UNIT – II

Regular Languages

Regular Expressions:-

→ Regular expression, in short RE, is the Language Part of the type 3 grammar, which is called regular grammar.

→ RE is accepted by the machine called finite automata (FA).

→ RE was first proposed by an american mathematician Stephen Kleene as a notion of describing the algebra of a regular set.

→ RE became useful in the Unix text processing program 'grep' and modern programming languages which ar 'PERL'.

→ They are called 'regular' because they describe a Language with very 'regular' properties.

Basics of Regular Expression

An RE can be defined as a Language or string accepted by an FA. FA consists of 5 tuples $\{Q, \Sigma, \delta, q_0, F\}$.

→ Among them, an RE is a string on Σ, i.e., it consists of only input alphabets.

→ In short, a regular expression is written as RE or regex or regexp.

Some Formal Recursive Definitions of RE:

1. Any terminals that is the symbols that belong to Σ are RE. The null string (Λ) and the null set ($\emptyset$) are also RE.
2. If P and Q are two REs, then the union of the two REs denoted by $P+Q$ is also an RE.
3. If P and Q are two REs, then their concatenation denoted by PQ is also an RE.

4. If P is an RE, then the iteration (Repetition or closure) denoted by P^* is also an RE.
5. If P is an RE, then (P) is an RE.
6. The expressions got by the repeated application of the rules from (1) to (5) over Σ are also REs.

Identities of Regular Expression

→ An identity is a relation which is tautologically true.
→ In mathematics, an equation which is true for every value of the variable is called an "identity equation".
Ex: $(a+b)^2 = a^2+2ab+b^2$ is an identity equation
→ In the RE also, there are some identities which are true for every RE.

1) $\emptyset + R = R + \emptyset = R$, Proof: LHS: $\emptyset + R$
$= \emptyset \cup R$
$= \{\} \cup \{\text{Elements of } R\}$
$= R = RHS$

2) $\emptyset R = R\emptyset = \emptyset$, Proof: LHS: $\emptyset R$
$= \emptyset \wedge R$
$= \{\} \wedge \{\text{Elements of } R\}$
$= \emptyset = RHS$

Note: In above the cases, $\emptyset$ denotes a null set, which contains nothing.

3) $\wedge R = R \wedge = R$, PROOF: LHS: $\wedge R$
= Null string concatenated with any symbol of R
= Same symbol $\in R = R = RHS$

4) $\wedge^* = \wedge$ & $\emptyset^* = \wedge$

Proof: LHS: $\wedge^* = \{\wedge, \wedge\wedge, \wedge\wedge\wedge \ldots\}$
$= \{\wedge, \wedge, \wedge, \ldots\}$ [according to identity (3)]
$= \wedge = RHS$

Same for $\emptyset^*$.

5) $R+R=R$

Proof: LHS: $R+R=R(\Lambda+\Lambda)=R\Lambda$

$=R$ [according to identity (3)] = RHS

6) $R^*R^*=R^*$

Proof: LHS: $R^*R^*=\{\Lambda,R,RR\ldots\}\{\Lambda,R,RR\ldots\}$

$=\{\Lambda\Lambda,\Lambda R,\Lambda RR,\ldots,R\Lambda,RR,\ldots,RR\Lambda,RRR,\ldots\}$

$=\{\Lambda,R,RR,RRR,\ldots\}$ [using identity (3)]

$=R^*=RHS$

7) $R^*R=RR^*$

Proof: LHS: $R^*R=\{\Lambda,R,RR,RRR,\ldots\}R$

$=\{\Lambda R,RR,RRR,RRRR,\ldots\}$

$=R\{\Lambda,R,RR,RRR,\ldots\}$.

$=RR^*=RHS$.

8) $(R^*)^*=R^*$

Proof: LHS: $(R^*)^*=\{\Lambda,R^*R^*,R^*R^*R^*,\ldots\}$

$=\{\Lambda,R^*,R^*,\ldots\}$ [using identity (6)]

$=\{\Lambda,\{\Lambda,R,RR,RRR,\ldots\},\{\Lambda,R,RR,RRR,\ldots\},\ldots\}$

$=R^*=RHS$

9) $\Lambda+RR^*=\Lambda+R^*R=R^*$

Proof: LHS: $\Lambda+RR^*=\Lambda+R\{\Lambda,R,RR,RRR,\ldots\}$

$=\Lambda+\{R\Lambda,RR,RRR,RRRR,\ldots\}$

$=\Lambda+\{R,RR,RRR,RRRR,\ldots\}$

$=\{\Lambda,R,RR,RRR,RRRR,\ldots\}$

$=R^*=RHS$

10) $(PQ)^*P=P(QP)^*$

Proof: RHS: $(PQ)^*P=\{\Lambda,PQ,PQPQ,PQPQPQ,\ldots\}P$

$=\{P,PQP,PQPQP,PQPQPQP,\ldots\}$

$=P\{\Lambda,QP,QPQP,QPQPQP,\ldots\}$

$=P(QP)^*=RHS$.

11) $(P+Q)^*=(P^*Q^*)^*=(P^*+Q^*)^*$ (D' Morgan's theorem)

12) $(P+Q)R=PR+QR$

Proof: Let $a\in(P+Q)R$

$=a\in PR$ or QR = RHS

Ex-1: From the identities of RE, Prove that

$(1+100^*)+(1+100^*)(0+10^*)(0+10^*)^* = 10^*(0+10^*)^*$

Solution: LHS

$(1+100^*)+(1+100^*)(0+10^*)(0+10^*)^*$

$=(1+100^*)(\wedge+(0+10^*)(0+10^*)^*)$

$=(1+100^*)(0+10^*)^*$ (according to $\wedge+RR^*=R^*$)

$=1(\wedge+00^*)(0+10^*)^*$

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

(2) From the identities of RE, Prove that

$10+(1010)^*[\wedge+(1010)^*] = 10+(1010)^*$

Sol: LHS

$10+(1010)^*[\wedge+(1010)^*]$

$\Rightarrow 10+\wedge(1010)^*+(1010)^*(1010)^*$

$\Rightarrow 10+(1010)^*+(1010)^*$ As $\wedge R=R$ and $RR=R$

$\Rightarrow 10+(1010)^* = RHS$. As $R+R=R$

The Arden's Theorem

The Arden's theorem is used to construct the RE from a transitional system of FA.

Theorem: Statement: Let P and Q be two REs over Σ. If P does not contain $\wedge$, then the equation $R=Q+RP$ has a unique (one and only one) solution, $R=QP^*$.

