

-----MODULE-1-----

Introduction to Automata Theory: Central Concepts of Automata theory, Deterministic Finite Automata (DFA), Non- Deterministic Finite Automata(NFA) ,Epsilon- NFA, NFA to DFA Conversion, Minimization of DFA

Introduction to Compiler Design: Language Processors, Phases of Compilers

INTRODUCTION TO FINITE AUTOMATA

Introduction: Automata theory is the study of abstract computing devices, or machines. Automata are essential for the study of the limits of computations. Finite Automata are a useful model for many important parts of hardware and software.

Alphabets: A symbol is an abstract entity. Letters and digits are examples of frequently used symbol. An alphabet is a finite, non-empty set of symbol and is denoted by Σ .

Example : $\Sigma = \{0,1\}$, $\Sigma = \{a,b,c\}$

Strings: is a finite set sequence of symbols chosen from alphabet.

Example: 011101 is a string from alphabet $\Sigma = \{0,1\}$.

Operations on string:

Concatenation: of two strings is formed by writing first string followed by second string with no space. *Ex.* $V = 'a'$, $W = 'cat'$ then $V.W = 'acat'$

Reverse: of the string W is obtained by writing the symbols of the string in reverse order and is denoted as W^R . *ex.* $W = 'the'$ then $W^R = 'eht'$.

Length: of a string W, denoted $|W|$ is the number of symbols composing the string. *Ex.* $W = 'the'$ then $|W| = 3$

Empty String: Denoted by ϵ is the string with zero symbol.

Power of Alphabets: if Σ is alphabet, the set of all strings of certain length can be expressed from that alphabet by using exponential notation. *Ex.* $\Sigma = \{a,b\}$ then

$$\Sigma^0 = \{ \epsilon \}$$

$$\Sigma^1 = \{a,b\}$$

$$\Sigma^2 = \{aa, ab, bb, ba\}$$

$$\Sigma^3 = \{aaa, aab, aba, abb, bbb, baa, bab, bba\}$$

$$\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \dots \cup \Sigma^n$$

Language: set of strings, all are chosen from Σ^* , where Σ is a particular alphabet is called a language.

$$L \subseteq \Sigma^*$$

Empty Language: represented as \emptyset where $\emptyset = \{ \}$, does not contain any element.

Introduction to Finite Automata and formal language

An automaton is a construct that possesses all the indispensable features of a digital computer. It accepts input, produces output, may have some temporary storage and can make decisions in transforming the input into the output.

A formal language is an abstraction of the general characteristics of programming languages. A formal language consists of a set of symbols and some rules of by which these symbols can be combined into entities called sentences.

Finite automata are computing devices that accept/recognize regular languages and are used to model operations of many systems. Their operations can be simulated by a very simple computer program.

Automaton:

A *finite automaton* (FA, also called a *finite-state automaton* or a *finite-state machine*) is a mathematical tool used to describe processes involving inputs and outputs. An FA can be in one of several states and can switch between states depending on symbols that it inputs. Once it settles in a state, it reads the next input symbol, performs some computational task associated with the new input, outputs a symbol, and switches to a new state depending on the input. Notice that the new state may be identical to the current state.

DFA: Deterministic Finite Automata

Definition: DFA is a finite automaton in which for each input symbol there is exactly one transition out of each state

A DFA is 5-tuple or quintuple $M = (Q, \Sigma, \delta, q_0, F)$ where

Q is non-empty, finite set of states.

Σ is non-empty, finite set of input alphabets.

δ is transition function, which is a mapping from $Q \times \Sigma \rightarrow Q$.

$q_0 \in Q$ is the start state.

$F \subseteq Q$ is set of accepting or final states.

Note: For each input symbol a , from a given state there is exactly one transition (there can be no

transitions from a state also) and it is sure (or can determine) to which state the machine enters. So, the machine is called *Deterministic machine*. Since it has finite number of states the machine is called ***Deterministic finite machine or Deterministic Finite Automaton*** or Finite State Machine (FSM).

The language accepted by DFA is

$$L(M) = \{ w \mid w \in \Sigma^* \text{ and } \delta^*(q_0, w) \in F \}$$

Designing a FA: The automaton can be described by

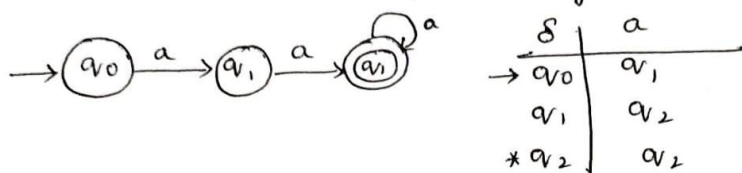
1. *Transition diagram:* is a graph
 - a. For each state in Q there is a node
 - b. For each state q in Q and each input symbol a in Σ , let $\delta(q,a) = p$, then the transition diagram has an arc from node q labeled a .
 - c. There is an arrow into the start state q_0 labeled start
 - d. Nodes corresponding to accepting states are marked by double circle
2. *Transition table* : is a conventional tabular representation of a function like δ that take two arguments and returns a value
 - a. Rows of the table corresponds to the states and the columns correspond to the input

Examples

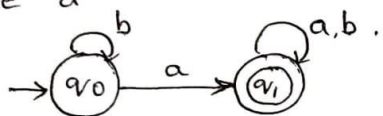
1. Draw a DFA to accept string of a's having atleast one 'a'.



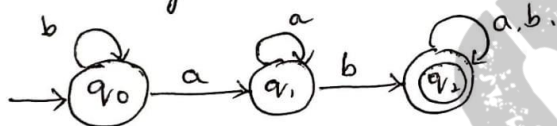
2. Draw a DFA to accept string of a's having atleast 2 a's.



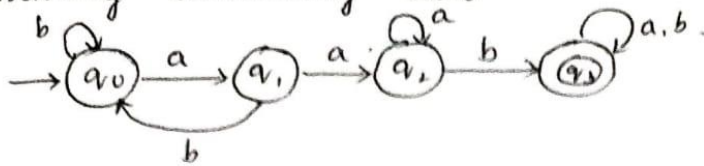
3. Draw DFA to accept strings of a's & b's having atleast one 'a'.



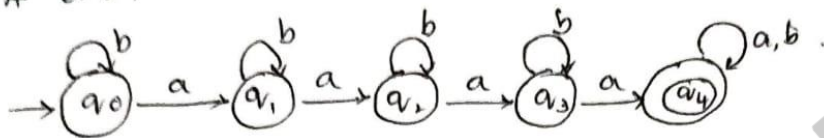
4. Draw a DFA to accept string of a's & b's having substring 'ab'.



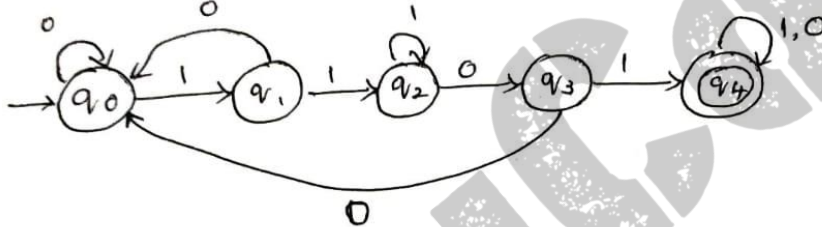
5. Draw a DFA to accept strings of a's & b's having substring 'aab'.



6. Obtain DFA to accept strings of a's & b's having 4 a's.



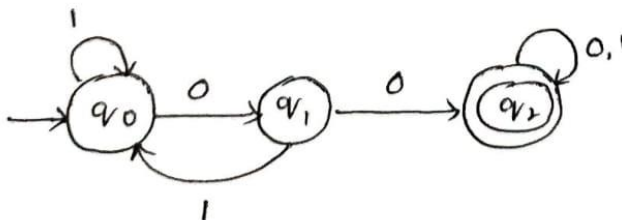
7. Obtain DFA to accept the set of all strings containing 1101 as a substring.



