# Unit 2

## Regular Languages and classifications of Grammars

### Grammar

A Grammar G can be formally written as a 4-tuple $(N, T, S, P)$ where

- N or $V_N$ is a set of Non-Terminal Symbols.
- T or $\Sigma$ is a set of Terminal symbols.
- S is the start Symbol, $S \in N$.
- P is Production rules for terminals and non-terminals.

Example 1.

Grammar $G_1$:

$$(\{S, A, B\}, \{a, b\}, S, \{S \rightarrow AB,\ A \rightarrow a,\ B \rightarrow b\})$$

- Here, S, A and B are non-terminal symbols.
- a and b are Terminal symbols
- S is the start Symbol, $S \in N$.
- Products P : $S \rightarrow AB$
  $A \rightarrow a$
  $B \rightarrow b$

Example 2:

Grammar $G_2 = (\{S, A\}, \{a, b\}, S, \{S \to aAb, aA \to aaAb, A \to \epsilon\})$

- Here, S and A are non-terminal symbols.
- a and b are terminal symbols, $\epsilon$ is empty string
- S is the start symbol $S \in N$.
- Productions P: $S \to aAb$
  $aA \to aaAb$
  $A \to \epsilon$

## Derivations from a Grammar

Strings may be derived from other strings using the productions in a grammar. If a grammar G has a production $\alpha \to \beta$, we can say that $x\alpha y$ derives $x\beta y$ in G.

Example:-

$G = (\{S, A\}, \{a, b\}, S, \{S \to aAb, aA \to aaAb, A \to \epsilon\})$

Some of the strings that can be derived are:-

$S = aAb$ using Production $S \to aAb$

$= aaAbb$ using Production $aA \to aaAb$

$= aaaAbbb$ using Production $aA \to aaAb$

$= aaabbb$ using Production $A \to \epsilon$

## Language generated by Grammar

The set of all strings that can be derived from a grammar is said to be the language generated from that grammar.

### Example 1

Consider a grammar G :-

$N = \{S, A, B\}$, $T = \{a, b\}$ $P = \{S \rightarrow AB$

$A \rightarrow a$

$B \rightarrow b\}$

Here S produces AB, and we can replace A by a, and B by b.

Here the only accepted string is ab,

i,e $L(G) = \{ab\}$

### Example 2:

Consider Grammar G :-

$N = \{S, A, B\}$, $T = \{a, b\}$ $P = \{S \rightarrow AB,$

$A \rightarrow aA \mid a,$

$B \rightarrow bB \mid b\}$

$L(G) = \{ab, aab, abb, aabb, \ldots\}$

# Construction of a Grammar Generating a language.

We consider some languages and convert it into a grammar G which produces those languages.

Example :-

Suppose $L(G) = \{a^m b^n \mid m \geq 0 \text{ and } n > 0\}$. We have to find out the grammar G which produces L(G)

Solution.

The strings accepted can be rewritten as

$$L(G) = \{b, ab, bb, aab, abb, \ldots\}$$

Here, start symbol have to take atleast one 'b' preceded by any number of 'a's including null.

To accept the string set {b, ab, bb, aab, ....} we have considered the following productions :-

$S \to aS$, $S \to B$, $B \to b$ and $B \to bB$

$S \to B \to b$ (Accepted)

$S \to B \to bB \to bb$ (Accepted)

$S \to aS \to aB \to ab$ (Accepted)

$S \to aS \to aaS \to aaB \to aab$ (Accepted)

$S \to aS \to aB \to abB \to abb$ (Accepted)

Thus we can prove every single string in L(G) is accepted by the language generated by the production set.

Example 2:

Suppose $L(G) = \{ a^m b^n \mid m > 0 \text{ and } n \geq 0 \}$, we have to find out the grammar G which produces $L(G)$

Solution

The set of strings accepted can be rewritten as:

$L(G) = \{ a, aa, ab, aaa, aab, abb, \ldots \}$

Here, the start symbol have to take atleast one 'a' followed by any number of 'b' including null.

To accept string set $\{ a, aa, ab, aaa, \ldots \}$ we have taken the productions:

$S \to aA$ , $A \to aA$ , $A \to B$ , $B \to bB$ , $B \to \lambda$

$S \to aA \to aB \to a\lambda \to a$

$S \to aA \to aaA \to aaB \to aa\lambda \to aa$ (Accepted)

$S \to aA \to aB \to abB \to ab\lambda \to ab$ (Accepted)

$S \to aA \to aaA \to aaaA \to aaaB \to aaa\lambda \to aaa$ (Accepted)

$S \to aA \to aaA \to aaB \to aabB \to aab\lambda \to aab$ (Accepted)

$S \to aA \to aB \to abB \to abbB \to abb\lambda \to abb$ (Accepted)