Proof: Now, Point out the statements in the Arden's theorem in general form.

→ P and Q are two REs.

→ P does not contain the $\wedge$ symbol.

→ $R=Q+RP$ has a solution, i.e., $R=QP^*$

→ This solution is the one and only solution of the equation.

If $R=P/Q$

If $R = Qp^*$ is a solution of the equation $R = Q + RP$, then by putting the value of R in the equation, we shall get the value '0'.

$$R = Q + RP$$
$$R - Q - RP = 0$$

LHS $R - Q - RP$

(Putting the value of R in the LHS, we get)

$$Qp^* - Q - Qp^* p$$
$$= Qp^* - Q(\Lambda + p^* p)$$
$$= Qp^* - Qp^* \quad [\text{Ar } (\Lambda + R^* R) = R^*]$$
$$= 0$$

So, from here it is proved that $R = Qp^*$ is a solution of the equation $R = Q + RP$.

Process of constructing Regular Expression from FA

There are some assumptions:

→ In the transitional graph, there must be no Λ-move

→ In the FA, there is only one initial state.

① Now, we have to construct equations for all the states. There are n number of equations if there are n-states.

② For any FA, these equations are constructed in the following way.

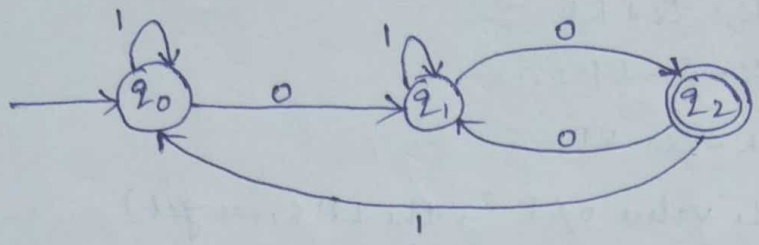
<state name> = Σ [<state name from which input is coming> . <input>]

③ For the beginning state, there is an arrow at the beginning coming from no state. So, a Λ is added with the equation of the beginning state.

④ Then, these equations have to be solved by the Identities of RE. The expression obtained for the final state and consists of only the input symbol (Σ) is the Regular Expression for the Finite Automata.

Examples

① Construct an RE from the given FA in below fig. by the algebraic method using Arden's theorem.



Sol :

For the given FA, the equations are

$$q_0 = q_0.1 + q_2.1 + \Lambda \longrightarrow (1)$$

$$q_1 = q_0.0 + q_1.1 + q_2.0 \longrightarrow (2)$$

$$q_2 = q_1.0 \longrightarrow (3)$$

Put the value of q_2 in equation no. (2)

$$q_1 = q_0.0 + q_1.1 + q_1.0.0$$

$$q_1 = q_0.0 + q_1(1 + 0.0) \longrightarrow (4)$$

This equation is in the form $R = Q + RP$

where $R = q_1$, $Q = q_0.0$ and $P = (1+0.0)$.

So, the solution of the equation is $R = QP^*$ that is

$$q_1 = q_0.0.(1+0.0)^*$$

Putting the value $q_2 = q_1.0$ in equation no. (1), we get

$$q_0 = q_0.1 + q_1.0.1 + \Lambda$$

Again putting the value of q_1 in the previous equation, we get

$$q_0 = q_0.1 + q_0.0(1+0.0)^*.0.1 + \Lambda$$

$$q_0 = q_0(1 + 0.(1+0.0)^*.0.1) + \Lambda$$

This equation is in the form $R = Q + RP$

$R = q_0$, $Q = \Lambda$ and $P = (1 + 0.(1+0.0)^*.0.1)$

So, the equation is $q_0 = \Lambda.(1 + 0.(1+0.0)^*.0.1)^*$

$$q_0 = (1 + 0.(1+0.0)^*.0.1)^*$$

Putting the value of q_0 in the solution of q_1,

we get $q_1 = (1+0.(1+0.0)^*.0.1)^*.0.(1+0.0)^*$.

Replacing the value of q_1 in equation (3),

we get $q_2 = (1+0.(1+0.0)^*.0.1)^*.0.(1+0.0)^*.0$.

As q_2 is the final state of the FA, all the strings will halt on this state.

Therefore, the RE is $(1+0.(1+0.0)^*.0.1)^*.0.(1+0.0)^*.0$, thus accepting the FA.

Equivalence of Two Finite Automata's

Two FA are said to be equivalent if they generate the same (equivalent) RE. If both the FA are minimized, they reveal the equivalence by being minimized to the same number of states and the same transitional functions.

The Process of Proving the equivalence between two FA

Let there are two FA, M and M', where the number of input symbols are the same.

1) Make a comparison table with n+1 columns, where n is the number of input symbols.
2) In the first column, there will be a pair of vertices (q, q') where $q \in M$ and $q' \in M'$. The first pair of vertices will be the initial states of the two machines M and M'. The second column consists of (q_a, q'_a), where q_a is reachable from the initial state of the machine M for the first input, and q'_a is reachable from the initial state of the M' for the first input. The other n-2 columns consist of a pair of vertices from M and M' for n-1 inputs, where $n = 2, 3, \ldots n-1$.

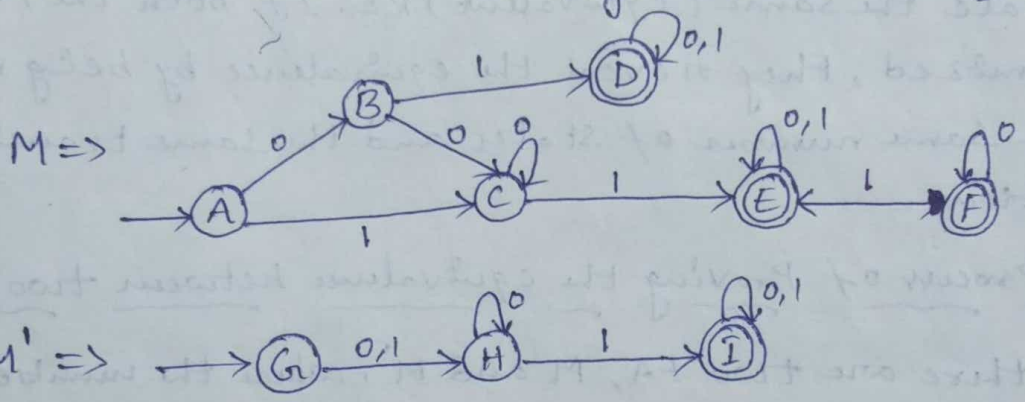
3) If any new pair of states appear in any of the n-1 next state columns, which were not taken in the first column, take that pair in the present state column and construct subsequent column elements like the first row.