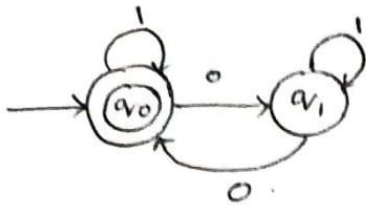
8. Obtain DFA to accept set of all strings containing at least 2 o's. [not consecutive].



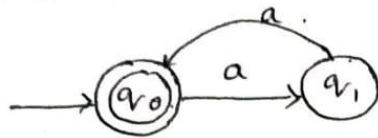
(b) 2 consecutive o's.



9. Language of all strings in which the number of 0's is even.



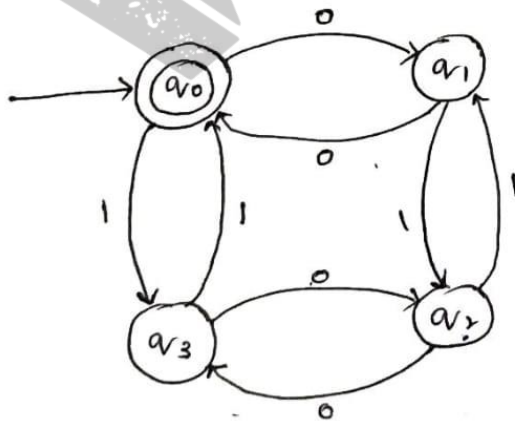
10. Obtain a DFA that recognises all even numbers of a's where $\Sigma = \{a\}$.



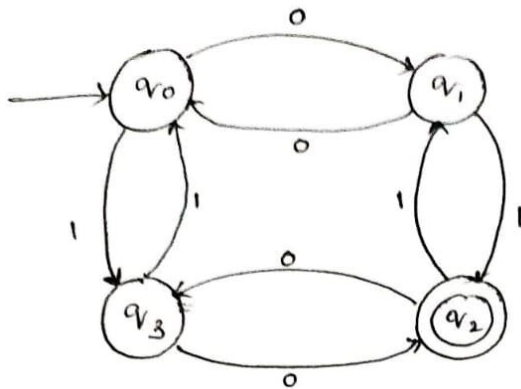
10. Obtain a DFA that recognises all odd number of a's where $\Sigma = \{a\}$.



11. Language of all strings in which both the number of 0's & 1's are even.

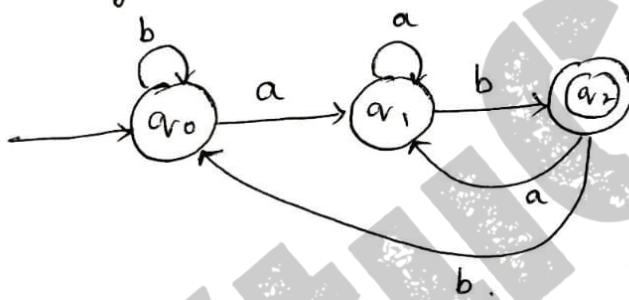


12. Language of all strings in which both 0's & 1's are odd.

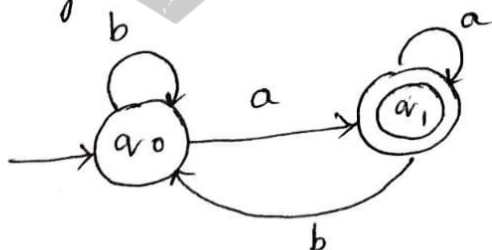


Ending

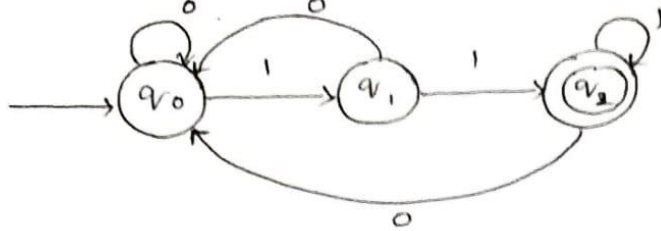
2. Draw a dfa to accept the strings of a & b ending with ab. $\Sigma = \{a, b\}$.



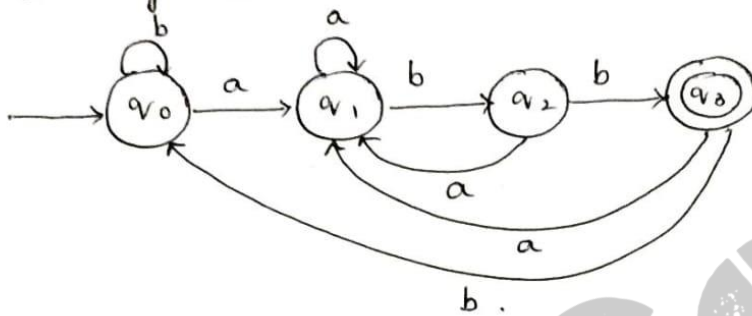
1. Draw a dfa to accept the strings of a & b ending with 'a'.



3. Obtain dfa accepting the set of all strings that end with 11.

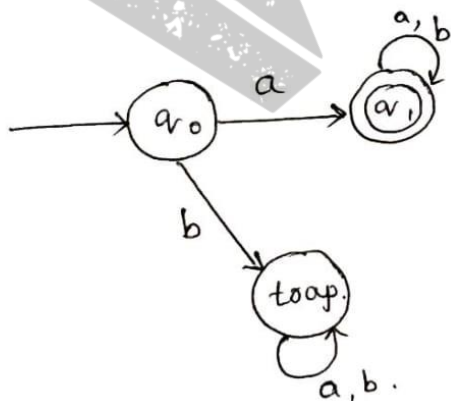


4. Draw a dfa to accept the strings of a & b ending with abb. $\Sigma = \{a, b\}$.

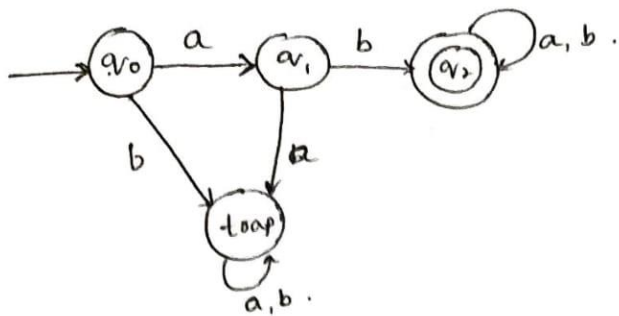


starting:-

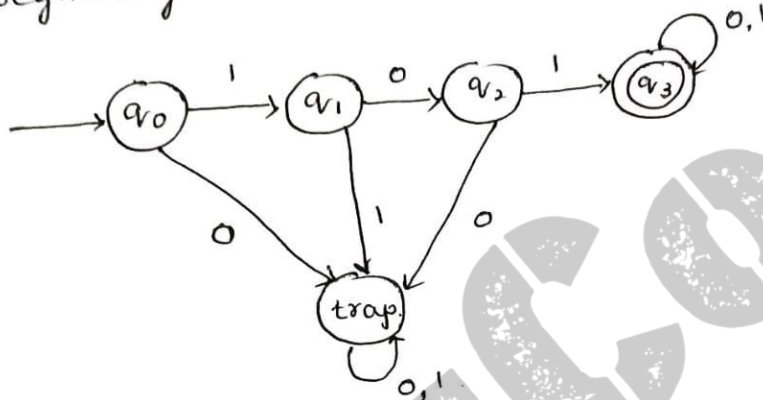
1. Draw a dfa to accept the set of all strings that start with 'ab'. $\Sigma = \{a, b\}$.



