Basics of Regular Expression

- There are following different type of characters of a regular expression:

Anchors:

1. The caret (^) Symbol:

- The caret symbol tells the computer that the match must start at the beginning of the string or line.
- Ex: ^The- Matches any string that starts with The.

2. The Dollar ($) Symbol:

- It tells the computer that the match must occur at the end of the string or line.
- Ex: End$- Matches a string that ends with end

2. Quantifiers:

- The quantifiers are used in the regular expression for specifying the number of occurrences of a character.

1. The plus symbol (+):

- It tells the computer to repeat the preceding character (or set of characters) at atleast one or more times(up to infinite).

Ex: abc+ :matches a string that has ab followed by one or more c

2. The asterisk symbol (*):

- It tells the computer to match the preceding character (or set of characters) for 0 or more times (upto infinite).

Ex: abc* : matches a string that has ab followed by zero or more c

3. The curly braces { … }

It tells the computer to repeat the preceding character (or set of characters) for as many times as the value inside this bracket.

Ex: abc{2} : matches a string that has ab followed by 2 c.

4. Optional character (?)

This symbol tells the computer that the preceding character may or may not be present in the string to be matched.

Ex: We may write the format for document file as – "docx?", The '?' tells the computer that x may or may not be present in the name of file format.

5. Wildcard (.)
The dot symbol can take the place of any other symbol, that is why it is called the wildcard character.
Example : The Regular expression .* will tell the computer that any character can be used any number of times.
---
3. OR Operator- | or [ ]

1. It is used to match a particular character or a group of characters on either side. If the character on the left side is matched, then the right side's character is ignored.

Ex: a(b|c): Matches a string that has a followed by b or c.

2. It is used to match any character from a range of characters defined in the square bracket.
Ex: **xz[atp]r** is an expression which matches with the following strings: **"xzar", "xztr", and "xzpr“**

---

4. Character Classes:

\s: It is used to matches white space character.
\S: It is used to matches non-white space character.
\0: It is used to match a NULL character.
\d: It is used to match one decimal digit, which means from 0 to 9.
\D: It is used to match any non-decimal digit.
\n : It is used to match new line.
\w: It is used to match the alphanumeric [0-9, a-z, A-Z] characters.
\W: It is used to match non-word character

Finite Automata

Finite Automata(FA) is the simplest machine to recognize patterns. It is used to characterize a Regular Language.

- It takes the string of symbol as input and changes its state accordingly. When the desired symbol is found, then the transition occurs.
- At the time of transition, the automata can either move to the next state or stay in the same state.
- Finite automata have two states, **Accept state** or **Reject state**. When the input string is processed successfully, and the automata reached its final state, then it will accept.

A finite automaton is a collection of 5-tuple (Q, ∑, δ, q0, F), where:

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

Types of Automata:

There are two types of finite automata:

1. DFA(deterministic finite automata)
2. NFA(non-deterministic finite automata)

DFA(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.

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:

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

Graphical Representation of DFA

A DFA can be represented by digraphs called state diagram. In which:

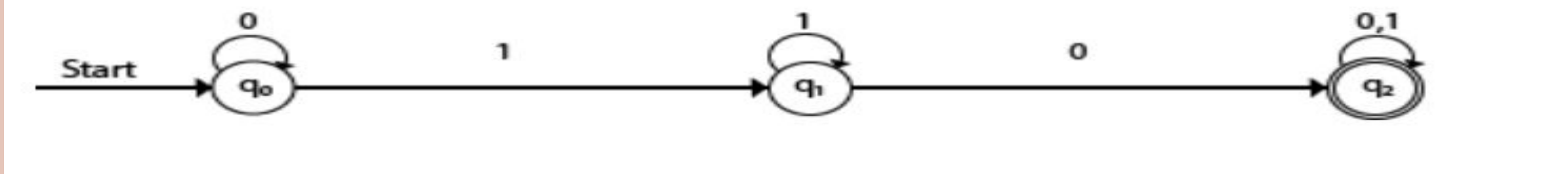
1. The state is represented by vertices.
2. The arc labeled with an input character show the transitions.
3. The initial state is marked with an arrow.
4. The final state is denoted by a double circle.