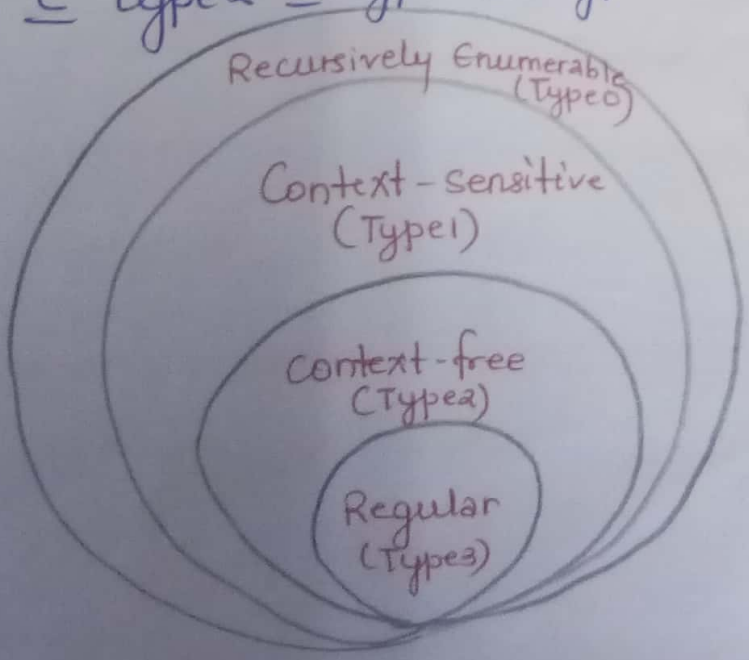
Thus we can prove every single string in $L(G)$ is accepted by the language generated by the production set.

# Chomsky classifications of Grammars

The hierarchy of grammars was described by Noam Chomsky. four Types of Grammars :-

| Grammar | Grammar Accepted | Language Accepted | Automaton |
|---|---|---|---|
| Type 0 | unrestricted grammar | Recursively Enumerable language | Turing Machine |
| Type 1 | Context-Sensitive grammar | Context-Sensitive language | Linear Bounded Automaton |
| Type 2 | context-free grammar | context-free language | Pushdown Automaton |
| Type 3 | Regular grammar | Regular language | finite state Automaton |

The following diagram shows containment of Type 3 $\subseteq$ Type 2 $\subseteq$ Type 1 $\subseteq$ Type 0.



## Type-3 Grammar

- They generate Regular languages.
- They must have a single non-terminal on the left hand side and a right hand side consisting of a single terminal or single terminal followed by a single Non-terminal.
- The productions must be in the form $X \longrightarrow a$, or $X \longrightarrow aY$ where $X, Y \in N$ (non-terminal) and $a \in T$ (Terminal)
- The Rule $S \longrightarrow \epsilon$ is allowed if S does not appear on the right side of any rule.

Example :-

$X \longrightarrow \epsilon$

$X \longrightarrow a$

$X \longrightarrow aY$

## Type 2 :- Grammar

- They generate context free languages.
- The productions must be in the form $A \longrightarrow Y$ where $A \in N$ (non-terminal) and $Y \in (T \cup N)^*$ (Strings of terminals and non-terminals).
- These languages generated by these grammars are to be recognized by non-deterministic Pushdown automaton.

Example :-

$S \longrightarrow Xa$

$X \longrightarrow a$

$X \longrightarrow aX$

$X \longrightarrow abc$

$X \longrightarrow \epsilon$

# Type 1 : Grammar

- They generate context-sensitive languages.
- The Productions must be in the form

  $\alpha A \beta \rightarrow \alpha \gamma \beta$ where $A \in N$ (non-terminal) and $\alpha, \beta, \gamma \in (T \cup N)^*$ (strings of terminal and non-terminals). The strings $\alpha$ and $\beta$ may be empty, but $\gamma$ must be non-empty.
- The Rule $S \rightarrow \epsilon$ is allowed if S does not appear on the right side of any rule.
- The languages generated by these grammars are recognized by linear Bounded automaton.

Example :-

$AB \rightarrow AbBC$

$A \rightarrow bcA$

$B \rightarrow b$

## Type 0 Grammar

- They generate recursively enumerable languages.
- The production have no restrictions. They are any Phase structure grammar including all formal Grammars.
- They generate the languages that are recognized by Turing machine.
- The production can be in the form like $\alpha \rightarrow \beta$ where $\alpha$ is a string of terminals and non-terminals with atleast one non terminal and $\alpha$ cannot be null. $\beta$ is a string of terminals and non-terminals.

Example.

$S \rightarrow ACaB$

$Bc \rightarrow acB$

$CB \rightarrow DB$

$aD \rightarrow Db$

# Regular Languages And Regular Grammars

## Regular Expressions

A Regular Expression (RE) can be recursively defined as follows:

1. $\varepsilon$ is a Regular Expression that indicates the language containing an empty string.
2. $(L(\varepsilon) = \{\epsilon\})$
3. $\phi$ is a Regular Expression denoting an empty Language. $(L(\phi) = \{\ \})$
4. X is a Regular Expression. where $L = \{X\}$
5. If X is a Regular Expression denoting the language $L(X)$ and Y is a Regular Expression denoting the language $L(Y)$, then
   - $X+Y$ is a Regular Expression corresponding to the language $L(X) \cup L(Y)$ where
     $L(X+Y) = L(X) \cup L(Y)$
   - $X.Y$ is a Regular Expression corresponding to the language $L(X).L(Y)$ where
     $L(X.Y) = L(X).L(Y)$
   - $R^*$ is a Regular Expression corresponding to the language $L(R^*)$ where $L(R^*) = (L(R))^*$
6. If we apply any of Rules several times from 1-5 they are Regular Expressions.

Regular Expression is a Mathematical symbolism or an Explicit formula to describe a language.