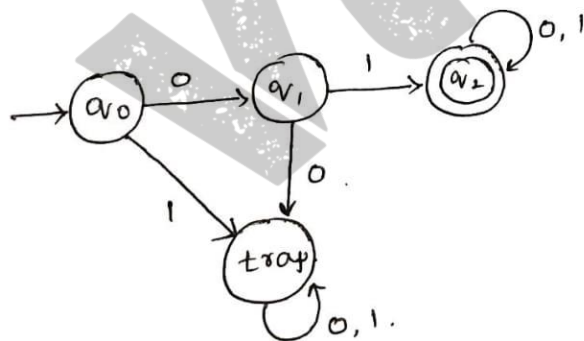
2. Draw a dfa to start with 'ab'



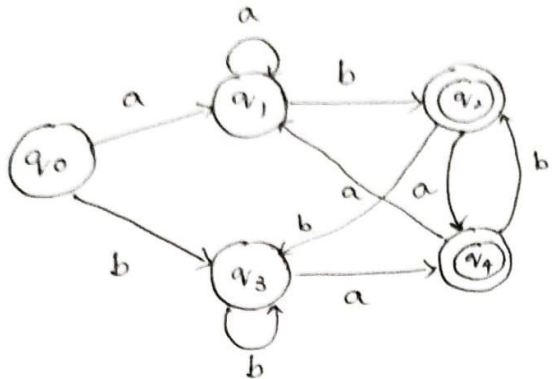
3. obtain a DFA accepting the set of all strings beginning with 101.



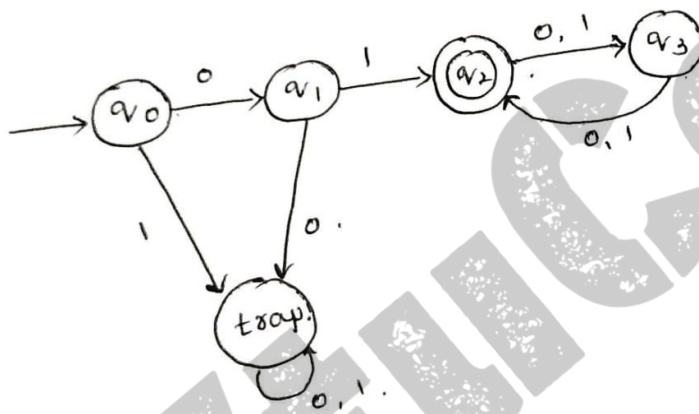
4. obtain DFA accepting the set of all strings that begin with 01.



Construct a DFA to accept strings of a's + b's ending with ab or ba.

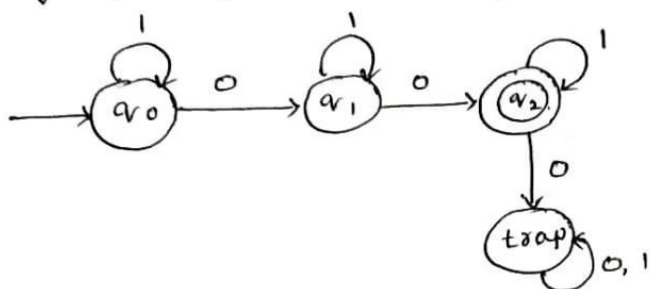


Design a DFA to accept the language $L = \{w \mid w \text{ is of even length \& begins with } 01\}$.



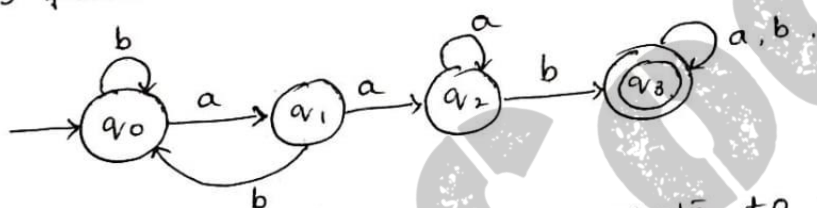
Misc:-

1. Language of all strings containing exactly two 0's.

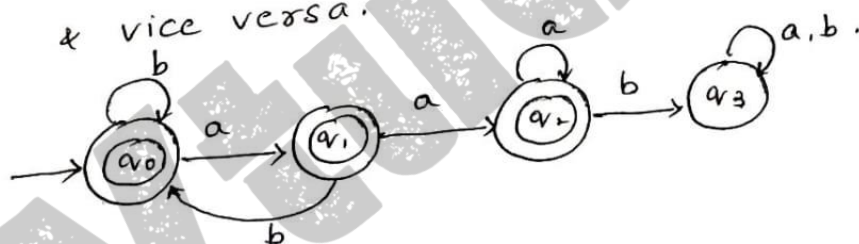


2. Obtain a DFA to accept strings of a's & b's except those containing substring "aab".

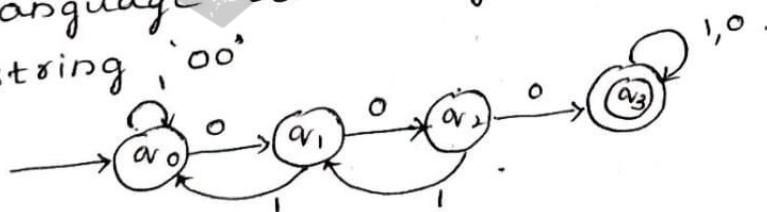
(i) find DFA to accept substring 'aab'.



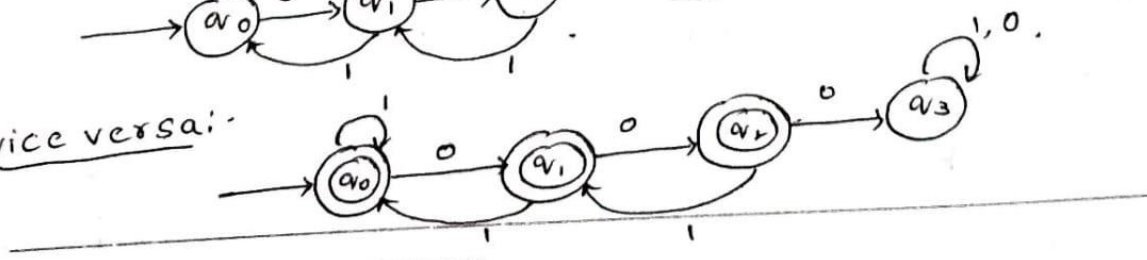
(ii) Now change all final state to non-final state & vice versa.



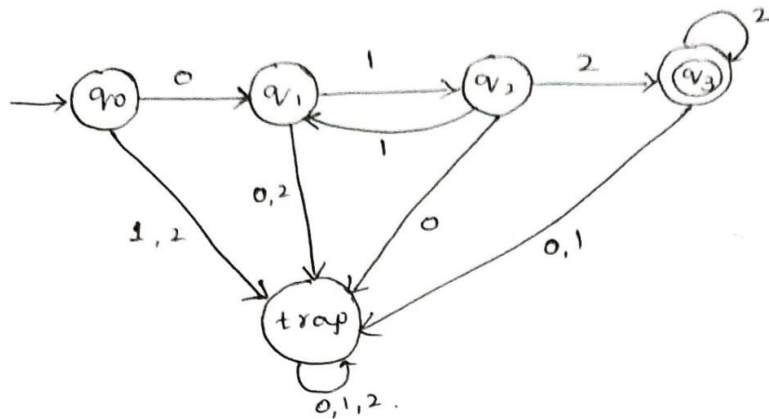
3. Language containing no more than 1 occurrence of string '00'.