4) If a pair of states (q, q') appear in any of the n columns for a pair of states in the present state column, where q is the final state of M and q' is the non-final state of M' or vice versa, terminate the construction and conclude that M and M' are not equivalent.

5) If no new pair of states appear, which were not taken in the first column, stop the construction and declare that M and M' are equivalent.

Examples

① Find whether the two DFAs given below are equivalent or not.



Sol

→ In the two DFAs, the inputs are two '0' and '1'. Therefore, the comparison table is of three columns.

→ The beginning state pair is (A, G). From there, for input 0, we are getting (B, H) and for input 1 we are getting (C, H).

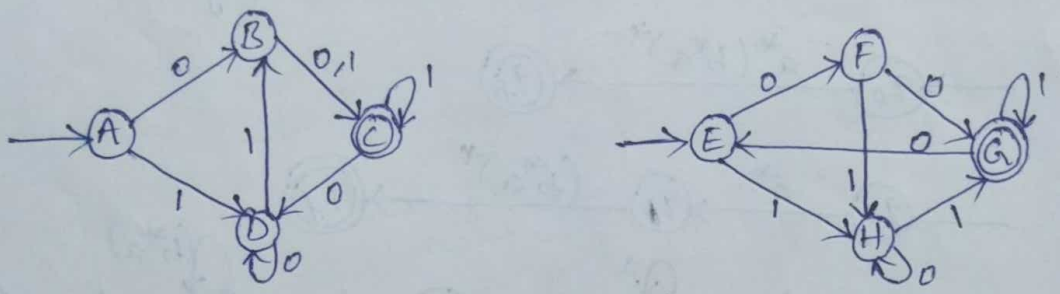
→ Both of them are new combination of states. Among them, first take (B, H) in the Present state column. The next state pairs for input 0 and 1 are constructed. By this; Process, the comparison continues.

→ After the fifth step, further construction is stopped as no new state combinations have appeared.

→ In the whole table, no such type of combination of states appear when one state is the final state of one machine and the other is the non-final state of another machine.

Present state (q,q')	Next State For I/p '0' (q_0,q_0')	Next State For I/p '1' (q_1,q_1')
(A, G)	(B, H)	(C, H)
(B, H)	(C, H)	(D, I)
(C, H)	(C, H)	(E, I)
(E, I)	(E, I)	(E, I)
(D, I)	(D, I)	(D, I)

(2) Find whether the two DFAs given below are equivalent or not.



→ The beginning state pair is (A,E). From there, for input 0, we get (B,F) and for input 1 we get (D,H). Both of them are new combination of states.

→ Among them, take (B,F) in the present state column. The next state pairs for input 0 is (C,G) and for input 1 is (C,H).

→ (C,H) is a combination of states where C is the final state of M and H is the non-final state of M'. Further construction stops here declaring that the two DFAs are not equivalent.

Present state	Next State For I/p '0' (q_0,q_0')	Next State For I/p '1' (q_1,q_1')
(A,E)	(B,F)	(D,H)
(B,F)	(C,G)	(C,H)

Equivalence of Two Regular Expressions

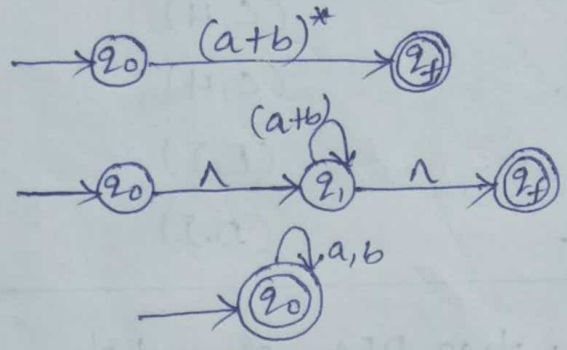
For every RE, there is an accepting FA. If the FA constructed from both of the REs are the same, then we can say that two REs are equivalent.

Examples

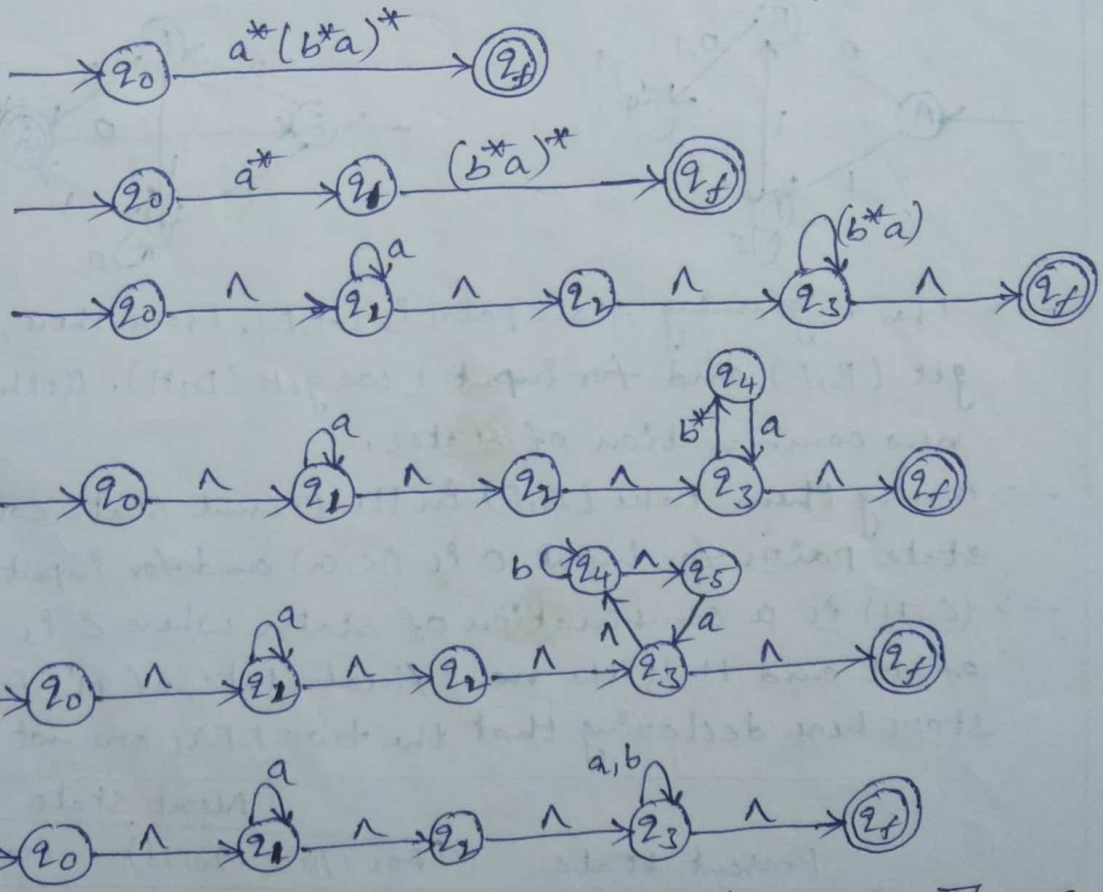
① Prove that the following REs are equivalent.

