2.3 SPECIFICATION OF TOKENS (REGULAR EXPRESSIONS)

- To specify tokens, Regular Expressions are used.
- It provides convenient and useful notation for representing tokens.
- Regular Expressions define the language accepted by finite Autom (Transition Diagram).
- Regular Expressions are defined over an alphabet Σ.
- If R is a Regular Expression, then L(R) represents language denoted regular expression.

Language : It is a collection of strings over some fixed alphabet. Empty str can be denoted by ε.

E.g. if L (Language) = set of strings of 0's & 1's of length two

then L = {00, 01, 10, 11}

E.g. → If $L = \{1\}$

then $L^* = L^0 \cup L^1 \cup L^2 \cup \ldots\ldots$ Here * can be 0, 1, 2, 3,............

$\therefore$ $L^* = \{\varepsilon\} \cup \{1\} \cup \{11\} \cup \ldots\ldots$

$\therefore$ $L^* = \{\varepsilon, 1, 11, 111, \ldots\ldots\ldots\}$

Operations on Languages :

If $L_1 = \{00, 10\}$ & $L_2 = \{01, 11\}$

Operation	Description	Example
Union	$L_1 \cup L_2$ = {set of string in L_1 & strings in L_2	$L_1 \cup L_2 = \{00, 10, 01, 11\}$
Concatenation	L_1L_2 = {Set of string in L_1 followed by strings in L_2}	$L_1L_2 = \{0001, 0011, 1001, 1011\}$
Kleen closure of L_1 L_1^*	$L_1^* = L_1^0 \cup L_1^1 \cup L_1^2 \cup \ldots\ldots$ $L_1^* = \bigcup_{i=0}^{\infty} L_1^i$	$L_1^* = \{\varepsilon, 00, 10, 1010, 0010, 1000, 0000, 000000, 001000, \ldots\}$
Positive Closure L_1^+	$L_1^+ = L_1^1 \cup L_1^2 \cup \ldots\ldots$ $L_1^+ = \bigcup_{i=1}^{\infty} L_1^i$	$L_1^+ = \{00, 10, 1010, 0010, 1000, 0000, 000000, 001000, \ldots\ldots\}$

Rules of Regular Expressions

1. ε is a Regular expression.
2. Union of two Regular Expressions R_1 and R_2.
   i.e. $R_1 + R_2$ or $R_1 \mid R_2$ is also a Regular Expression.
3. Concatenation of two Regular Expressions R_1 and R_2
   i.e. R_1R_2 is also a Regular Expression.
4. Closure of Regular Expression R *i.e* , R^* is also a Regular Expression.
5. If R is Regular Expression then (R) is also a Regular Expression.

Algebric Laws

1. $R_1 \mid R_2 = R_2 \mid R_1$ or $R_1 + R_2 = R_2 + R_1$ (Commutative)
2. $R_1 \mid (R_2 \mid R_3) = (R_1 \mid R_2) \mid R_3$ (Associative)

or $R_1 + (R_2 + R_3) = (R_1 + R_2) + R_3$

3. $R_1 (R_2 R_3) = (R_1R_2) R_3$ (Associati...)

4. $R_1 (R_2 \mid R_3) = R_1 R_2 \mid R_1R_3$ (Distributi...)

or $R_1 (R_2 + R_3) = R_1R_2 + R_1R_3$

5. $\varepsilon R = R \varepsilon = R$ (Concatenati...)

Example 2: *Find Regular Expression for following Language.*

(a) $L = \{\varepsilon, 1, 11, 111, \ldots\ldots\}$

{**Hint :** * = 0, 1, 2, 3, ...} $\{\therefore 1^0 = \varepsilon, 1^1 = 1, 1^2 = 11, 1^3 = 111, \ldots$

Ans. 1^*

(b) $L = \{\varepsilon, 11, 1111, 111111, \ldots\ldots\}$

Ans. $(11)^*$

(c) L = *Set of all strings of 0's and 1's* = $\{\varepsilon, 0, 1, 01, 11, 00, 000, 101, \ldots\}$

Ans. $(0 + 1)^*$ or $(0 \mid 1)^*$

(d) L = *Set of all strings of 0's and 1's ending with 11.*

Ans. $(0 + 1)^*11$

(e) L = *Set of all strings of 0's and 1's beginning with 0 and Ending with 1.*

Ans. $0 (0 + 1)^*1$

Example 3 : *Write Regular Expressions for following language over* $\Sigma = \{a, b\}$