vice versa:-

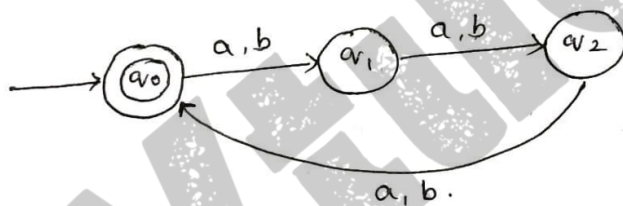


1. Obtain a DFA to accept strings of 0's, 1's & 2's beginning with '0' followed by odd number of 1's and ending with 2.



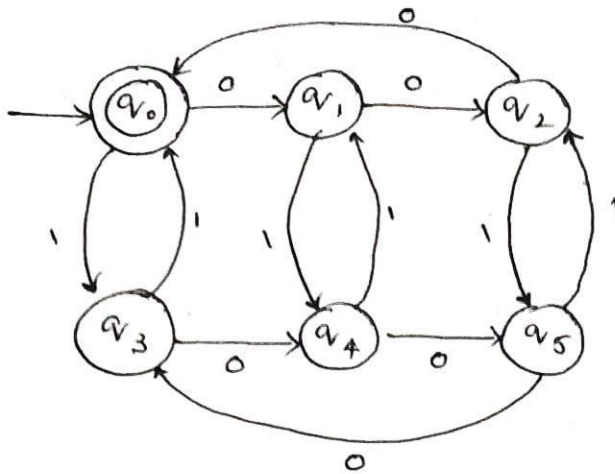
8. Obtain DFA to accept the language $L = \{w : |w| \bmod 3 = 0\}$ on $\Sigma = \{a, b\}$.

$$L = \{\epsilon, aaa, aab, aba, abb, baa, bab, \dots\}$$



9. Obtain DFA to accept strings of a's & b's such that

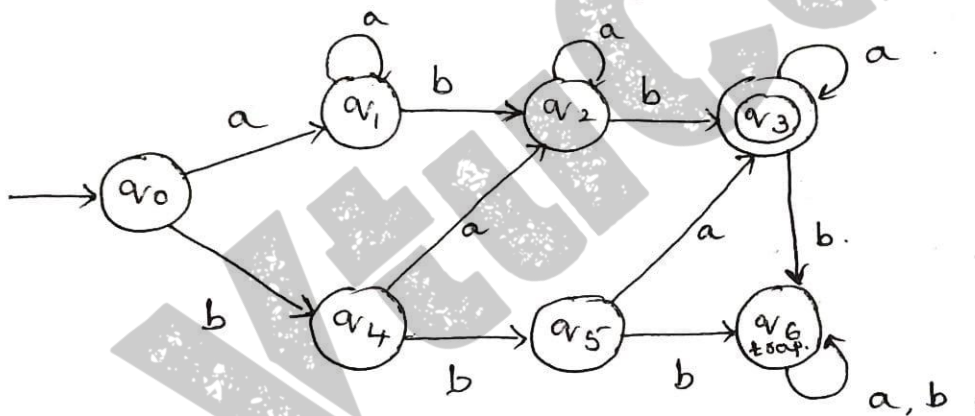
obtain DFA accepting the set of all strings such that number of 1's is even & the number of 0's is multiple of 3.



$$n_0(w) \bmod 3 = 0$$

$$n_1(w) \bmod 2 = 0$$

Draw a DFA to accept the language $L = \{w \mid n_a(w) \geq 1, n_b(w) = 2\}$.



Nondeterministic Finite Automaton: NFA

NFA is defined as follows: at some point in processing a string on a machine, the machine has a choice of moves; when this happens, it selects a move in an unspecified way. In other words, there can be zero, one or more than one transition out of a state with the same label. So the machine must choose which path to take. This leads us to the observation that a string may have more than one path through the machine that is each entry in the table for NFA is a set. This represents a relaxation of the rules for defining FA's.

Definition: An NFA is a 5-tuple or quintuple $M = (Q, \Sigma, \delta, q_0, F)$ where

Q is non empty, finite set of states.

Σ is non empty, finite set of input alphabets.

δ is transition function which is a mapping from $Q \times \Sigma$ to subsets of 2^Q . This function shows the change of state from one state to a set of states based on the input symbol.

$q_0 \in Q$ is the start state.

$F \subseteq Q$ is set of accepting or final states

Language Accepted by NFA:

Definition: Let $M = (Q, \Sigma, \delta, q_0, A)$ be a DFA where Q is set of finite states, Σ is set of input alphabets (from which a string can be formed), δ is transition function from $Q \times \{\Sigma \cup \epsilon\}$ to 2^Q , q_0 is the start state and A is the final or accepting state. The string (also called language) w accepted by an NFA can be defined in formal notation as:

$$L(M) = \{ w \mid w \in \Sigma^* \text{ and } \delta^*(q_0, w) = Q \text{ with at least one Component of } Q \text{ in } A \}$$

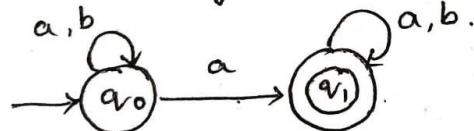
Examples

I obtain an NFA to accept set of all strings.
over the alphabet $\Sigma = \{a, b\}$.

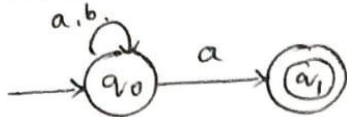
(a) starting with 'a'.



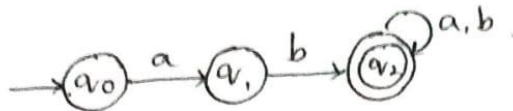
(b) containing 'a'



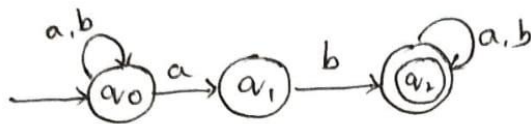
(c) ends with 'a'



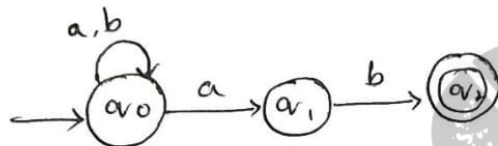
(d) starts with a, b.



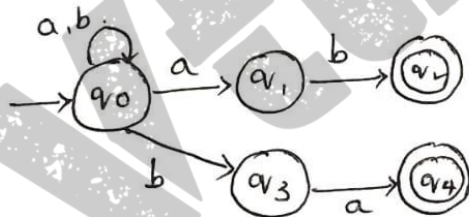
(e) contain 'ab'



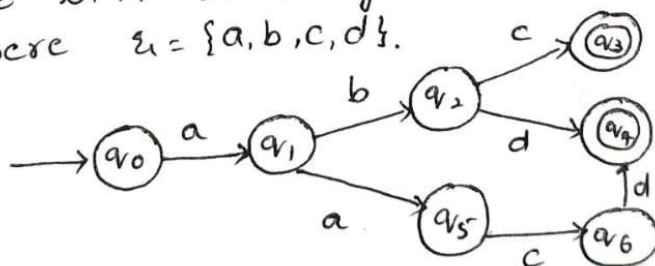
(f) ends with ab.



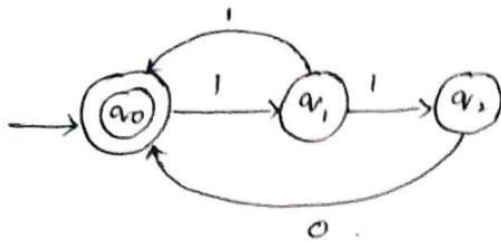
2. Obtain an NFA to accept strings ending with 'ab' or 'ba' over $\Sigma = \{a, b\}$.