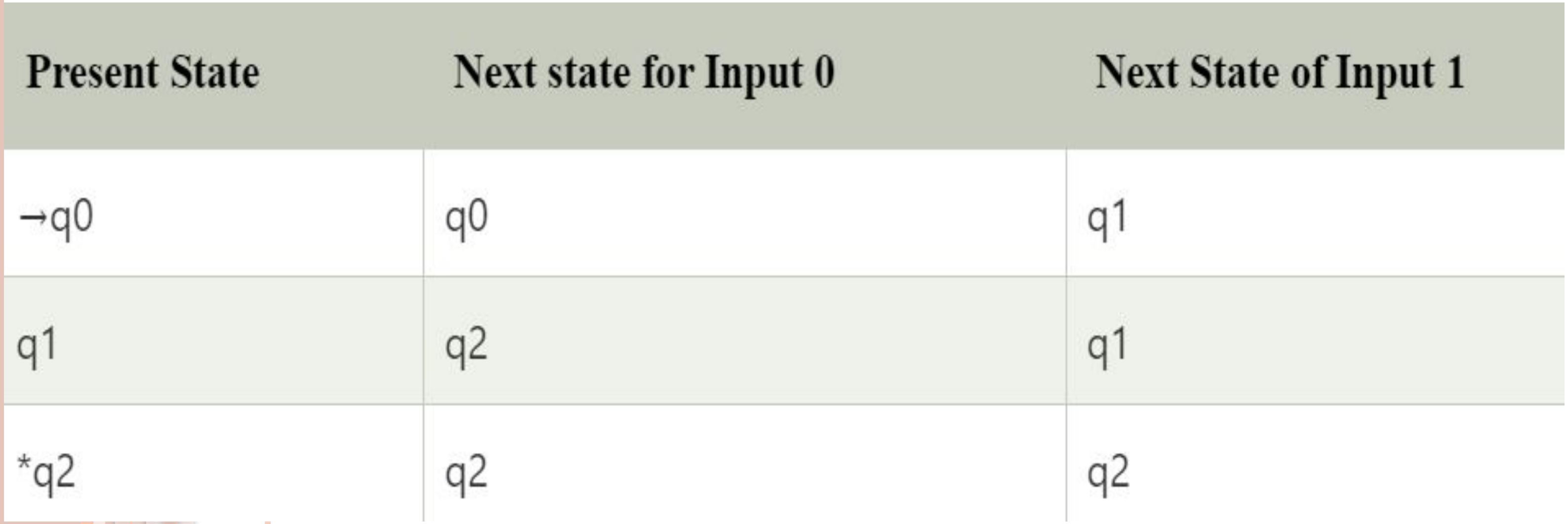
Example 1:

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

Solution:

Transition Diagram:



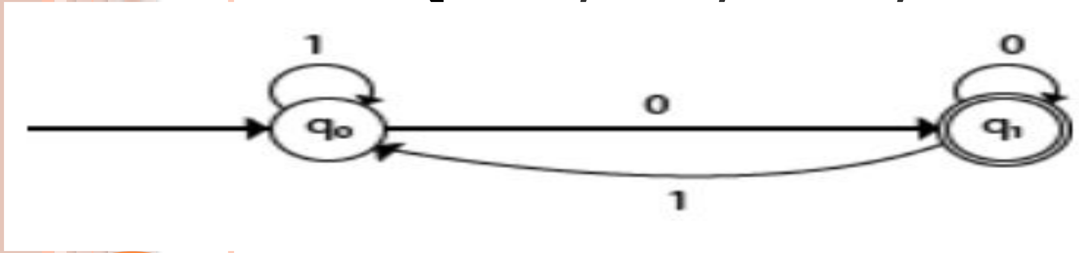


Present State	Next state for Input 0	Next State of Input 1
→q0	q0	q1
q1	q2	q1
*q2	q2	q2

Example 2:

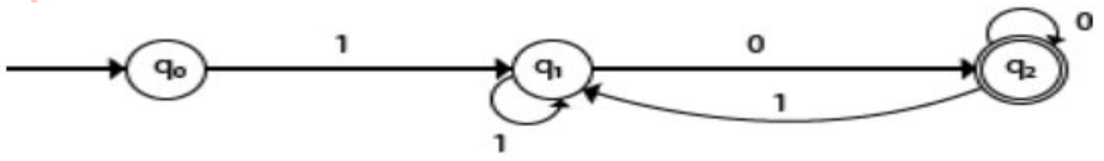
DFA with ∑ = {0, 1} accepts all ending with 0.