Q) Construct Regular Expression for following Languages
$L \in \{a,b\}^*$

1. strings begins with a

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

$L = \{ a, aa, ab, aaa, aab, aba, abb, \ldots \}$

To represent the 1st symbol of every string

$r_1 = a$

To represent remaining symbol of w,

$r_2 = (a+b)^*$

Thus the Regular Expression for L is

$$r_1 r_2 = a(a+b)^*$$

2. strings begins with a and ends with b.

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

$L = \{ ab, aab, abb, aaab, aabb, abab, \ldots \}$

To represent the 1st symbol of w

$r_1 = a$

To represent the middle symbol of w

$r_2 = (a+b)^*$

To represent the last symbol of w

$r_3 = b$

Thus the Regular Expression for L is

$$r_1 r_2 r_3 = a(a+b)^* b$$

3. Strings ending with abb.

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

$L = \{abb, aabb, babb, aaabb, \ldots\}$

To represent the beginning part

$r_1 = (a+b)^*$

To represent third last symbol of w

$r_2 = a$

To represent second last symbol of w

$r_3 = b$

To represent last symbol of w

$r_4 = b$

Thus the Regular Expression for L is

$$r_1 r_2 r_3 r_4 = (a+b)^* abb$$

4. Strings containing even number of a's.

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

$L = \{aa, bb, aba, aab, baa, \ldots\}$

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

Thus the Regular Expression for L is

$$r = (b^* a b^* a b^*)^*$$

if consecutive a a is even then $(aa)^*$ will also be even.

5) strings of length two or more

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

$L = \{ aa, ab, ba, bb, aaa, aab, aba, \ldots\ldots \}$

To represent the first symbol of w

$r_1 = (a+b)$

To represent the second symbol of w

$r_2 = (a+b)$

Thus the Regular Expression for the language is

$$r = (a+b)(a+b)(a+b)^*$$

6) $L = \{ a^n b^m \mid n+m \text{ is even} \}$

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

$L = \{ \in, ab, ba, aabb, abbb, aaab, \ldots\ldots \}$

Category 1 :- n and m are even.

To represent the first part of pattern

$r_a = (aa)^*$

To represent the even number of b's

$r_b = (bb)^*$

$$r_1 = r_a r_b = (aa)^* (bb)^*$$

Category 2 : n and m are odd

To represent odd number of a

$r_a = a(aa)^*$

To represent odd number of b.

$r_b = b(bb)^*$

$$r_2 = r_a r_b = a(aa)^* b(bb)^*$$

$r = r_1 + r_2$

$$r = (aa)^* (bb)^* + a(aa)^* b(bb)^*$$

7) strings containing exactly one a.

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

$L = \{ a, ab, ba, abb, bab, bba, \ldots \}$

To represent the string containing with a.

$$r = b^* a b^*$$

8) strings containing atleast two b.

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

$L = \{ bb, abb, bab, bba, \ldots \}$

Thus the Regular Expression for the language L is given by

$$r = (a+b)^* b (a+b)^* b (a+b)^*$$

9) $n_a(w) \bmod 3 = 0$

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

$L = \{\epsilon, aaa, b, bb, aaab, baaa, aaaaaa, \ldots\}$

Thus the Regular Expression for the language is given by.

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

## Operations on Regular Expressions

(i) union

(ii) concatenation

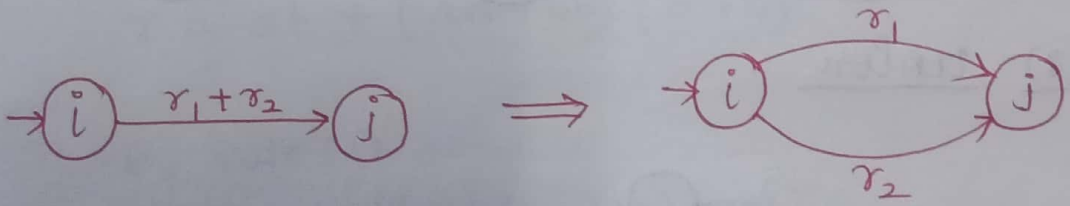
(iii) closure

# Connection Between Regular Expressions and Regular Languages

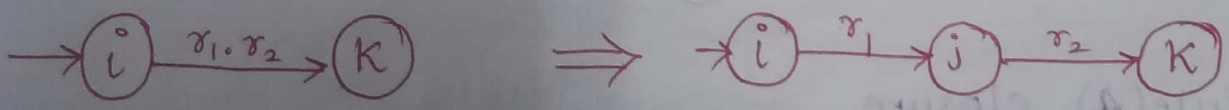
- Let 'r' be a regular Expression describing the language, then there is an equivalent FSA M such that $L(M) = L(r)$

**Rule 1 :-** Create two states one is initial state and another one is final state. Make a transition from initial state to final state and place the regular Expression r in between them.


**Rule 2:** If r has union (+) then make parallel transitions between two states.




**Rule 3:** If r has concatenation (.) then insert 1 more stable state inbetween two states. then $r_1 r_2$ becomes regular Expression.




**Rule 4 :** If r has closure, then loop will be added to next state and mark $\varepsilon$ (Epsilon) between two states.

1) Construct FSA for given expression.