3. Write NFA to recognise the string abc, abd, aacd where $\Sigma = \{a, b, c, d\}$.

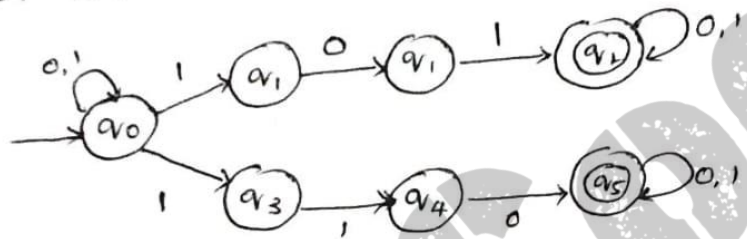


4. Write NFA to accept $\{11, 110\}^*$.

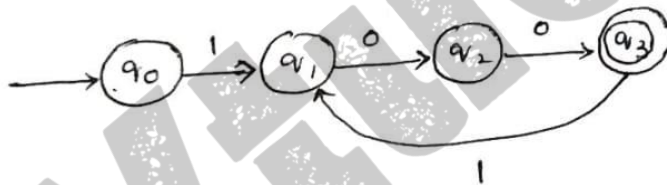


5. Write NFA to accept the following languages over $\{0, 1\}^*$.

(a) set of all strings such that containing either 101 or 110 as substring.

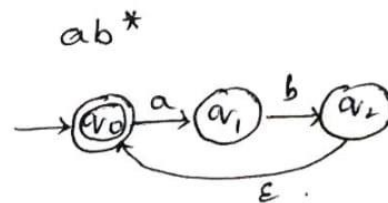
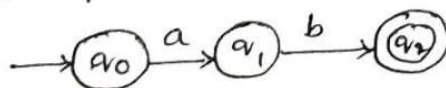


(b) Every '1' followed by '00'.

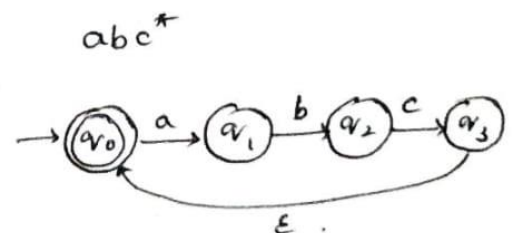
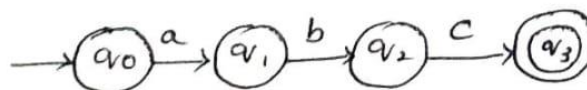


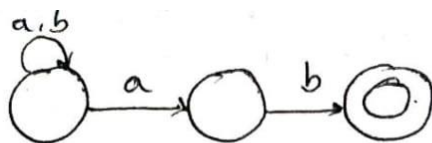
6. Design an NFA with 4 states that accept the language $\{ab, abc\}^*$.

Sol: To accept ab .



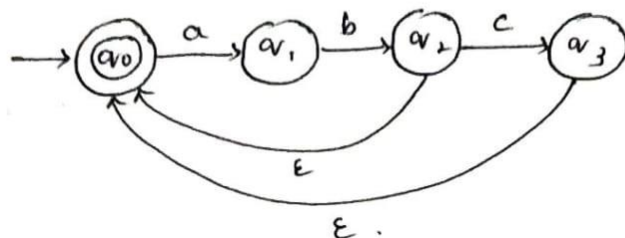
To accept abc .





ababab...

combining these 2.

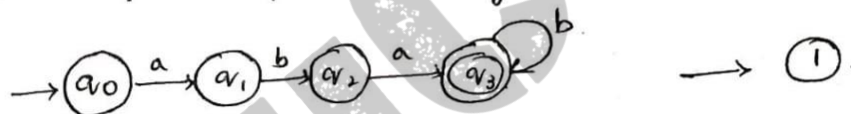


7. Design NFA with no more than 5 states for the set $\{abab^n \mid n \geq 0\} \cup \{aba^n \mid n \geq 0\}$.

solⁿ: consider the set $\{abab^n \mid n \geq 0\}$

for $n=0$ the string accepted is aba.

Draw a path for string $abab^n$ where $n \geq 0$.

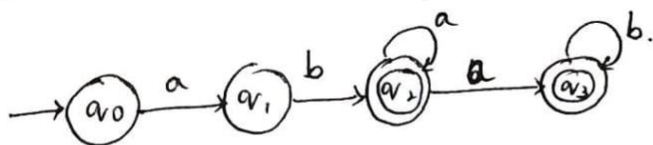


iii) for the set $\{aba^n \mid n \geq 0\}$.

for $n=0$ the string accepted is ab.



for set $\{abab^n\} \cup \{aba^n\}$. combine (1) + (2).



Extended Transition δ^* : Describes what happens when we start in any state and follow sequence of inputs.

Definition:

Let $M = (Q, \Sigma, \delta, q_0, F)$ where

Q is non-empty, finite set of states.

Σ is non-empty, finite set of input alphabets.

$q_0 \in Q$ is the start state.

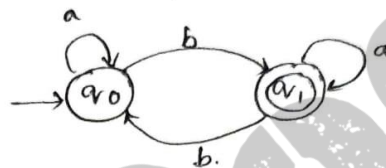
$F \subseteq Q$ is set of accepting or final states.

δ^* is extended transition function, which is a mapping from $Q \times \Sigma^* \rightarrow Q$. as follows:

- i. For any $q \in Q$, $\delta^*(q, \epsilon) = q$
- ii. For any $q \in Q, y \in \Sigma^*, a \in \Sigma$
 $\delta^*(q, ya) = \delta(\delta^*(q, y), a)$

Example:

consider a DFA



Find whether the string aabbab is accepted.

Sol: $y = aabbab$.

$$\delta^*(q_0, \epsilon) = q_0$$

$$\delta^*(q_0, a) = \delta(\delta^*(q_0, \epsilon), a) \Rightarrow \delta(q_0, a) = q_0$$

$$\delta^*(q_0, aa) = \delta(\delta^*(q_0, a), a) \Rightarrow \delta(q_0, a) = q_0$$

$$\delta^*(q_0, aab) = \delta(\delta^*(q_0, aa), b) \Rightarrow \delta(q_0, b) = q_1$$

$$\delta^*(q_0, aabb) = \delta(\delta^*(q_0, aab), b) \Rightarrow \delta(q_1, b) = q_0$$

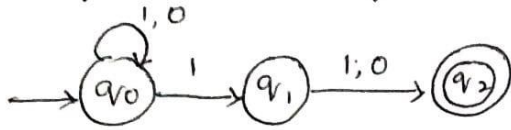
$$\delta^*(q_0, aabb a) = \delta(\delta^*(q_0, aabb), a) \Rightarrow \delta(q_0, a) = q_0$$

$$\delta^*(q_0, aabb ab) = \delta(\delta^*(q_0, aabb a), b) \Rightarrow \delta(q_0, b) = q_1$$

q_1 is final state hence the string

aabbab is accepted.

Describe the processing of input 00101 by the following NFA using extended transition.



Solⁿ: $\hat{\delta}(q_0, \epsilon) = \{q_0\}$.

$$\hat{\delta}(q_0, 0) = \delta(q_0, 0) = \{q_0\}.$$

$$\hat{\delta}(q_0, 00) = \hat{\delta}(\hat{\delta}(q_0, 0), 0) = \delta(q_0, 0) = \{q_0\}.$$

$$\hat{\delta}(q_0, 001) = \delta(\hat{\delta}(q_0, 00), 1) = \delta(q_0, 1) = \{q_0, q_1\}.$$

$$\begin{aligned} \hat{\delta}(q_0, 0010) &= \delta(\hat{\delta}(q_0, 001), 0) = \delta(\{q_0, q_1\}, 0) = \\ &= \delta(q_0, 0) \cup \delta(q_1, 0) = \{q_0, q_2\} \end{aligned}$$

$$\begin{aligned} \hat{\delta}(q_0, 00101) &= \delta(\hat{\delta}(q_0, 0010), 1) = \delta(\{q_0, q_2\}, 1) \\ &= \delta(q_0, 1) \cup \delta(q_2, 1) \\ &= \{q_0, q_2\}. \end{aligned}$$

Since $q_2 \in F$, string is accepted.

FINITE AUTOMATA WITH ϵ TRANSITIONS

ϵ -NFA Epsilon NFA

Is an NFA with Epsilon transitions. The NFA which includes transitions on the empty input ϵ is called ϵ -NFA

Definition: An ϵ -NFA is a 5-tuple or quintuple $M = (Q, \Sigma, \delta, q_0, F)$ where

Q is non empty, finite set of states.