$$L_1 = (a+b)^* \quad L_2 = a^*(b^*a)^*$$

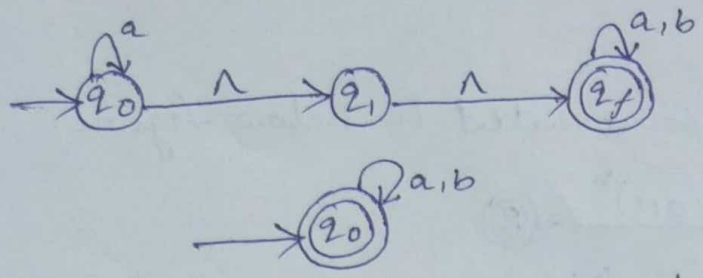
Sol: The FA for L_1 is constructed in below figure.



The FA for L_2 is constructed in below figure



q_5 is merged with q_4. q_4 is merged with q_3. Thus, a and b are both placed as loop on q_3.



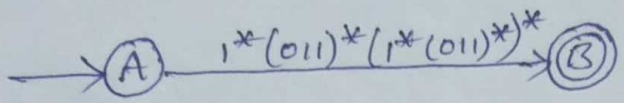
we are seeing that the FA generated by the two REs L_1 and L_2 are the same. So, we can decide that the two FA are equivalent.

(2) Prove that the following REs are equivalent.

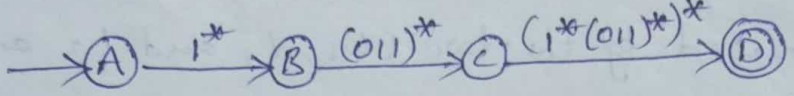
$$L_1 = 1^*(011)^*(1^*(011)^*)^* \quad L_2 = (1+011)^*$$

Sol: The FA for L_1 is constructed in below figure.

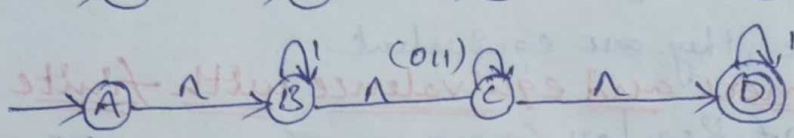
(a)



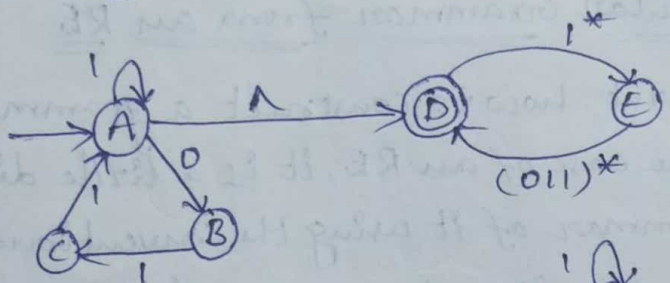
(b)



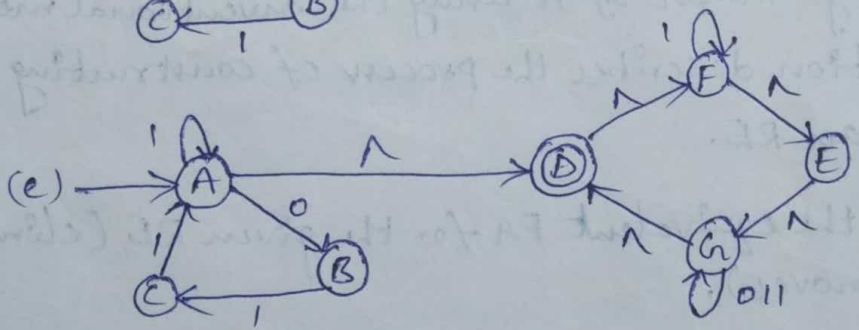
(c)



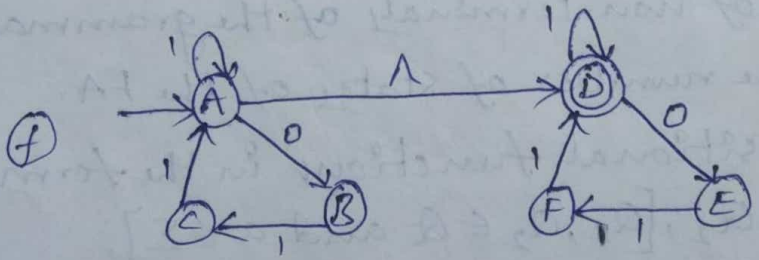
(d)



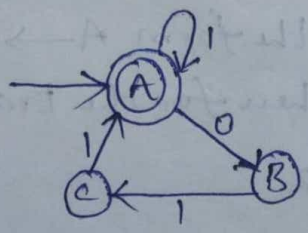
(e)



(f)



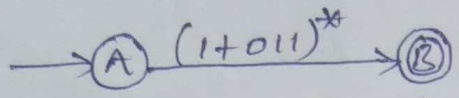
(g)



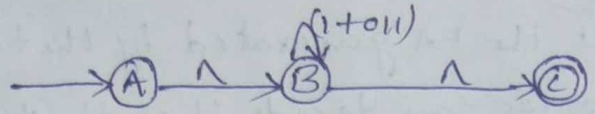
(24) Prove that

The FA for L_2 is constructed in below figure.

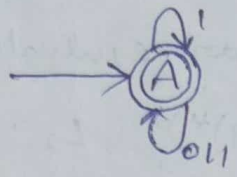
(a)



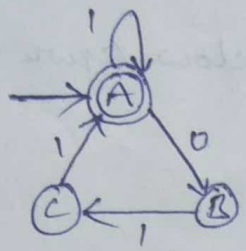
(b)



(c)



(d)



The FA generated by the two REs, L_1 and L_2 are the same and, therefore, they are equivalent.