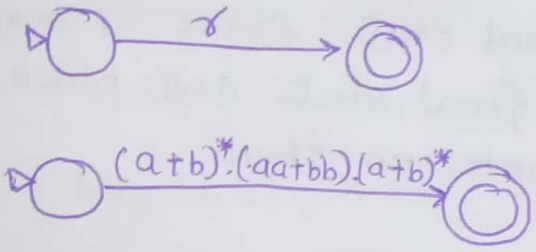
$$r = (a+b)^*\ (aa+bb)\ (a+b)^*$$

Soln

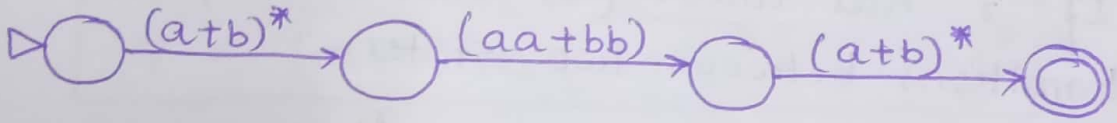
Given

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

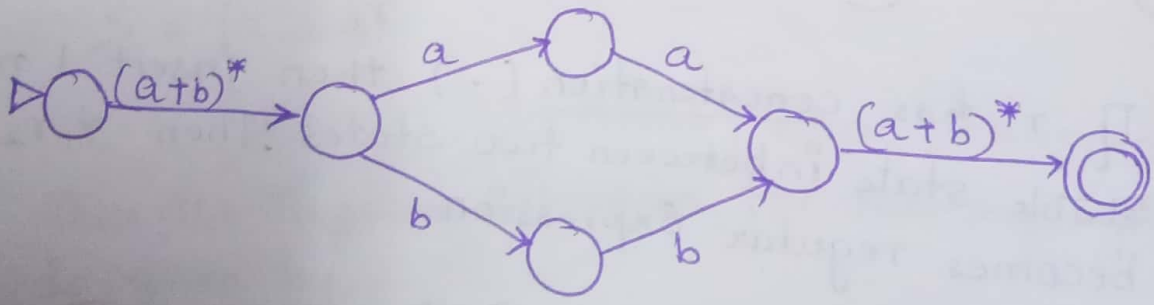
By rule (1)