Σ is non empty, finite set of input alphabets.

δ is transition function which is a mapping from $Q \times \{\Sigma \cup \epsilon\}$ to subsets of 2^Q .

This function shows the change of state from one state to a set of states based on the input symbol.

$q_0 \in Q$ is the start state.

$F \subseteq Q$ is set of accepting or final states

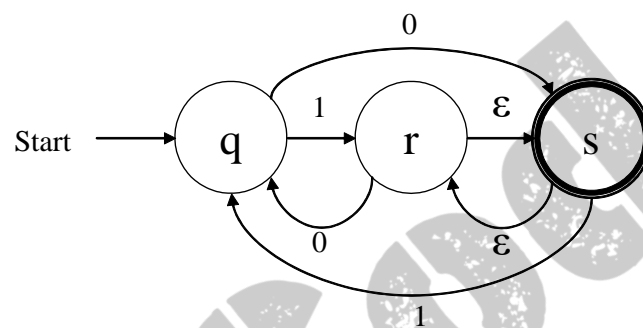
ϵ -Closure of a state

Definition:

ϵ -closure is a set of all vertices p such that there is a path from q to p labelled ϵ .

Let $M = (Q, \Sigma, \delta, q_0, F)$ be an NFA with $M = (Q, \Sigma, \delta, q_0, F)$ transitions and let S be any subset of Q . The ϵ -closure of S denoted as $\epsilon(S)$ is defined by

1. Every element of S is an element of $\epsilon(S)$.
2. For any $q \in \epsilon(S)$ every element of $\delta(q, \epsilon)$ is in $\epsilon(S)$
3. No other element are in $\epsilon(S)$

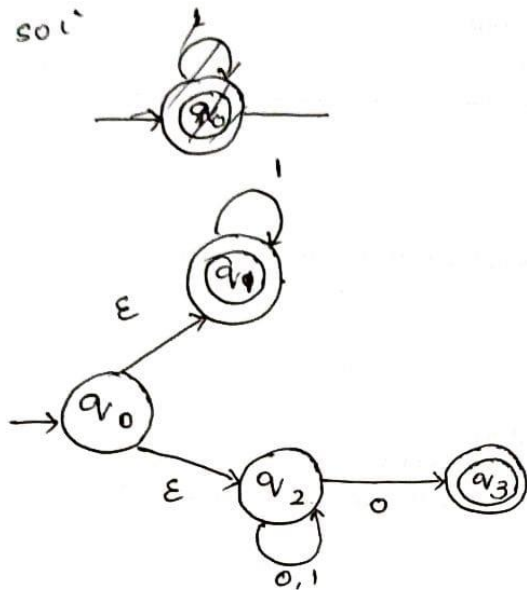


- $\epsilon\text{-closure}(q) = \{ q \}$
- $\epsilon\text{-closure}(r) = \{ r, s \}$

Examples

2. Draw a ϵ -NFA that accepts all binary strings where the last symbol is '0' or that contain only 1's.

sol:



$$\epsilon\text{-closure}(q_0) = \{q_0, q_1, q_2\}.$$

$$\epsilon\text{-closure}(q_1) = \{q_1\}.$$

$$\epsilon\text{-closure}(q_2) = \{q_2\}.$$

$$\epsilon\text{-closure}(q_3) = \{q_3\}.$$

3. Design ϵ -NFA that accepts set of all strings consisting of zero or more a's followed by zero or more b's followed by zero or more c's.



$$\epsilon\text{-closure}(q_0) = \{q_0, q_1, q_2, q_3\}.$$

$$\epsilon\text{-closure}(q_1) = \{q_1, q_2, q_3\}.$$

$$\epsilon\text{-closure}(q_2) = \{q_2, q_3\}.$$

$$\epsilon\text{-closure}(q_3) = \{q_3\}.$$

With an NFA, at any point in a scanning of the input string we may be faced with a choice of any number of paths to be followed to reach a final state. With a DFA there is never a choice of paths. So, when we construct a DFA which accepts the same language as a particular NFA, the conversion process effectively involves merging all possible states which can be reached on a particular input character, from a particular state, into a single, composite, state which represents all those paths.

Let $M_N = (Q_N, \Sigma, \delta_N, q_0, F_N)$ be an NFA and accepts the language $L(M_N)$. There should be an equivalent DFA $M_D = (Q_D, \Sigma_D, \delta_D, q_0, F_D)$ such that $L(M_D) = L(M_N)$. The procedure to convert an NFA to its equivalent DFA is shown below

Step1: The start state of NFA M_N is the start state of DFA M_D . So, add q_0 (which is the start state of NFA) to Q_D and find the transitions from this state.

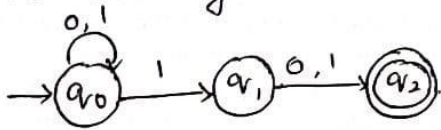
Step2: For each state $[q_i, q_j, \dots, q_k]$ in Q_D , the transitions for each input symbol in Σ can be obtained as shown below:

- $\delta_D([q_i, q_j, \dots, q_k], a) = \delta_N(q_i, a) \cup \delta_N(q_j, a) \cup \dots \delta_N(q_k, a)$
 - $a = [q_l, q_m, \dots, q_n]$ say.
- Add the state $[q_l, q_m, \dots, q_n]$ to Q_D , if it is not already in Q_D .
- Add the transition from $[q_i, q_j, \dots, q_k]$ to $[q_l, q_m, \dots, q_n]$ on the input symbol a iff the state $[q_l, q_m, \dots, q_n]$ is added to Q_D in the previous step.

Step3: The state $[q_a, q_b, \dots, q_c] \in Q_D$ is the final state, if at least one of the state in $q_a, q_b, \dots, q_c \in A_N$ i.e., at least one of the component in $[q_a, q_b, \dots, q_c]$ should be the final state of NFA.

Examples:-

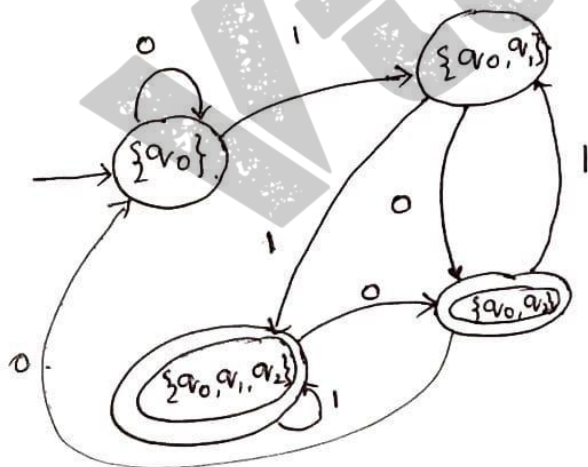
1. convert the given NFA to DFA.