Regular grammers and equivalence with finite automata

Construction of Regular Grammar from an RE

We have learnt how to construct a grammar from a language. But in the case of an RE, it is a little difficult to find the exact grammar of it using the conventional methods. The following section describes the process of constructing regular grammar from an RE.

Step I : Construct the equivalent FA for the given RE (eliminate all null moves).

Step II : The number of non-terminals of the grammar will be equal to the number of states of the FA.

Step III : For all transitional functions in the form

$\delta(Q_1, a) \rightarrow Q_2$, $[Q_1, Q_2 \in Q$ and $a \in \Sigma]$,

The production rule is in the form $A \rightarrow aA_1$.

If Q_2 is a final state, then for the transitional function $\delta(Q_1, a) \rightarrow Q_2$

The Production rules are $A \rightarrow aA$, and $A \rightarrow a$.
The start symbol is the symbol representing the initial state of the FA.

Examples

① Construct a regular grammar for the following REs.

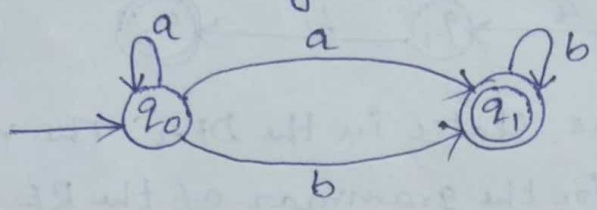
i) $a^*(a+b)b^*$

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

Sol:-

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

The FA for the string $a^*(a+b)b^*$ is constructed in below figure.



The number of states of the FA is 2, which means that there are two non-terminals in the grammar for the RE. Let us take them as A (for q_0) and B (for q_1).

For the transitional function $\delta(q_0, a) \rightarrow q_0$, the Production rule will be $A \rightarrow aA$. For the transitional function $\delta(q_0, a) \rightarrow q_1$, the Production rule will be $A \rightarrow aB$. As q_1 is a final state, there will be another production rule $A \rightarrow a$.

$\delta(q_0, a) \rightarrow q_0$: $A \rightarrow aA$

$\delta(q_0, a) \rightarrow q_1$: $A \rightarrow aB$ and $A \rightarrow a$
(as q_1 is a final state)

$\delta(q_0, b) \rightarrow q_1$: $A \rightarrow bB$ and $A \rightarrow b$
(as q_1 is a final state)

$\delta(q_1, b) \rightarrow q_1$: $B \rightarrow bB$ and $B \rightarrow b$
(as q_1 is a final state)

The start symbol will be A as q_0 is the beginning state.

The grammar G for the RE $a^*(a+b)b^*$ is $\{V_N, \Sigma, P, S\}$ where

$V_N = \{A, B\}$

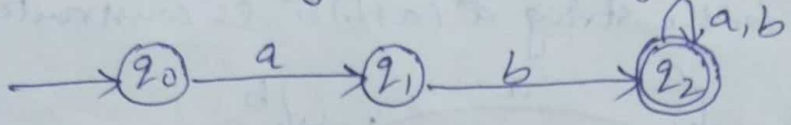
$\Sigma = \{a, b\}$

$P = \{A \to aA/aB/bB/a/b, B \to bB/b\}$

$S = \{A\}$.

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

The DFA for the string $ab(a+b)^*$ is given in below figure.



There are three states in the DFA. The number of non-terminals for the grammar of the RE will be 3. Let us take them as A (for q_0), B (for q_1) and C (for q_2).

For the transitional function

$\delta(q_0, a) \to q_1$, the Production rule will be $A \to aB$

$\delta(q_1, b) \to q_2$: $B \to bC$ and $B \to b$ (as q_2 is a final state)

$\delta(q_2, a) \to q_2$: $C \to aC$ and $C \to a$ (")

$\delta(q_2, b) \to q_2$: $C \to bC$ and $C \to b$ (")

The start symbol will be A as q_0 is the beginning state.

The grammar G for the RE $ab(a+b)^*$ is $\{V_N, \Sigma, P, S\}$ where

$V_N = \{A, B, C\}$

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

$P : \{A \to aB, B \to bC/b, C \to aC/bC/a/b\}$

$S : \{A\}$.

Constructing FA from Regular Grammar

FA can directly be constructed from regular grammar. This can be considered as a reverse process of constructing the FA from an RE. The following section describes the process of constructing the FA from an RE.

Step 1:

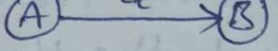
i) If the grammar does not produce any null string, then the number of states of the FA is equal to the number of non-terminals of the regular grammar + 1. Each state of the FA represents each non-terminal and the extra state is the final state of the FA. If it produces a null string, then the number of states is the same as the number of non-terminals.

ii) The initial state of the FA is the start symbol of the regular grammar.

iii) If the language generated by the regular grammar contains a null string, then the initial state is also the final state of the constructing FA.

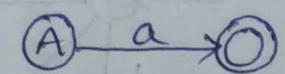
Step 2:

i) For a production in the form $A \rightarrow aB$, make a δ function

$\delta(A, a) \rightarrow B$ 

There is an arc from state A to state B with label a.

ii) For a production in the form $A \rightarrow a$, make a δ function

$\delta(A, a) \rightarrow$ final state 

iii) For a production $A \rightarrow \epsilon$, make a δ function

$\delta(A, \epsilon) \rightarrow A$, and A is the final state.

Examples

① Convert the following regular grammar into FA.

$S \rightarrow aA/bB/a/b$

$A \rightarrow aS/bB/b$

$B \rightarrow aA/bS.$

Sol: In the grammar, there are three non-terminals, namely S, A, and B. Therefore, the number of states of the FA is 4. Let us name the final state as C.

i) For the Production $S \rightarrow aA/bB$, the transitional diagram is given below

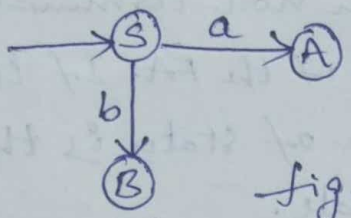


fig.(a)

ii) For the production $S \rightarrow a/b$, the transitional diagram including the previous one becomes as

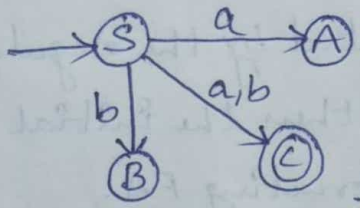


fig.(b)

iii) For the production $A \rightarrow aS/bB/b$, the transitional diagram including the previous one looks like fig (c).