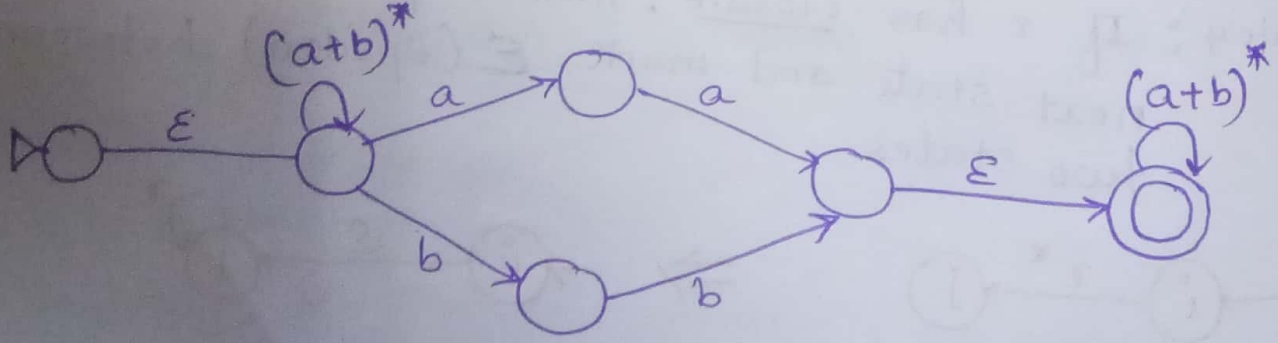

By Rule (3) concatenation



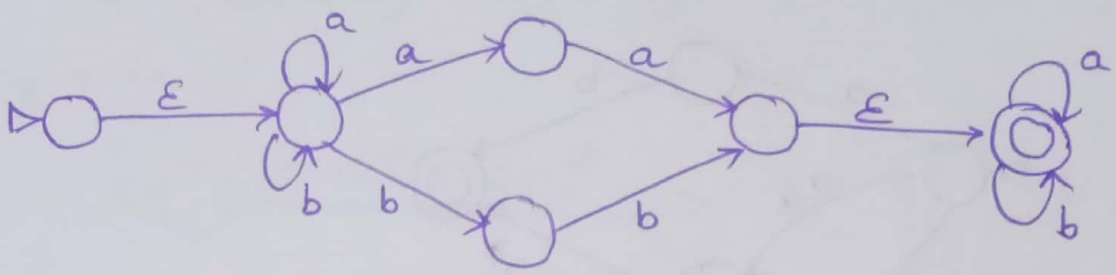

By Rule (2) union



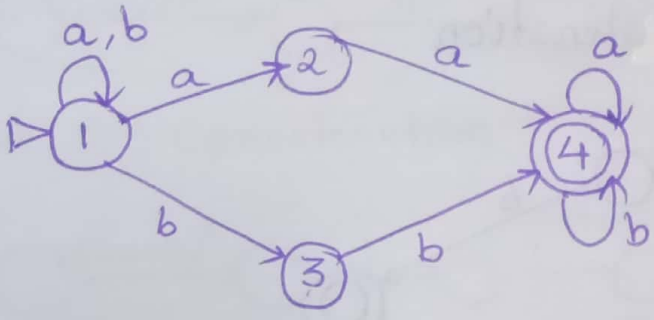

By Rule (4) closure



By Rule(2) union




After optimising ε-transitions

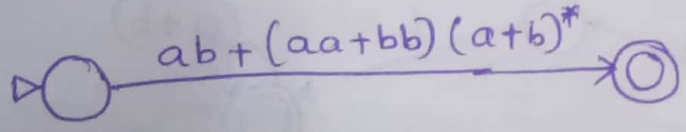



2) Construct a FSA from the regular expression

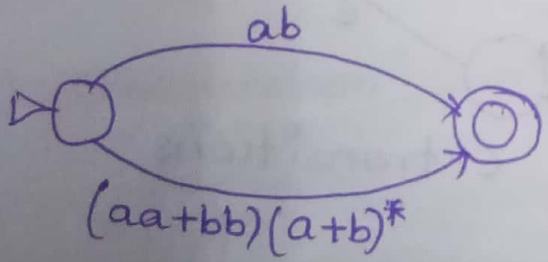
$$r = ab + (aa+bb)(a+b)^*$$

Soln

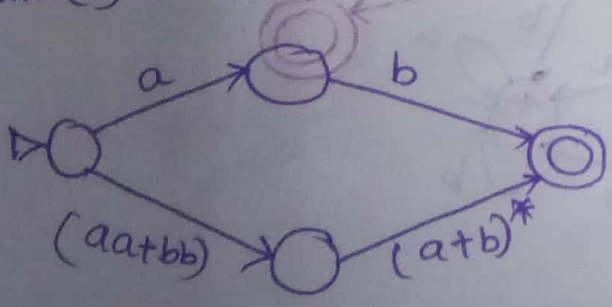
By rule (1)



