CHAPTER -1

Q- 1 Properties of Regular expression

Ans - Regular expressions :

- It is a way of representing regular languages.
- The algebraic description for regular languages is done using regular expressions.
- They can define the same language that various forms of finite automata can describe.
- Regular expressions offer something that finite automata do not, i.e. it is a declarative way to express the strings that we want to accept. They act as input for many systems. They are used for string matching in many systems(Java, python etc.)
- For example, Lexical-analyzer generators, such as Lex or Flex.

The widely used operators in regular expressions are Kleene closure(∗) ,concatenation(.) , Union(+).

Rules for regular expressions :

- The set of regular expressions is defined by the following rules.
- Every letter of ∑ can be made into a regular expression, null string, ∈ itself is a regular expression.

If r1 and r2 are regular expressions, then (r1), r1.r2, r1+r2, r1*, r1+ are also regular expressions.

Example – ∑ = {a, b} and r is a regular expression of language made using these symbols

Regular language	**Regular set**
∅	{ }
∈	{∈}
a*	{∈, a, aa, aaa …..}
a+ b	{a, b}
a.b	{ab}
a* + ba	{∈, a, aa, aaa,…… , ba}

Operations performed on regular expressions :
1. Union –
The union of two regular languages, L1 and L2, which are represented using L1 ∪ L2, is also regular and which represents the set of strings that are either in L1 or L2 or both.
Example
L1 = (1+0).(1+0) = {00 , 10, 11, 01} and
L2 = {∈ , 100}
then L1 ∪ L2 = {∈, 00, 10, 11, 01, 100}.
2. Concatenation –
The concatenation of two regular languages, L1 and L2, which are represented using L1.L2 is also regular and which represents the set of strings that are formed by taking any string in L1 concatenating it with any string in L2.
Example –
L1 = { 0,1 } and L2 = { 00, 11} then L1.L2 = {000, 011, 100, 111}.
3. Kleene closure –
If L1 is a regular language, then the Kleene closure i.e. L1* of L1 is also regular and represents the set of those strings which are formed by taking a number of strings from L1 and the same string can be repeated any number of times and concatenating those strings.
Example –
L1 = { 0,1} = {∈, 0, 1, 00, 01, 10, 11 …….} , then L* is all strings possible with symbols 0 and 1 including a null string.
Algebraic properties of regular expressions :
Kleene closure is an unary operator and Union(+) and concatenation operator(.) are binary operators.
1. Closure –
If r1 and r2 are regular expressions(RE), then

- r1* is a RE
- r1+r2 is a RE
- r1.r2 is a RE

2. Closure laws –

- (r*)* = r, closing an expression that is already closed does not change the language.

- $\varnothing^* = \in$, a string formed by concatenating any number of copies of an empty string is empty itself.
- $r_+ = r.r^* = r^*r$, as $r^* = \in + r + rr + rrr$ …. and $r.r^* = r + rr + rrr$ ……
- $r^* = r^* + \in$

3. Associativity –
If r1, r2, r3 are RE, then
i.) r1+ (r2+r3) = (r1+r2) +r3
- For example : r1 = a , r2 = b , r3 = c, then
- The resultant regular expression in LHS becomes a+(b+ c) and the regular set for the corresponding RE is {a, b, c}.
- for the RE in RHS becomes (a+ b) + c and the regular set for this RE is {a, b, c}, which is same in both cases. Therefore, the associativity property holds for union operator.

ii.) r1.(r2.r3) = (r1.r2).r3
- For example – r1 = a , r2 = b , r3 = c
- Then the string accepted by RE a.(b.c) is only abc.
- The string accepted by RE in RHS is (a.b).c is only abc ,which is same in both cases. Therefore, the associativity property holds for concatenation operator.

Associativity property does not hold for Kleene closure(*) because it is unary operator.