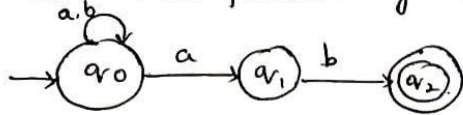
Solⁿ ∴

| δ_{NFA} | 0 | 1 |
|-------------------|-----------|----------------|
| $\rightarrow q_0$ | $\{q_0\}$ | $\{q_0, q_1\}$ |
| q_1 | $\{q_2\}$ | $\{q_2\}$ |
| q_2 | ϕ | ϕ |

| δ_{DFA} | 0 | 1 |
|----------------------------|----------------|---------------------|
| $\rightarrow \{q_0\}$ | $\{q_0\}$ | $\{q_0, q_1\}$ |
| $\{q_0, q_1\}$ | $\{q_0, q_2\}$ | $\{q_0, q_1, q_2\}$ |
| $\times \{q_0, q_2\}$ | $\{q_0\}$ | $\{q_0, q_1\}$ |
| $\times \{q_0, q_1, q_2\}$ | $\{q_0, q_2\}$ | $\{q_0, q_1, q_2\}$ |



2. Convert the following NFA to DFA.

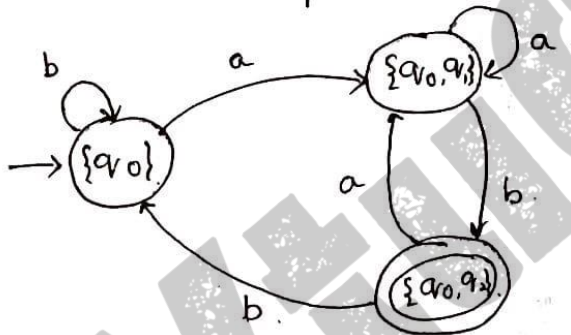


Sol: given δ_{NFA}

| | a | b |
|-------------------|----------------|-------------|
| $\rightarrow q_0$ | $\{q_0, q_1\}$ | $\{q_0\}$ |
| q_1 | \emptyset | $\{q_2\}$ |
| q_2 | \emptyset | \emptyset |

conversion

| δ_{DFA} | a | b |
|-----------------------|----------------|----------------|
| $\rightarrow \{q_0\}$ | $\{q_0, q_1\}$ | $\{q_0\}$ |
| $\{q_0, q_1\}$ | $\{q_0, q_1\}$ | $\{q_0, q_2\}$ |
| * $\{q_0, q_2\}$ | $\{q_0, q_1\}$ | $\{q_0\}$ |

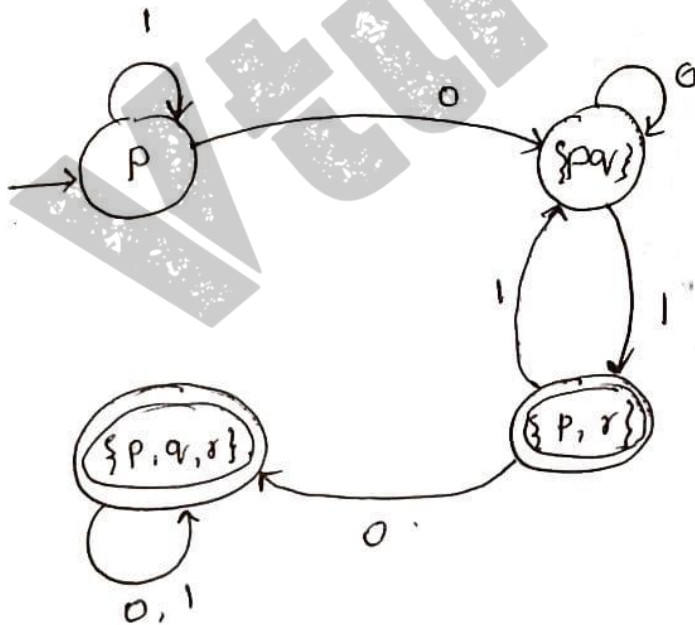


3. convert the following NFA to DFA.

| δ_{NFA} | 0 | 1 |
|-----------------|------------|---------|
| $\rightarrow P$ | $\{P, q\}$ | $\{P\}$ |
| q | q | $\{r\}$ |
| $* r$ | $\{P, r\}$ | $\{q\}$ |

Solⁿ:

| δ_{DFA} | 0 | 1 |
|---------------------|---------------|---------------|
| $\rightarrow \{P\}$ | $\{P, q\}$ | $\{P\}$ |
| $\{P, q\}$ | $\{P, q\}$ | $\{P, r\}$ |
| $* \{P, r\}$ | $\{P, q, r\}$ | $\{P, q\}$ |
| $* \{P, q, r\}$ | $\{P, q, r\}$ | $\{P, q, r\}$ |



4. Convert the given NFA to DFA.

| δ_{NFA} | 0 | 1 |
|-----------------|------------|-------------|
| $\rightarrow p$ | $\{p, q\}$ | $\{q\}$ |
| q | $\{r, s\}$ | $\{p\}$ |
| $* r$ | $\{p, s\}$ | $\{r\}$ |
| $* s$ | $\{q, r\}$ | \emptyset |

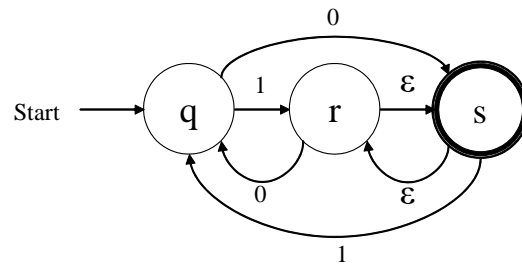
Solⁿ:-

| δ_{DFA} | 0 | 1 |
|---------------------|------------------|---------------|
| $\rightarrow \{p\}$ | $\{p, r\}$ | $\{q\}$ |
| $\{q\}$ | $\{r, s\}$ | $\{p\}$ |
| $\{p, r\}$ | $\{p, r, s\}$ | $\{q, r\}$ |
| $\{r, s\}$ | $\{p, s, q, r\}$ | $\{r\}$ |
| $\{p, q, r, s\}$ | $\{p, r, s, q\}$ | $\{q, p, r\}$ |
| $\{p, r, s\}$ | $\{p, q, s\}$ | $\{p, s\}$ |

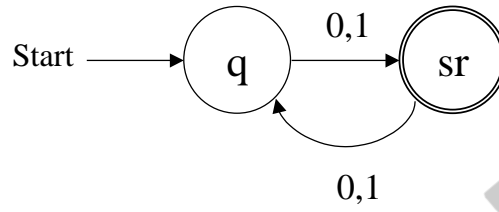
Conversion from ϵ -NFA-DFA (Subset Construction)

To eliminate ϵ -transitions, use the following to convert to a DFA

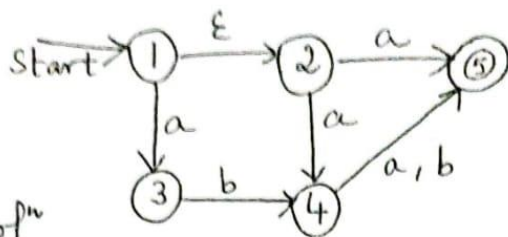
1. Compute ϵ -closure for the current state, resulting in a set of states S .
2. $\delta_D(S, a)$ is computed for all a in Σ by
 - a. Let $S = \{p_1, p_2, \dots, p_k\}$
 - b. Compute $\bigcup_{i=1}^k \delta(p_i, a)$ and call this set $\{r_1, r_2, r_3 \dots r_m\}$ This set is achieved by following input a , not by following any ϵ -transitions
 - c. Add the ϵ -transitions in by computing $\delta(S, a) = \bigcup_{i=1}^m \epsilon\text{-closure}(r_i)$
3. Make a state an accepting state if it includes any final states in the ϵ -NFA.



