=q_1 1+q_2 1$ .... (2)

$q_3=q_2 0+q_3 0+q_3 1$ .... (3)

(2) → $q_2=q_1 1+q_2 1$

$q_2=q_1^{11^*}$ .... (4)

(1) → $q_1=\in +q_1 0$ (Apply Arden's theorem)

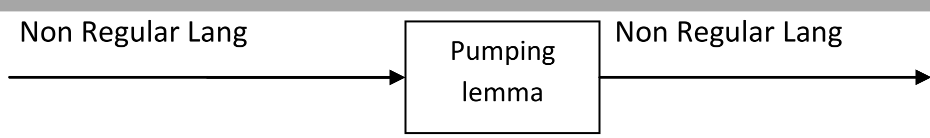
$q_1=\in 0^*$

$q_1=0^*$

(4) → $q_2=0^*11^*$

$q_2=0^*1^*$

4. Pumping lemma for Regular languages



Note

(1) Pumping Lemma is used to prove non-regularity of language.

(2) Pumping Lemma uses pigeon hole principle to show that certain languages are not Regular.

STATEMENT :Let 'L' be an infinite Regular language.then there exits some positive integer 'n' such that any $Z\in L$ with $|z|\geq n$ can be decomposed as Z=uvw

With $|uv| \leq n$, and $|v|\geq 1$,such that $z=uv^iw$ is also in L for all i=0,1,2,…….

Let us apply it on $L=\{0^n1^n/n\geq 1\}$

Proof :

Assume L to be Regular and apply pumping lemma.

By pumping Lemma

There exists some $n \geq 0$

We choose $w=0^n1^n$

Clearly $w \in L$ and $|w| \geq n$

$\Rightarrow$ By pumping Lemma

Choose $Z = uvw$ such that $v \neq \in$ and $|uv| \leq n$

But, since uv occurs at the beginning of the word and its total length can't be more than n, it is bound to have only 0's in it.

Let $|u|=a$,a,$|v|=b$ and $|uv|=a+b\leq n$,$b\geq0$

Then $z=0^a\ 0^b0^{n-a-b.}1^n$

If we choose k=0,

$Uv^kw \in L$

$U,w \in L$

$0^{n-b}1^n \in L$

But for b>0 ,this is not true ,a contradiction .

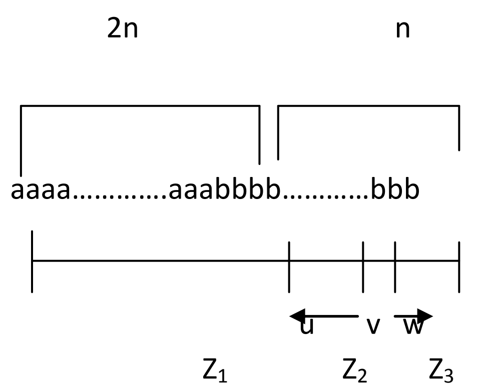
Hence L is not a Regular Language.

Example:

1. Prove or disprove the regularity if the language $\{a^ib^i \mid i>j\}$

Solution :

Let the given language is regular, So it must satisfy strong form of pumping lemma, and there exists a DFA with 'n' states.



(Z_3 Is ϕ)

$Z_1uv^iwZ_3 \in L$

Choose |v| =d and i =3n.

$a^{2n} b^{(n-d)+3nd} = a^{2n} b^{(3d+1)\ n-\ d}$

which has more number of the b's than a's .So the given language is not regular.

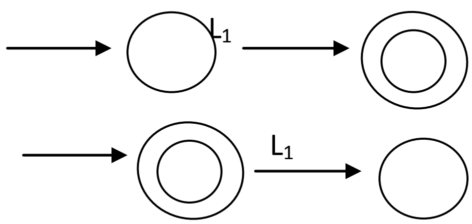
5. Closure Properties of Regular Languages

1.Regular languages are closed under union, concation and kleene closure.

2. Regular languages are closed under complementation.i.e .,if L_1 is a Regular language and $L_1 \subseteq \Sigma^*$,then $\overline{L_1} = \Sigma^* - L_1$ is Regular language.

$L_1 \Rightarrow$

$\overline{L_1} \Rightarrow$



3. Regular languages are closed under intersection .

i.e., if L_1 and L_2 are Regular languages then $L_1 \cap L_2$ is also a Regular language.

4. Regular languages are closed under difference.

i.e., if L and M are Regular languages then so is L-M

5. Regular languages are closed under Reversal operator.

6. Regular languages are closed under substitution.

7. Regular languages are closed under homomorphism and inverse homomorphism.