(a) *Strings of length zero or one.*

Ans. $\varepsilon \mid a \mid b$ or $(\varepsilon + a + b)$

(b) *Strings of length two.*

Ans. $aa \mid ab \mid ba \mid bb$ or $(aa + ab + ba + bb)$

(c) *Strings of Even length.*

Ans. $(aa \mid ab \mid ba \mid bb)^*$ or $(ac + ab + ba + bb)^*$

(d) *Set of all strings of a's and b's having atleast two occurrences of aa.*

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

Example 4 : *Write Regular Expressions for all types of Tokens.*

Ans. (a) Keywords = begin | end | if | then | else

(b) Identifier =letter (letter + digit)*

(c) Constant = digit digit* = digit^+

(d) Relation-op = < | > | <= | >= | <>

Example 5. *Write Regular Expression in which second letter from right end of stri... is 1 where* $\Sigma = \{0,1\}$.

Sol. $(0 + 1)^*1\,(0 + 1)$

Example 6. *Write Regular Expression for* $\Sigma = \{a, b\}$

(a) L = *set of strings having atleast one occurrence of double letter*

Ans. $(a + b)^*\ (aa + bb)\ (a + b)^*$

(b) L = *set of strings having double letter at Beginning and Ending of string.*

Ans. $(aa + bb)\ (a + b)^*\ (aa + bb)$

(c) L = *set of string having double letter at beginning or on ending of string.*

Ans. $(aa + bb)\ (a + b)^* + (a + b)^*\ (aa + bb) + (aa + bb)\ (a + b)^*\ (aa + bb)$.

Example 7. *Write* regular expression for set of words having *a, e, i, o, u appearing in the order although not necessarity consecutively.*

Ans. If α = $([b - d] + [f - h] + [j - n] + [p - t] + [v - z]$ + Blankspace$)^*$

$\therefore$ Required Regular Expression will be

$\alpha\ a\ \alpha\ e\ \alpha\ i\ \alpha\ o\ \alpha\ u\ \alpha$

2.4. RECOGNITION OF TOKEN (FINITE AUTOMATA)

Regular Expression are used to represent various type of Tokens. From these Regular Expression, Finite Automata is generated. **Finite Automata is basically other name of Lexical Analyzer** which checks or recognize whether a string is Token or not.

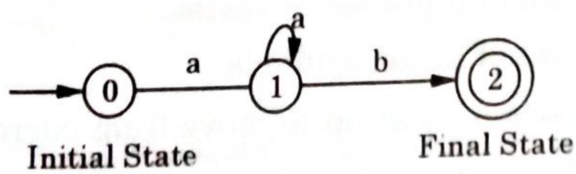
Finite Automata :

- It is a Machine or a Recognizer for a language that is used to check whether string is accepted by a language or not.
- It gives answer "Yes" if string is accepted else "no".
- In **Finite Automata, Finite** means finite number of states & **Automata** means, the Automatic Machine which works without any interference of human being.

Representation of Finite Automata : There are 2 ways to represent finite Automata.

1. **Transition Diagram :** It is a directed graph or flow chart having states & edges.

E.g.



0, 1, 2 → States   0 → Initial state   2 → Final State

a, b → Input symbols.

2. **Transition Table :**

States \ Input	a	b
0	1	–
1	1	2
2	–	–

- **Finite Automata can be Represented by 5 tuple ($Q, \Sigma, \delta, q_0, F$)**

(*a*) Q is finite non-empty set of states.

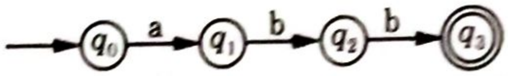
(*b*) Σ is finite set of input symbols .

(*c*) δ is transition function.

(*d*) $q_0 \in Q$ is initial state.

(*e*) $F \subseteq Q$ is the set of final states.

Example 8: *Design Finite Automata which accepts string "abb"*

Ans. 

States	:	$Q = \{q_0, q_1, q_2, q_3\}$
Input symbols	:	$\Sigma = \{a, b\}$
Transition Function δ	:	$\{\delta(q_0, a) = q_1,\ \delta(q_1, b) = q_2, \delta(q_2, b) = q_3\}$
Initial State	:	q_0
Final State (F)	:	$\{q_3\}$

2.4.1 Types of Finite Automata

(*a*) Deterministic Finite Automata (DFA)

(*b*) Non-Deterministic Finite Automata (NFA)

(*a*) Deterministic Finite Automata (DFA) : "Deterministic" means that on each input there is one and only one state to which the automata can be transition from its current state.