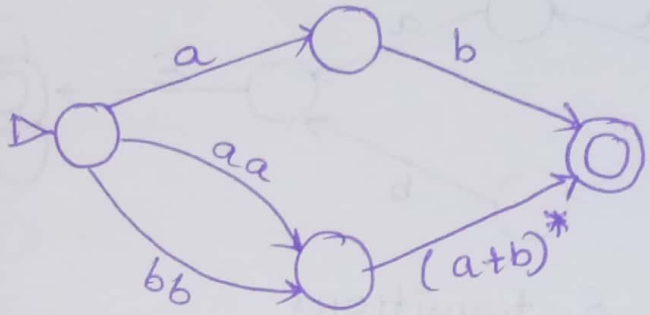

By rule (2): union



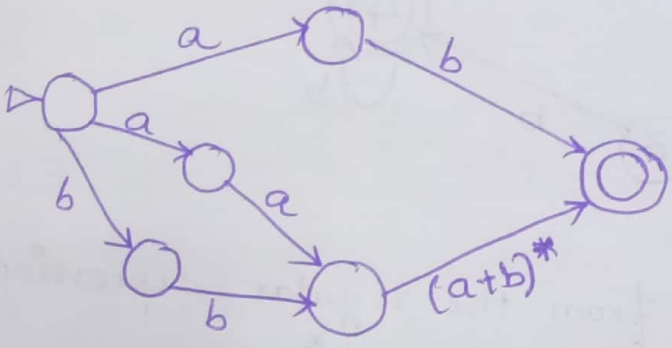

By rule (3): concatenation



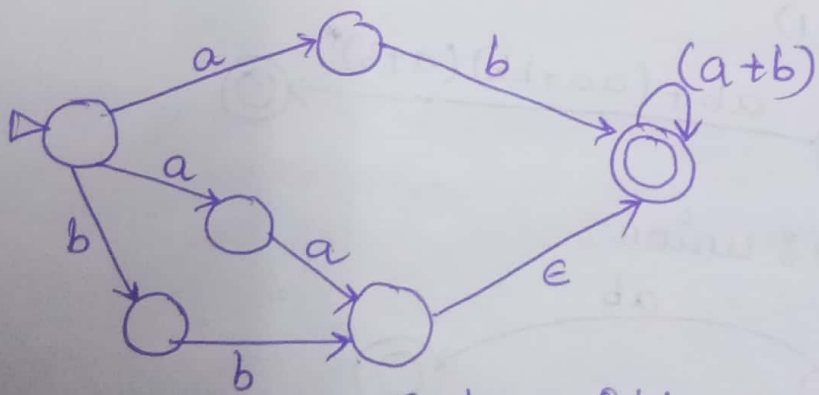
By rule (2) : union



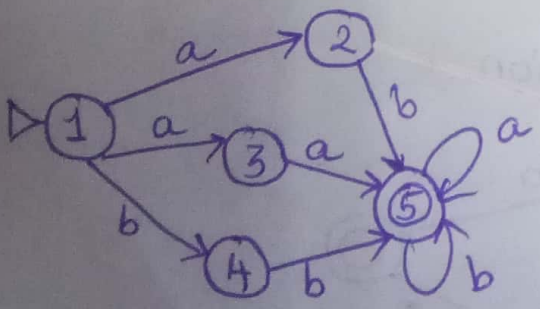

By rule (3) : Concatenation




By rule (4) :- closure



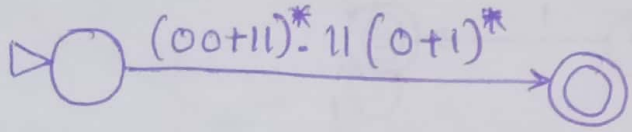

After optimising ϵ-transitions



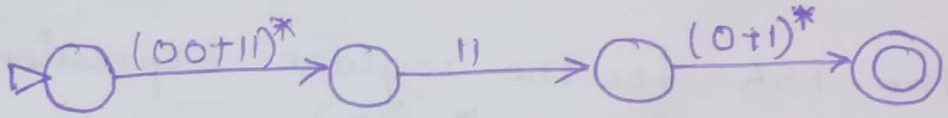
3) construct a FSA

$$r = (00+11)^* . 11(0+1)^*$$

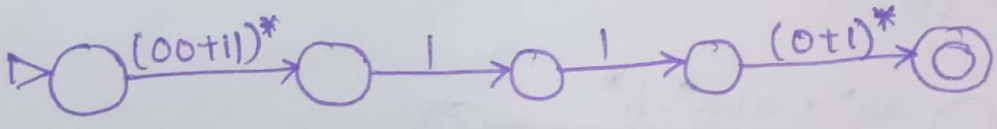
By rule (1)