4. Identity –
In the case of union operators
if $r + x = r \Rightarrow x = \varnothing$ as $r \cup \varnothing = r$, therefore $\varnothing$ is the identity for +.
Therefore, $\varnothing$ is the identity element for a union operator.
In the case of concatenation operator –
if r.x = r , for $x = \in$
$r.\in = r \Rightarrow \in$ is the identity element for concatenation operator(.) .
5. Annihilator –
- If $r + x = r \Rightarrow r \cup x = x$, there is no annihilator for +

- In the case of a concatenation operator, r.x = x, when x = ∅, then r.∅ = ∅, therefore ∅ is the annihilator for the (.)operator. For example {a, aa, ab}.{ } = { }

6. Commutative property –
If r1, r2 are RE, then
- r1+r2 = r2+r1. For example, for r1 =a and r2 =b, then RE a+ b and b+ a are equal.
- r1.r2 ≠ r2.r1. For example, for r1 = a and r2 = b, then RE a.b is not equal to b.a.

7. Distributed property –
If r1, r2, r3 are regular expressions, then
- (r1+r2).r3 = r1.r3 + r2.r3 i.e. Right distribution
- r1.(r2+ r3) = r1.r2 + r1.r3 i.e. left distribution
- (r1.r2) +r3 ≠ (r1+r3)(r2+r3)

8. Idempotent law –
- r1 + r1 = r1 ⇒ r1 ∪ r1 = r1 , therefore the union operator satisfies idempotent property.
- r.r ≠ r ⇒ concatenation operator does not satisfy idempotent property.

9. Identities for regular expression –
There are many identities for the regular expression. Let p, q and r are regular expressions.
- ∅ + r = r
- ∅.r= r.∅ = ∅
- ∈.r = r.∈ =r
- ∈* = ∈ and ∅* = ∈
- r + r = r
- r*.r* = r*
- r.r* = r*.r = r^+.
- (r*)* = r*

- $\in + r.r^* = r^* = \in + r.r^*$
- $(p.q)^*.p = p.(q.p)^*$
- $(p + q)^* = (p^*.q^*)^* = (p^* + q^*)^*$
- $(p+ q).r= p.r+ q.r$ and $r.(p+q) = r.p + r.q$

Q-2 Convert dfa to regular expression

Ans- 2015 wale m h or rgpv m h imp h

Q-3 Pumping lemma for regular lamguage

Ans- Pumping Lemma for Regular Languages
For any regular language L, there exists an integer n, such that for all $x \in L$ with $|x| \geq n$, there exists $u, v, w \in \Sigma^*$, such that $x = uvw$, and
(1) $|uv| \leq n$
(2) $|v| \geq 1$
(3) for all $i \geq 0$: $uv^iw \in L$

In simple terms, this means that if a string v is 'pumped', i.e., if v is inserted any number of times, the resultant string still remains in L.

Pumping Lemma is used as a proof for irregularity of a language. Thus, if a language is regular, it always satisfies pumping lemma. If there exists at least one string made from pumping which is not in L, then L is surely not regular. The opposite of this may not always be true. That is, if Pumping Lemma holds, it does not mean that the language is regular.
https://media.geeksforgeeks.org/wp-content/cdn-uploads/gq/2016/03/p1.png
For example, let us prove $L_{01} = \{0^n1^n \mid n \geq 0\}$ is irregular.
Let us assume that L is regular, then by Pumping Lemma the above given rules follow.
Now, let $x \in L$ and $|x| \geq n$. So, by Pumping Lemma, there exists u, v, w such that (1) – (3) hold.

We show that for all u, v, w, (1) – (3) does not hold.
If (1) and (2) hold then $x = 0^n1^n = uvw$ with $|uv| \leq n$ and $|v| \geq 1$.
So, $u = 0^a$, $v = 0^b$, $w = 0^c1^n$ where : $a + b \leq n$, $b \geq 1$, $c \geq 0$, $a + b + c = n$
But, then (3) fails for $i = 0$
$uv^0w = uw = 0^a0^c1^n = 0^{a + c}1^n \notin L$, since $a + c \neq n$.
https://media.geeksforgeeks.org/wp-content/cdn-uploads/gq/2016/03/p2.png

Q-4 explain dfa and nfa with example

Ans- DFA refers to deterministic finite automata. Deterministic refers to the uniqueness of the computation. The finite automata are called deterministic finite automata if the machine is read an input string one symbol at a time.

- ❍In DFA, there is only one path for specific input from the current state to the next state.
- ❍DFA does not accept the null move, i.e., the DFA cannot change state without any input character.
- ❍DFA can contain multiple final states. It is used in Lexical Analysis in Compiler.