For the Production $B \rightarrow aA/bS$, the transitional diagram including the previous one looks like fig (d)

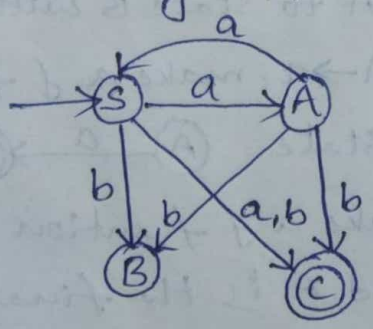


fig: (c)

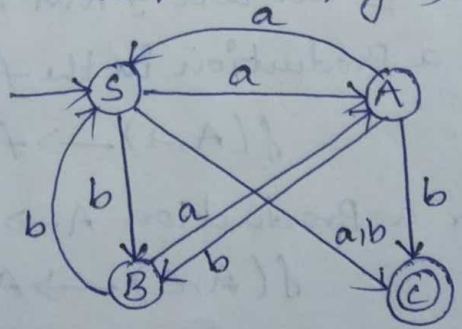


fig: (d)

(2) Convert the following regular grammar into FA.

$S \rightarrow aA/bS$

$A \rightarrow bB/a$

$B \rightarrow aS/b$

Sol:- In the grammar, there are three non-terminals, namely S, A, and B. So, the number of states of the FA will be 4. Let us name the final state as C.

i) For the Production $S \rightarrow aA/bS$, the transitional diagram becomes

ii) For the Production $A \rightarrow bB/a$, the transitional diagram including the previous one is

iii) For the Production $B \rightarrow aS/b$, the transitional diagram including the previous one looks like

This is the FA for the given regular grammar. The construction process is described in below figures.

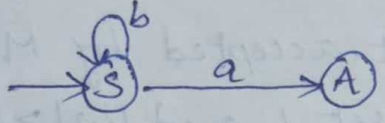


fig.(a)

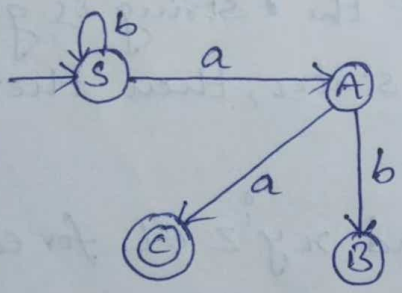


fig:(b)

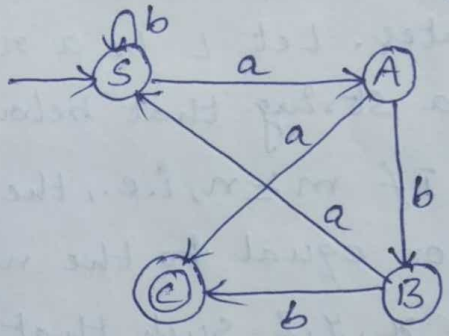


fig.(c)

Pumping Lemma for Regular Expression

There is a necessary condition for an input string to belong to a regular set. This necessary condition is the pumping lemma. Pumping means generating. This lemma is called a pumping lemma because it gives a method of generating many input strings from a given string.

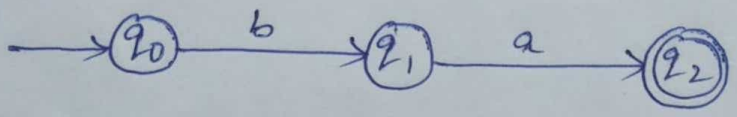
Pumping lemma is used to prove that certain sets are not regular. If any set fulfills all the conditions of pumping Lemma it can not be said confirm that the set is regular.

Theorem: Statement of the pumping lemma

Let $M = \{Q, \Sigma, \delta, q_0, F\}$ be an FA with n number of states. Let L be a regular set accepted by M. Let w be a string that belongs to the set L and $|w| \geq m$.

If $m \geq n$, i.e., the length of the string is greater than or equal to the number of states, then there exists x, y, z such that $w = xyz$

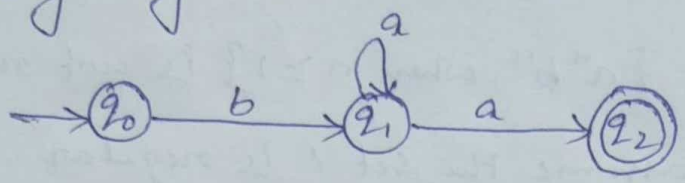
where $|xy| \leq n, |y| > 0$ and $xy^iz \in L$ for each $i \geq 0$.



The FA consisting of three states, q_0, q_1, and q_2. The string that is accepted by the FA is ba. The length of the string is 2 which is less than the number of states of the FA.

Here, from a state by getting a single input, it goes to a single distinct state. But, if the length of the string

is equal or greater than the number of states of the FA, then from a state by getting a single input it is not possible to get all single distinct states. There must be a repetition of at least a single state. This can be described by the following diagram.



The RE accepted by the automata is ba^*b. The expression can be divided into three parts: x, y, z. where y is the looping portion, x is the portion before looping, and z is the portion after the looping portion.

Applications of the pumping lemma

The pumping lemma is used to prove that certain sets are not regular. If an expression satisfies the conditions for the pumping lemma to be good, then it cannot be said that the expression is regular. But the opposite is true.

If any expression breaks any condition of the pumping lemma, it can be declared that the expression is not good.

This needs certain steps:

Step I: Assume that the let L is regular. Let n be the number of states of the FA accepting L.

Step II: choose a string w ($w \in L$) such that $|w| \geq n$.

By using the pumping lemma, we can write

$w = xyz$, with $|xy| \leq n$ and $|y| > 0$

Step III :- Find a suitable integer i such that $xy^iz \notin L$. This will contradict our assumption. From here, L will be declared as not regular.

Examples

① Prove that $L = \{a^n b^n \text{ where } n \geq 1\}$ is not regular

Sol :-

Step 1 :- Let us assume the set L is regular. Let n be the no. of states of the FA accepting L.

Step 2 :- Let $w = a^n b^n$. By the pumping lemma, we can write $|w| = xyz$ with $|xy| \leq n$ and $|y| > 0$.

Step 3 :- we want to find a suitable i so that $xy^iz \notin L$.