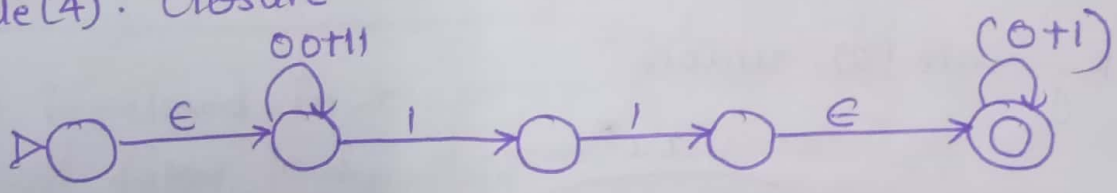

By rule (2). concatenation



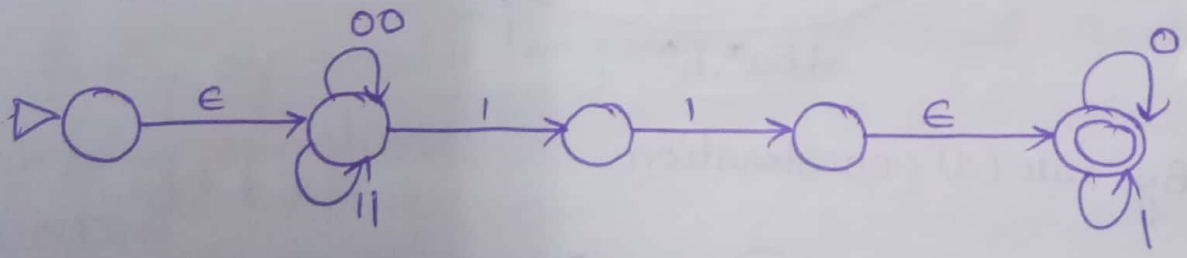

By rule (3) : Concatenation



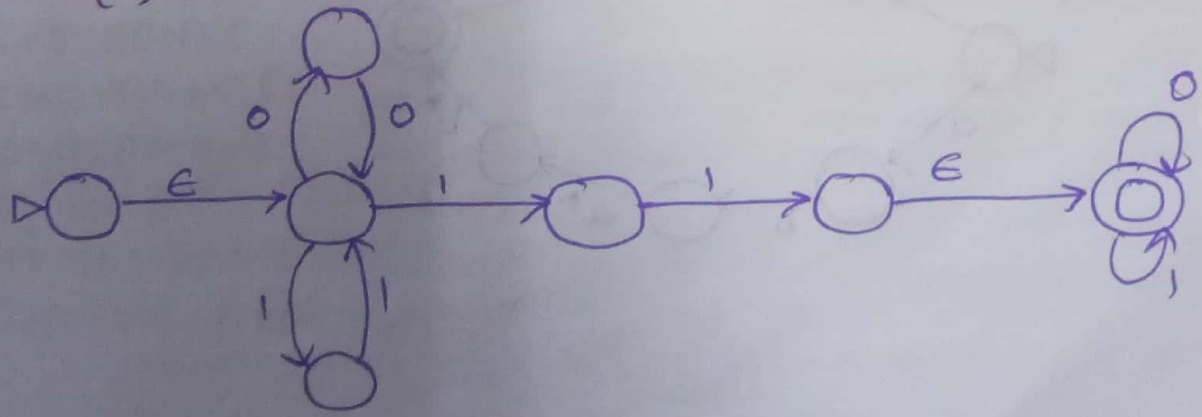

By rule (4) : closure



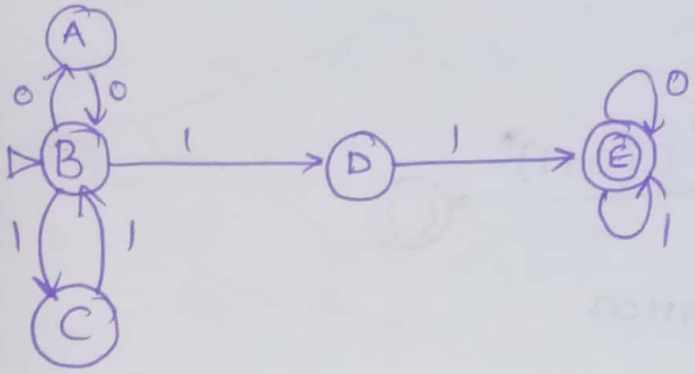

By rule (5) : union




By rule (6) : Concatenation



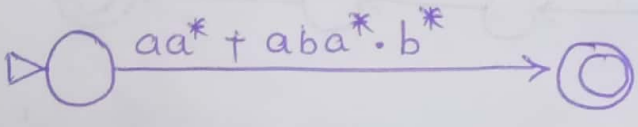
After optimising ε-transitions



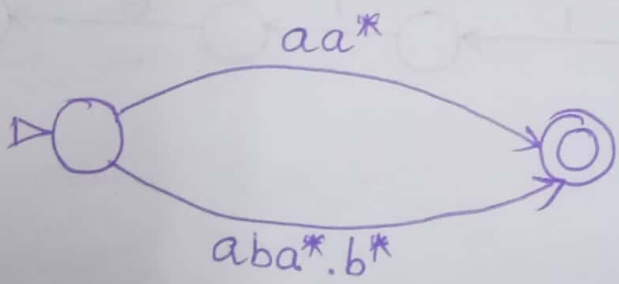

4) construct the FSA for the regular expression

$$r = (aa^* + aba^* \cdot b^*)$$

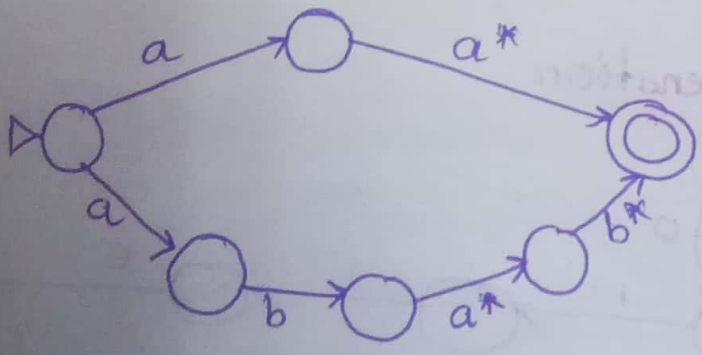
By rule (1)



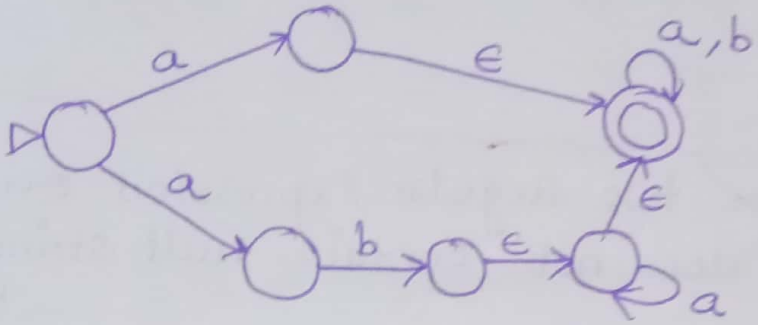

By rule (2). union



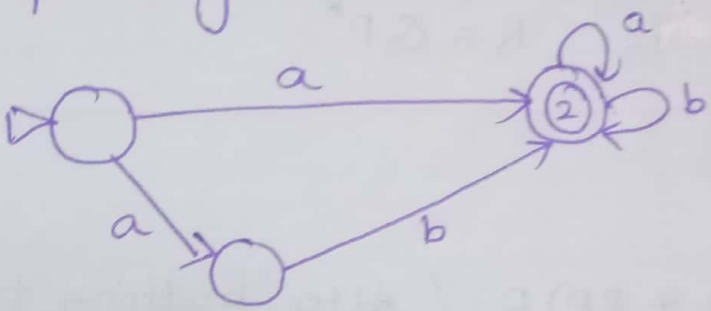

By rule (3) concatenation



By rule (4) : closure




After optimising ϵ-transitions




## Closure properties of Regular languages

A set is closed in operation if and only if the operation takes two different elements of set and produces elements of same.

A family of regular languages are closed under following operations.

1) UNION
2) INTERSECTION
3) DIFFERENCE
4) CONCATENATION
5) COMPLEMENT
6) CLOSURE
7) HOMOMORPHISM
8) INVERSE HOMOMORPHISM

# Construction of Regular Expression from DFA.

## Ardens Theorem

Let P and Q be two Regular Expression over alphabet $\Sigma$. If P does not contain null string $\epsilon$, then

$R = Q + RP$ has a unique solution that is $R = QP^*$

Proof :-

$R = Q + (Q + RP)P$ ( After putting the value $R = Q + RP$)

$= Q + QP + RPP$

When we put the value of R recursively again and again, we get the following eq?.

$R = Q + QP + QP2 + QP3 \ldots$

$R = Q(\epsilon + P + P_2 + P3 + \ldots)$

$R = QP^*$ [As $P^*$ represents $(\epsilon + P + P_2 + P_3 + \ldots)$

Hence proved

## Assumptions for applying Ardens Theorem

- The transition diagram must not have NULL transitions
- It must have only one initial state.