In the following diagram, we can see that from state q0 for input a, there is only one path which is going to q1. Similarly, from q0, there is only one path for input b going to q2.

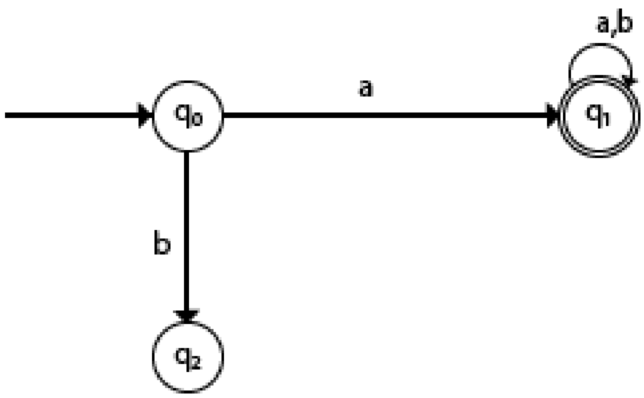


Fig:- DFA

Formal Definition of DFA

A DFA is a collection of 5-tuples same as we described in the definition of FA.

1. Q: finite set of states
2. ∑: finite set of the input symbol
3. q0: initial state
4. F: **final** state
5. δ: Transition function

   Transition function can be defined as:

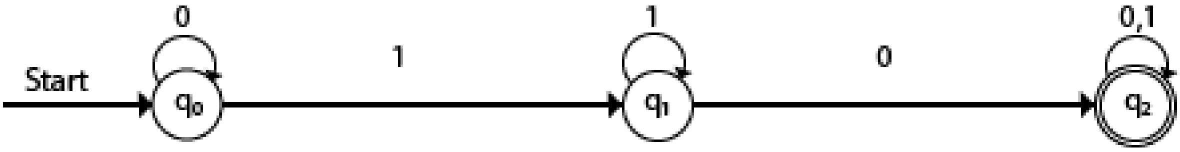
1. δ: Q x ∑→Q

Example 1:

1. Q = {q0, q1, q2}
2. ∑ = {0, 1}
3. q0 = {q0}
4. F = {q2}

Solution:

Transition Diagram:



Transition Table:

Present State	Next state for Input 0	Next State of Input 1
→q0	q0	q1
q1	q2	q1
*q2	q2	q2

NFA (Non-Deterministic finite automata)

❍NFA stands for non-deterministic finite automata. It is easy to construct an NFA than DFA for a given regular language.

❍The finite automata are called NFA when there exist many paths for specific input from the current state to the next state.

❍Every NFA is not DFA, but each NFA can be translated into DFA.

❍NFA is defined in the same way as DFA but with the following two exceptions, it contains multiple next states, and it contains ε transition.

In the following image, we can see that from state q0 for input a, there are two next states q1 and q2, similarly, from q0 for input b, the next states are q0 and q1. Thus it is not fixed or determined that with a particular input where to go next. Hence this FA is called non-deterministic finite automata.

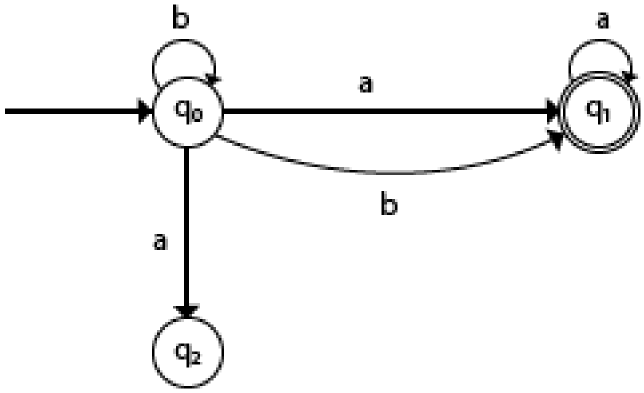


Fig:- NDFA

Formal definition of NFA:

NFA also has five states same as DFA, but with different transition function, as shown follows:

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

where,

1. Q: finite set of states
2. ∑: finite set of the input symbol
3. q0: initial state
4. F: **final** state
5. δ: Transition function

Example 1:

1. Q = {q0, q1, q2}
2. ∑ = {0, 1}
3. q0 = {q0}
4. F = {q2}

Solution:

Transition diagram:

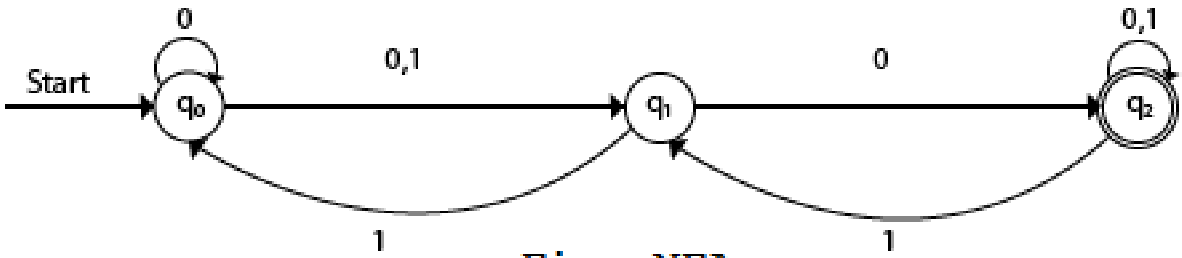


Fig: NFA

Transition Table:

Present State	Next state for Input 0	Next State of Input 1
→q0	q0, q1	q1
q1	q2	q0
*q2	q2	q1, q2

In the above diagram, we can see that when the current state is q0, on input 0, the next state will be q0 or q1, and on 1 input the next state will be q1. When the current state is q1, on input 0 the next state will be q2 and on 1 input, the next state will be q0. When the current state is q2, on 0 input the next state is q2, and on 1 input the next state will be q1 or q2.

Q-5 How dfa equivalence to nfa

Ans -

Q- 6 Difference between mealy and moore machine

Ans -

Difference Between Moore and Mealy Machine

Moore Machine –

- Output depends only upon present state.
- If input changes, output does not change.
- More number of states are required.
- There is less hardware requirement for circuit implementation.
- They react faster to inputs.
- Synchronous output and state generation.
- Output is placed on states.
- Easy to design.

Mealy Machine –

- Output depends on present state as well as present input.
- If input changes, output also changes.
- Less number of states are required.
- There is more hardware requirement for circuit implementation.
- They react slower to inputs(One clock cycle later).
- Asynchronous output generation.
- Output is placed on transitions.
- It is difficult to design.

Q-7 Mylle nerode theorem

Ans- Copy m se dekh lena

Q-8 application of pumping lemma

Ans - Applications of Pumping Lemma

Pumping Lemma is to be applied to show that certain languages are not regular. It should never be used to show a language is regular.

•If **L** is regular, it satisfies Pumping Lemma.

•If **L** does not satisfy Pumping Lemma, it is non-regular.

chapter – 2

Q-1 CFL generated by grammer

Ans – dekhna h kese krna h

Q-2 Regular grammmer

Ans – Copy m h

Q-3 CNF (CHOMSKY NORMAL FORM

ans - Chomsky's Normal Form (CNF)

CNF stands for Chomsky normal form. A CFG(context free grammar) is in CNF(Chomsky normal form) if all production rules satisfy one of the following conditions:

❍Start symbol generating ε. For example, A → ε.

❍A non-terminal generating two non-terminals. For example, S → AB.

❍A non-terminal generating a terminal. For example, S → a.

For example:

1. G1 = {S → AB, S → c, A → a, B → b}
2. G2 = {S → aA, A → a, B → c}

The production rules of Grammar G1 satisfy the rules specified for CNF, so the grammar G1 is in CNF. However, the production rule of Grammar G2 does not satisfy the rules specified for CNF as S → aZ contains terminal followed by non-terminal. So the grammar G2 is not in CNF.

Steps for converting CFG into CNF

Step 1: Eliminate start symbol from the RHS. If the start symbol T is at the right-hand side of any production, create a new production as:

1. S1 → S

Where S1 is the new start symbol.

Step 2: In the grammar, remove the null, unit and useless productions. You can refer to the Simplification of CFG

.

Step 3: Eliminate terminals from the RHS of the production if they exist with other non-terminals or terminals. For example, production S → aA can be decomposed as:

1. S → RA
2. R → a

Step 4: Eliminate RHS with more than two non-terminals. For example, S → ASB can be decomposed as:

1. S → RS
2. R → AS