The string y can be of any one of the following

$$L = a^n b^n$$

$$L = aaabbb$$

Pumping length = P

$$L = a^P b^P$$

$S = a^P b^P$ $\quad P = 7$

x y z

$S = aaaaaaabbbbbbb$

Case-1 : The y is in the 'a' Part

$S = \underbrace{aa}_{x}\underbrace{aaaaa}_{y}\underbrace{abbbbbbb}_{z}$ $\Rightarrow$ $xy^iz \Rightarrow xy^2z$

$aaaaaaaaaaaabbbbbbb$

$11 \neq 7$

Case 2 :- The y is in the 'b' Part $\Rightarrow xy^iz \Rightarrow xy^2z$

$S = \underbrace{aaaaaaabb}_{x}\underbrace{bbbbb}_{y}\underbrace{b}_{z}$

$aaaaaaabbbbbbbb$

bbb

$7 \neq 11$

case 3:- The y is in the 'a' and 'b' Part

$$S = \underbrace{aaaaaa}_{x}\underbrace{aabb}_{y}\underbrace{bbbbb}_{z}$$

$xy^iz \Rightarrow xy^2z$

aaaaaa aabbaabbbbbbb

$|xy| \leq P \quad \therefore P = 7$ and $xy^2z \notin L$

case ① satisfied this condition and

case ② and case ③ not satisfied this condition.

So, ~~those~~ case(1) ~~not~~ satisfied these conditions and it is not regular, $xy^iz \notin L$.

② Prove that $L = \{yy / y \in (0,1)^*\}$ is not regular

Sol:

Step 1:- Let us assume the set L is regular. Let n be the no. of states of the FA accepting L.

Step 2:- Let $w = yy$. By the pumping Lemma. we can write $|w| = xyz$ with $|xy| \leq n$ and $|y| > 0$

step 3:- we want to find a suitable 'i' so that $xy^iz \notin L$.

The string y can be of any one of the following.

$$L = \{yy / y \in (0,1)^*\}$$

$$\downarrow\downarrow \quad \downarrow\downarrow$$

$$0101$$

pumping Lemma $P = 7$

$S = 0^P10^P1 \Rightarrow 0^710^71$

$S = 00000001000000001$

Case(i) :- The 'y' is in the 1st Part

$S = xyz$

$S = \underbrace{000000}_{x}\underbrace{0}_{y}\underbrace{0100000001}_{z}$

$xy^iz \Rightarrow xy^2z$

$S = 000000000000100000001$

$12 \neq 8$

$|xy| \leq 7 \quad |y| > 0$

$6 \leq 7 \quad 4 > 0$

$\therefore xy^iz \notin L$

Case(ii) :- The 'y' is in the 2nd Part

$S = \underbrace{00000001}_{x}\underbrace{00000001}_{y}\underbrace{}_{z}$

$xy^iz \Rightarrow xy^2z$

$S = 0000000000000000001$

$8 \neq 12$

$\therefore xy^iz \notin L$

$|xy| \leq 7 \quad |y| > 0$

$14 \leq 7 \quad 4 > 0$

Case(iii) :- The 'y' is in the 1 & 2 Parts

$S = \underbrace{00000001}_{x}\underbrace{00000001}_{y}\underbrace{}_{z}$

$xy^iz \Rightarrow xy^2z$

$S = 0000000100010000001$

$\therefore xy^iz \notin L$

$|xy| \leq 7 \quad |y| > 0$

$10 \leq 7 \quad 4 > 0$

So, $L = \{yy \mid y \in (0,1)^*\}$ is not regular

(3) show that $L = \{a^{i^2} \mid i \geq 1\}$ is not regular. (12)

step1: Assume the set L is regular. Let n be the number of states of the FA accepting the set L.

step2: Let $w = a^{n^2}$. $|w| = n^2$ which is greater than n, the number of states of the FA accepting L. By using the pumping lemma, we can write $w = xyz$ with $|xy| \leq n$ and $|y| > 0$.

step3: Take $i = 2$. So, the string will become xy^2z.

$$L = a^{i^2}$$

$$a^{i^2} = a^{1^2}, a^{2^2}, a^{3^2}, a^{4^2}, a^{5^2} \ldots$$

$$= a, a^4, a^9, a^{16}, a^{25} \ldots$$

$$L = a^4 = |aaaa| = 4$$

Pumping Lemma $P = 4$

$$s = aaaa$$

$$s = xyz \in L$$

$$s = \underset{x}{a}\underset{y}{aa}\underset{z}{a}$$

$$xy^iz \in L$$

$i = 2$, $xy^2z = aaaaaa$

$$|xy| \leq n$$

$$3 \leq 4$$

$$|y| > 0$$

$$2 > 0$$

~~So~~, $\therefore xy^2z \notin L$. This is a contradiction.

So, $L = \{a^{i^2} \mid i \geq 1\}$ is not regular.

④ show that $L = \{a^P \mid P \text{ is prime}\}$ is not regular.

Sol:

Step1: Assume that the set L is regular. Let n be the number of states in the FA accepting L.

Step2: Let P be a prime number which is greater than n. Let the string $w = a^P$, $w \in L$. By using the pumping lemma, we can write $w = xyz$ with $|xy| \leq n$ and $|y| > 0$. As the string w consists of only 'a', x, y and z are also a string of 'a's.

Step3: Let us take $i = P+1$.

$L = a^P$

$P = 1, 2, 3, 5, 7, 11, 13, \ldots$

$S = |xyz| = P$

$|xy^iz| = ?$

$i = P+1, \quad |xy^{P+1}z| = |xyz| + |y|^P$

$= P + P|y|$

$= P(1+|y|)$.

$\Rightarrow S = a^P$

$S = a^5 = \underbrace{a}_{x}\underbrace{aaa}_{y}\underbrace{a}_{z}$

$xy^iz \notin L$

$|xy| \leq n \Rightarrow 4 \leq 5$

$|y| > 0 \Rightarrow 3 > 0$

$i = 2, \quad xy^2z = a(aaa)^2a$

$= aaaaaaaaa = 8$

$xy^2z \notin L$

$P(1+|y|)$ is not a prime number as it has factors P and $(1+|y|)$ including 1 and $P(1+|y|)$.

So, $xy^iz \notin L$. This is a contradiction.

$\therefore L = \{a^P \mid P \text{ is prime}\}$ is not regular.

⑤ show that L = Palindrome over $\{a, b\}$ is not regular.

Sol:

Step1: Assume that the set L is regular. Let n be the number of states of the FA accepting L.