- **DFA is a 5-tuple ($\theta, \Sigma, \delta, q_0, F$)**

where (*a*) Q is finite non-empty set of states

(*b*) Σ is finite set of input-symbols

(*c*) δ is a transition function to move from current state to next state

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

(*d*) $q_0 \in Q$ is initial state

(*c*) $F \subseteq Q$ is set of final states

(b) Non-Deterministic Finite Automata (NFA) : "Non-Deterministic" means there can be several possible transitions. So, output is non-deterministic for a given input.

- **NFA is 5-tuple (θ, Σ, δ, q_0, F) where**

(a) Q is finite non-empty set of states.

(b) Σ is finite set of input symbols.

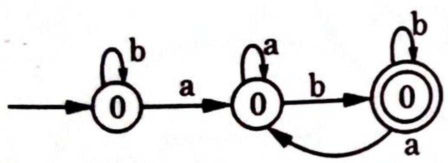
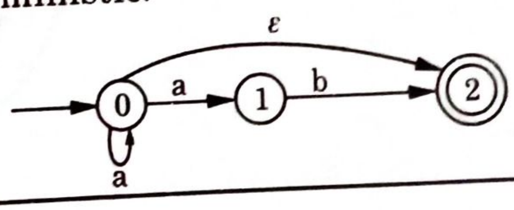
(c) δ is transition function to move from current state to next state.

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

(d) $q_0 \in Q$ is initial state

(e) $F \subseteq Q$ is set of final states.

- **Difference Between DFA and NFA**

	DFA	NFA
1.	Every transition from one state to other is unique & deterministic in nature.	There can be multiple transitions for an input *i.e.* non-deterministic.
2.	Null transitions (ε) are not allowed.	Null Transitions (ε) are allowed means transition form current state to next state without any input.
3.	Transition function $\delta : Q \times \Sigma \rightarrow Q$.	Transition function $\delta : Q \times \Sigma \rightarrow 2^Q$.
4.	Requires less memory as transitions & states are less	Requires more memory.
5.	**E.g** DFA In fig. Each State has unique input symbol Σ = {a, b} outgoing from it. 	E.g NFA In fig. As, there are multiple transitions of input symbol 'a' on state 0. & no transition from state 2, *i.e* non-deterministic. 

2.5 CONVERSION OF REGULAR EXPRESSION TO FINITE AUTOMATA (NFA)

A Regular Expression is basically a representation of Tokens. But Recognize a token, we need token Recognizer which is nothing but a Fi Automata (NFA). So, we can convert Regular Expression into NFA.

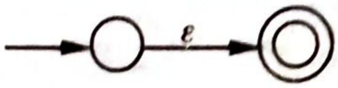
Algorithm for conversion of RE to NFA:

Input : A Regular Expression R

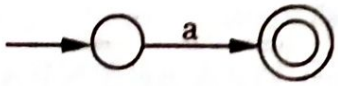
Output : NFA accepting language denoted by R

Method :