Solution: L={ 00, 10, 110, 100..}


$$\begin{tikzcd}
\arrow[r] & q_0 \arrow[loop above, "1"] \arrow[rrr, "0"] & & & q_1 \arrow[loop above, "0"] \arrow[lll, bend left, "1"]
\end{tikzcd}$$

Example 3:

Design a FA with ∑ = {0, 1} accepts those string which starts with 1 and ends with 0.


$$\begin{tikzcd}
\arrow[r] & q_0 \arrow[rr, "1"] & & q_1 \arrow[loop below, "1"] \arrow[rr, "0"] & & q_2 \arrow[loop above, "0"] \arrow[ll, bend left, "1"]
\end{tikzcd}$$

NFA (Non-Deterministic finite automata)

- 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.
- NFA also has five states same as DFA, but with different transition function, as shown follows:
- $\delta: Q \times \Sigma \rightarrow 2^Q$

Example 1:

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

Solution:

Transition diagram:

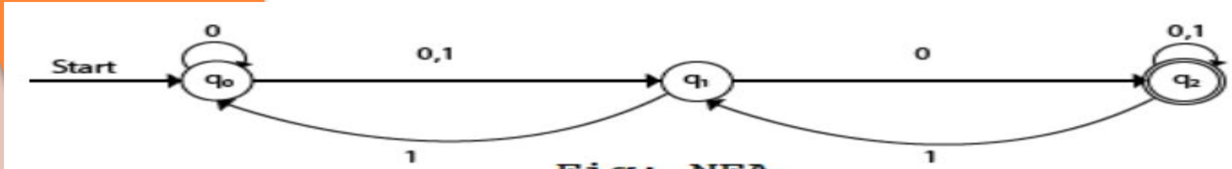
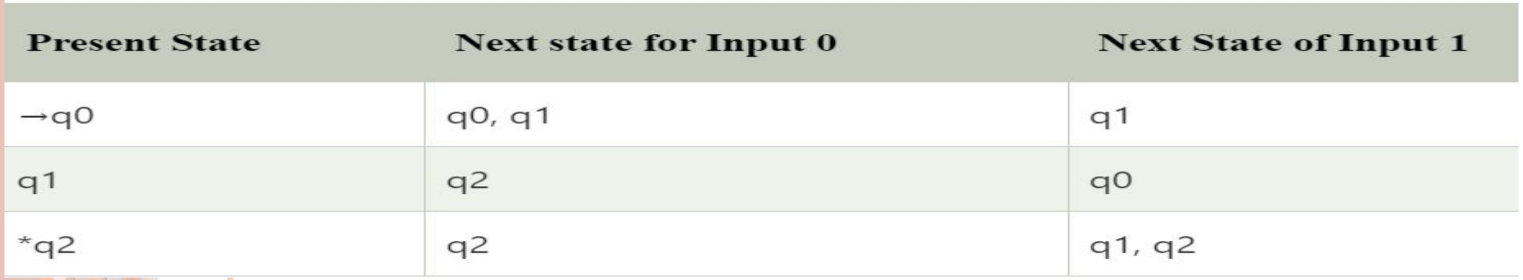


Fig: NFA



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

Example 2:
NFA with ∑ = {0, 1} accepts all strings with 01.

Solution:

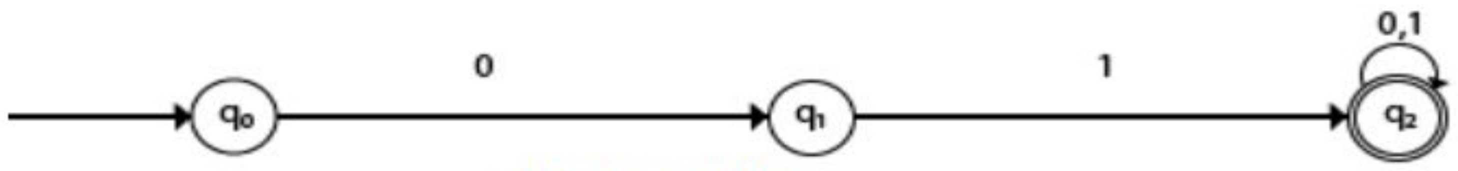


Fig: NFA

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