Step2: Let $w = a^n b a^n$, By the Pumping Lemma, we can write $w = xyz$ with $|xy| \leq n$ and $|y| > 0$.

step3: we want to find a suitable i so that $xy^i z \notin L$. The string y may consist of only 'a',

$L = a^n b a^n$

Pumping Lemma $P = 4$

$S = a^4 b a^4$

$S = xyz \in L$

$S = \underbrace{aaaa}_{x}\underbrace{a}_{y}\underbrace{baaaa}_{z}$

$xy^i z \in L$

$i = 2, \; xy^2z = aaaaaaabaaaa$

$7 \neq 4$

$xy^2z \notin L$

$|xy| \leq n$

$4 \leq 4$

$|y| > 0$

So, $xy^iz \notin L$. This is contradiction.

So, L = Palindrome over $\{a,b\}$ is not regular.

Closure Properties of Regular Set

A set is closed (under an operation) if and only if the operation on two elements of the set produces another element of the set. If an element outside the set is produced, then the operation is not closed.

—> closure is a property which describes when we combine any two elements of the set, the result is also included in the set.

—> If we multiply two integer numbers, we will get another integer number. Since this process is always true, it is said that the numb integer numbers are 'closed under

the operation of multiplication'. There is simply no way to escape the set of integer numbers when multiplying.

Let $S = \{1,2,3,4,5,6,7,8,9,10\ldots\}$ is a set of integer numbers.

$1 \times 2 = 2$

$2 \times 3 = 6$

$5 \times 2 = 10$

All are included in the set of integer numbers.

→ we can conclude that integer numbers are closed under the operation of multiplication.

Theorem: Two REs L_1 and L_2 over Σ are closed under union operation.

Proof:- we have to prove that if L_1 and L_2 are regular over Σ, then their union, i.e., $L_1 \cup L_2$ will be also regular. As L_1 and L_2 are regular over Σ, there must exist FA $M_1 = (Q_1, \Sigma, \delta_1, q_{01}, F_1)$ and $M_2 = (Q_2, \Sigma, \delta_2, q_{02}, F_2)$ such that $L_1 \in M_1$ and $L_2 \in M_2$

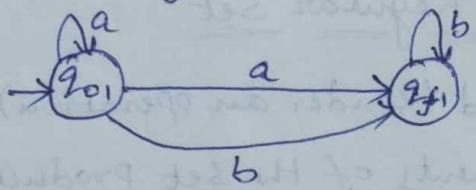
Example:

Let $L_1 = a^*(a+b)b^*$

$L_2 = ab(a+b)^*$

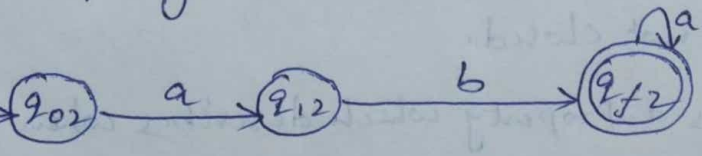
The FA M_1 accepting L_1 is as shown in figure(a) below

(a)



The FA M_2 accepting L_2 is as shown in figure(b) below

(b)



The machine M produced by combining M_1 and M_2 is as shown figure(c) below

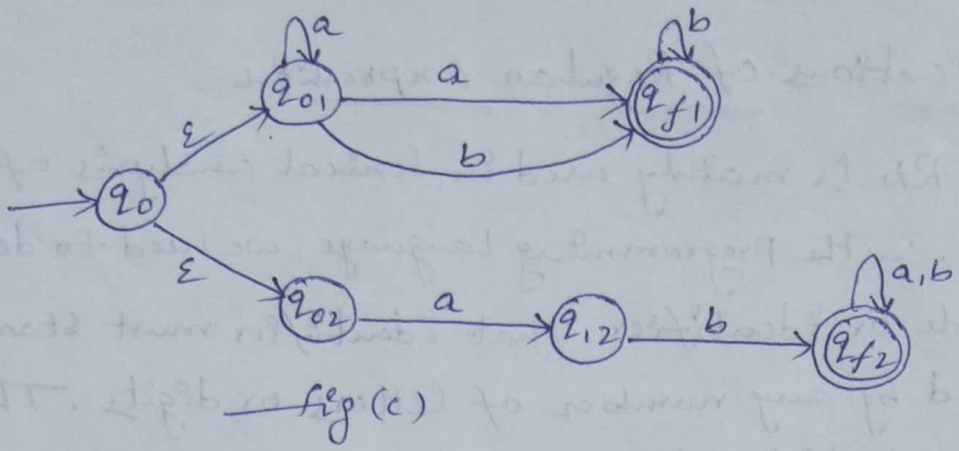


fig (c)

It accepts $L_1 \cup L_2$.

Decision Problems of Regular Expression

Decision Problems are the Problems which can be answered in 'yes' or 'no'. FA are one type of finite state machine which contain a finite number of memory elements. Thus, it can memorize only finite amount of information. FA can answer to those decision problems which require only a finite amount of memory.

The decision problems related to RE are

1. whether a string x belongs to a regular expression R? ($x \in R$?)
2. whether the language set of an FA M is empty ? [$L(M) = \emptyset$?]
3. whether the language set of an FA M is finite ? [$L(M)$ is finite ?]
4. whether there exists any string accepted by two FA, M_1 and M_2 ?
5. whether the language set of an FA M_1 is a subset of the language set of another FA M_2 ? [$L(M_1) \subseteq L(M_2)$?]
6. whether two FA M_1 and M_2 accept the same language ? [$L(M_1) = L(M_2)$]
7. whether two REs R_1 and R_2 are from the same language set ?
8. whether an FA is the minimum state FA for a language L ?

Applications of Regular Expression

RE is mainly used in lexical analysis of compiler design. In the Programming language, we need to declare a variable or identifier. That identifier must start with a letter followed by any number of letters or digits. The structure of the identifier is represented by RE.

The definition of an identifier in a programming language is

$$\text{Letter} \longrightarrow A|B|\ldots|Z|a|b|\ldots|z$$
$$\text{digit} \longrightarrow 0|1|\ldots|9|$$
$$\text{id} \longrightarrow \text{letter}(\text{letter}|\text{digit})^*$$

The definition of an unsigned number in a programming language is

$$\text{digit} \longrightarrow 0|1|\ldots|9$$
$$\text{digits} \longrightarrow \text{digit}^+$$
$$\text{opt-fraction} \longrightarrow (.\,\text{digits})^*$$
$$\text{opt-exponent} \longrightarrow (E(+|-)^*\,\text{digits})^*$$
$$\text{unsigned-num} \longrightarrow \text{digits opt-fraction opt-exponent}$$

The other application of RE is Pattern searching from a given text.