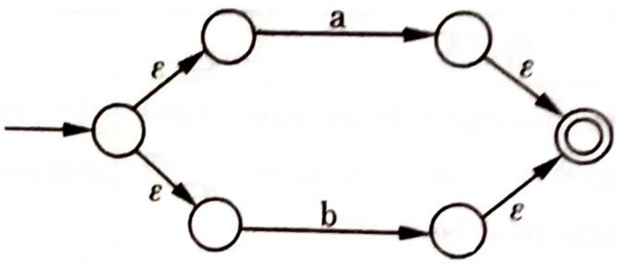
1. For ε, NFA is



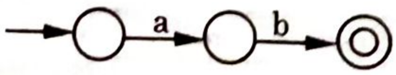
2. For a, NFA is



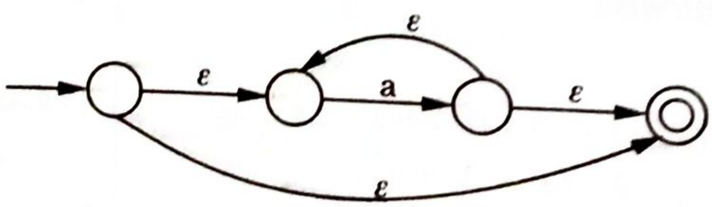
3. For a + b, or a | b NFA is



4. For ab, NFA is

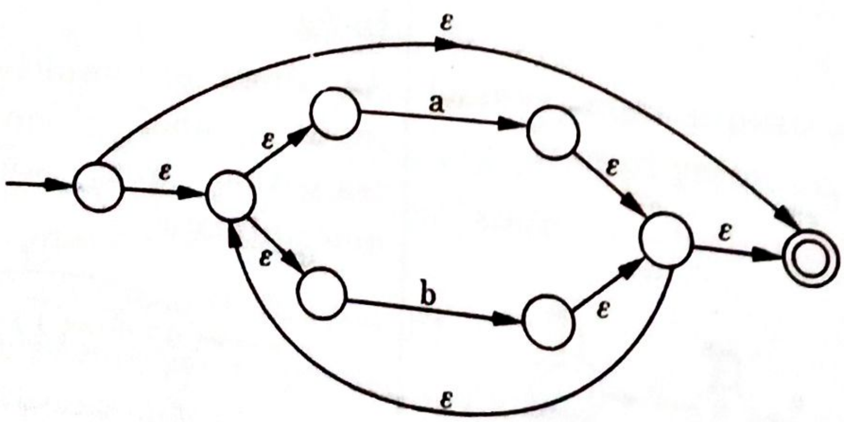


5. For a*, NFA is



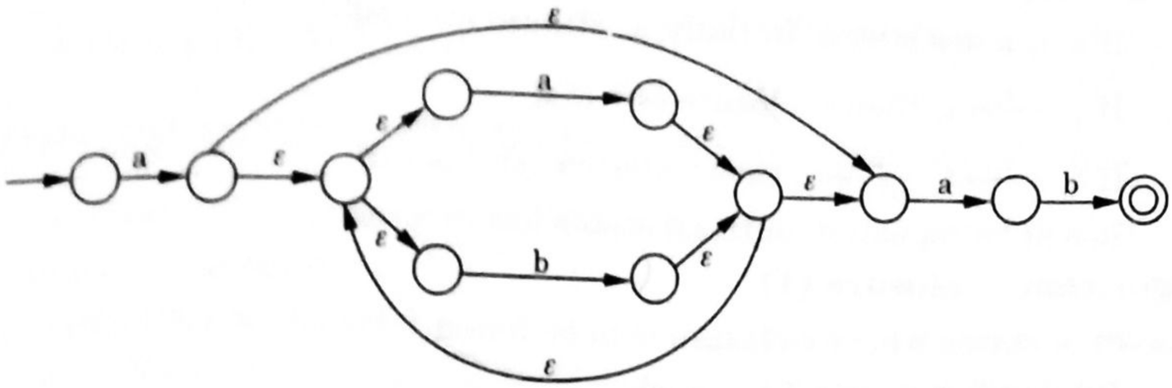
Example 9: *Draw NFA for Regular Expression (a+b)**

Ans.



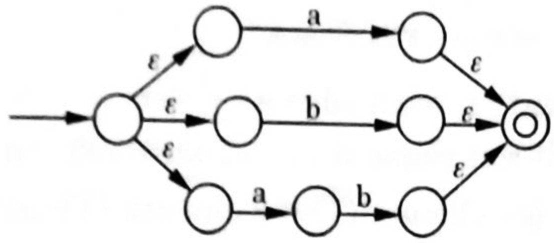
Example 10: *Draw NFA for Regular Expression* **a (a+b)* ab**

Ans.



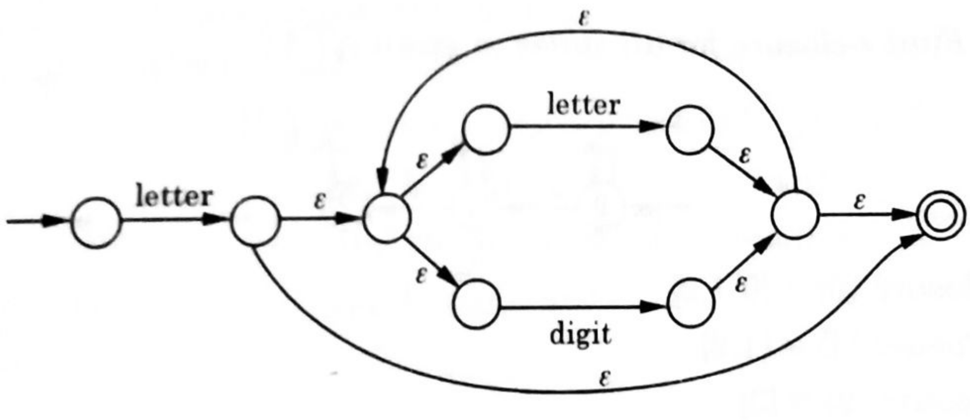
Example 11: *Draw NFA for* ***a+b+ ab***

Ans.



Example 12 : *Draw NFA for* **letter (letter + digit)***

Ans.



Example 13: *Draw NFA corresponding to* ***(0+1)* 1 (0+1)***

Ans.