Converts to



Convert the following ϵ -NFA to DFA



Solⁿ

- * The start state is the ϵ -closure of the start state of the ϵ -NFA
- * The final states of the DFA will correspond to the subsets of the ϵ -NFA that contain a final state.

$$E\text{-CLOSURE}(1) = \{1, 2\}$$

$$\begin{aligned} E\text{-CLOSURE of } (S(1, a) \cup S(2, a)) \\ &= E\text{-CLOSURE of } (\{3\} \cup \{4, 5\}) \\ &= E\text{-CLOSURE of } \{3, 4, 5\} = \{3, 4, 5\} \end{aligned}$$

$$\begin{aligned} E\text{-CLOSURE of } (S(1, b) \cup S(2, b)) \\ &= E\text{-CLOSURE of } (\{\phi\} \cup \{\phi\}) \\ &= \{\phi\} \end{aligned}$$

$$\begin{aligned} E\text{-CLOSURE of } (S(3, a) \cup S(4, a) \cup S(5, a)) \\ &= E\text{-CLOSURE of } (\{\phi\} \cup \{5\} \cup \{\phi\}) \\ &= \{5\} \end{aligned}$$

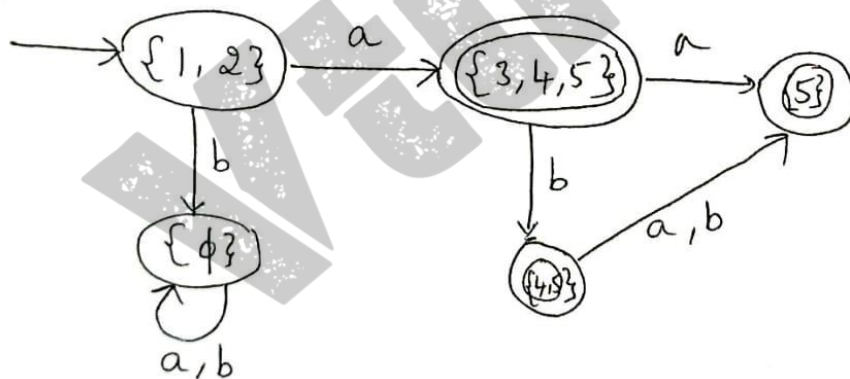
$$\begin{aligned}
 & \text{E-CLOSURE of } (s(3,b) \cup s(4,b) \cup s(5,b)) \\
 &= \text{E-CLOSURE of } (\{4\} \cup \{5\} \cup \{\phi\}) \\
 &= \text{E-CLOSURE of } (\{4,5\}) \\
 &= \{4,5\}
 \end{aligned}$$

$$\begin{aligned}
 & \text{E-CLOSURE of } (s(5,a)) \\
 &= \{\phi\}
 \end{aligned}$$

$$\text{E-CLOSURE of } (s(5,b)) = \{\phi\}$$

$$\begin{aligned}
 & \text{E-CLOSURE of } (s(4,a) \cup s(5,a)) \\
 &= \text{E-CLOSURE of } (\{5\} \cup \{\phi\}) \\
 &= \{5\}
 \end{aligned}$$

$$\begin{aligned}
 & \text{E-CLOSURE of } (s(4,b) \cup s(5,b)) \\
 &= \text{E-CLOSURE of } (\{5\} \cup \{\phi\}) = \{5\}
 \end{aligned}$$



One can show the connection between state $\{5\}$ & $\{\phi\}$ or rule out.

If $M_D = (Q_D, \Sigma_D, \delta_D, q_0, F_D)$ is the DFA constructed from NFA $M_N = (Q_N, \Sigma, \delta_N, q_0, F_N)$ by the subset construction, then $L(M_D) = L(M_N)$.

Proof: Let $|w|=0$, that is $w = \epsilon$. By the basis definitions of δ^* for DFA's and NFA's both $\delta^*({q_0}, \epsilon)$ and $\delta^*({q_0}, \epsilon)$ are $\{q_0\}$

Let w be of length $n+1$, and assume the statement for length n . break w as $w=xa$, where a is the final symbol of w . by the inductive hypothesis $\delta^*({q_0}, x) = \delta^*({q_0}, x)$. let both these sets of N's states be $\{p_1, p_2, \dots, p_k\}$

The inductive part of the definition of δ^* for NFA's tells that:

$$\delta^*({q_0}, w) = \bigcup_{i=1}^k \delta_N(p_i, a) \text{ -----1}$$

The subset construction, on the other hand, tells that :

$$\delta_D(\{p_1, p_2, \dots, p_k\}, a) = \bigcup_{i=1}^k \delta_N(p_i, a) \text{ -----2}$$

Using eqn 2 and the fact that $\delta^*({q_0}, x) = \{p_1, p_2, \dots, p_k\}$ in the inductive part of the definition of δ^* for DFA's

$$\delta^*({q_0}, x) = \delta_D(\delta^*({q_0}, x), a) = \delta_D(\{p_1, p_2, \dots, p_k\}, a) = \bigcup_{i=1}^k \delta_N(p_i, a) \text{ -----3}$$

Thus equation 2 and 3 demonstrate that $\delta^*({q_0}, w) = \delta_N^*({q_0}, w)$

Hence proved that $L(M_D) = L(M_N)$.

Difference between DFA, NFA, ϵ -NFA

| DFA | NFA | ϵ -NFA |
|--|--|---|
| DFA is a 5-tuple $M = (Q, \Sigma, \delta, q_0, F)$ where <ul style="list-style-type: none"> • Q is non-empty, finite set of states. • Σ is non-empty, finite set of input alphabets. • $\delta : Q \times \Sigma \rightarrow Q$. • $q_0 \in Q$ is the start state. • $F \subseteq Q$ is set of accepting or final states. | $M = (Q, \Sigma, \delta, q_0, F)$ where <ul style="list-style-type: none"> • Q is non empty, finite set of states. • Σ is non empty, finite set of input alphabets. • $\delta : Q \times \Sigma \rightarrow 2^Q$. • $q_0 \in Q$ is the start state. • $F \subseteq Q$ is set of accepting or final states | $M = (Q, \Sigma, \delta, q_0, F)$ where <ul style="list-style-type: none"> • Q is non empty, finite set of states. • Σ is non empty, finite set of input alphabets • $\delta : \text{from } Q \times \{\Sigma \cup \epsilon\} \rightarrow 2^Q$. • $q_0 \in Q$ is the start state. • $F \subseteq Q$ is set of accepting or final states. |

| | | |
|---|--|---|
| There can be zero or one transition from a state on an input symbol; | There can be zero, one or more number of transitions from a state on an input symbol | There can be zero, one or more number of transitions from a state with or without an input symbol |
| Difficult to design | The NFA are easier to design | Easy to construct using regular expression |
| More number of transitions | Less number of transitions | More number of transitions compared to NFA |
| Less powerful since at any point of time it will be in only one state | More powerful; than DFA since it can be in more than one state | More powerful than NFA since at any point of time it will be in more than one state with or without giving any input. |

INTRODUCTION TO COMPILER DESIGN

OVERVIEW OF LANGUAGE PROCESSING SYSTEM

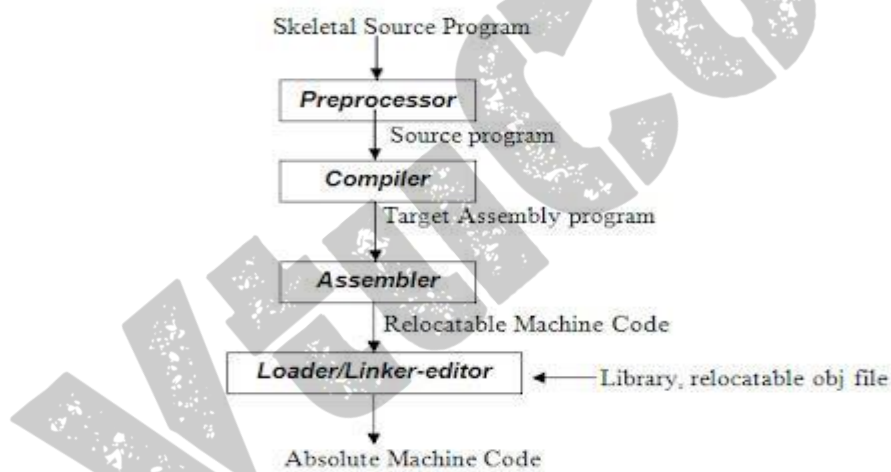


Fig 1.1 Language –processing System

Preprocessor

A preprocessor produce input to compilers. They may perform the following functions.

1. *Macro processing*: A preprocessor may allow a user to define macros that are short hands for longer constructs.
2. *File inclusion*: A preprocessor may include header files into the program text.
3. *Rational preprocessor*: these preprocessors augment older languages with more modern flow-of-control and data structuring facilities.
4. *Language Extensions*: These preprocessor attempts to add capabilities to the language by certain amounts to build-in macro