## Method

**Step 1** :- create equations as the following form for all the states of the DFA having n states with initial state $q_1$.

$$q_1 = q_1R_{11} + q_2R_{21} + \dots + q_nR_{n1} + \varepsilon$$

$$q_2 = q_1R_{12} + q_2R_{22} + \dots + q_nR_{n2}$$

$$\dots\dots\dots$$

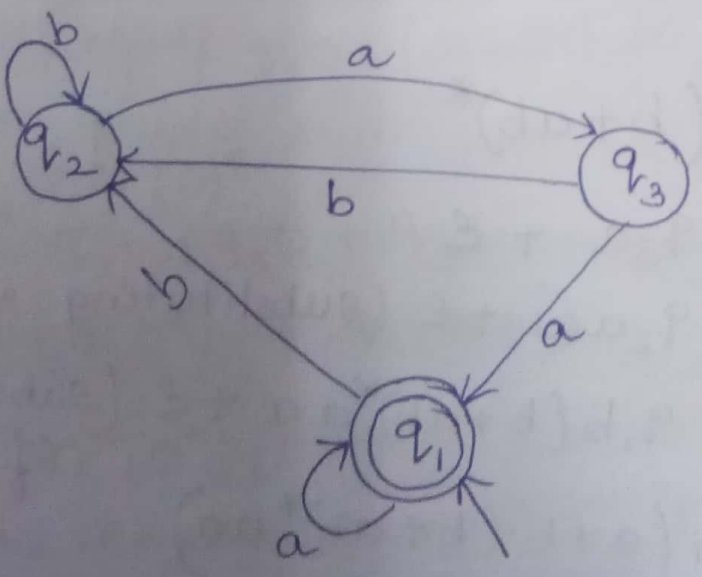
$$\dots\dots\dots$$

$$q_n = q_1R_{1n} + q_2R_{2n} + \dots + q_nR_{nn}$$

$R_{ij}$ represents the set of labels of edges from $q_i$ to $q_j$, if no such edge exists, then $R_{ij} = \phi$

**Step 2** : solve these equations to get the equation for the final state interms of $R_{ij}$.

Q) Construct a Regular Expression corresponding to the automata given below.



Solution

Here the initial state is $q_1$ and the final state is $q_1$

The equations for the three states $q_1$, $q_2$ and $q_3$ are as follows

$$q_1 = q_1a + q_3a + \varepsilon$$ ($\varepsilon$ move is because $q_1$ is the initial state)

$$q_2 = q_1b + q_2b + q_3b$$

$$q_3 = q_2a$$

Now, we will solve these three equations

$$q_2 = q_1b + q_2b + q_3b$$

$$= q_1b + q_2b + (q_2a)b$$ (substituting value of $q_3$)

$$= \underbrace{q_1b}_{Q} + \underbrace{q_2}_{R}\underbrace{(b+ab)}_{P}$$

Applying Arden's Theorem $R = Q + RP$, $q_2$ becomes

$$q_2 = q_1b(b+ab)^*$$

$$q_1 = q_1a + q_3a + \varepsilon$$

$$= q_1a + q_2aa + \varepsilon$$ (substituting value of $q_3$)

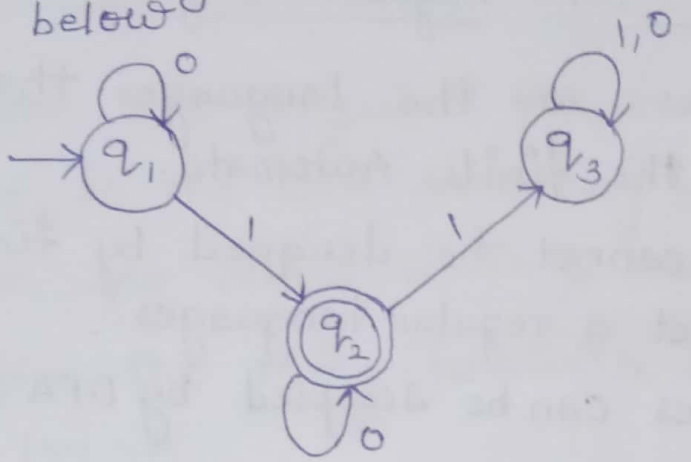
$$= q_1a + q_1b(b+ab)^*aa + \varepsilon$$ (substituting value of $q_2$)

$$= \underbrace{\varepsilon}_{Q} + \underbrace{q_1}_{R}\underbrace{(a+b(b+ab)^*aa)}_{P}$$

$$= \varepsilon(a+b(b+ab)^*aa)^*$$

Hence, the Regular Expression is $(a+b(b+ab)^*aa)^*$

Q) Construct a Regular Expression to the automata given below




Solution

Here the initial state is $q_1$ and the final state is $q_2$

The equations for the three states $q_1$, $q_2$ and $q_3$ are as follows.

$$q_1 = q_1 0 + \varepsilon$$

$$q_2 = q_1 1 + q_2 0$$

$$q_3 = q_2 1 + q_3 0 + q_3 1$$

Now, we solve these three equations.

$$q_1 = \underset{R}{q_1}\underset{P}{0} + \underset{Q}{\varepsilon}$$

$$q_1 = \varepsilon 0^* = 0^*$$

$$q_2 = \underset{Q}{0^* 1} + \underset{R}{q_2}\underset{P}{0} \quad \text{(substituting value of } q_1\text{)}$$

$$q_2 = 0^* 1 (0)^*$$

$$q_3 = q_2 1 + q_3 0 + q_3 1$$

$$= \underset{Q}{0^* 1 0^* 1} + \underset{R}{q_3}\underset{P}{(0+1)}$$

$$q_3 = 0^* 1 0^* 1 (0+1)^*$$

Hence the Regular Expression is $0^* 1 0^* 1 (0+1)^*$

# Pumping lemma for Regular Languages

- Regular languages are the languages that can be designed by the finite Automata.
- Some languages cannot be designed by finite automata are not a regular languages.
- Regular languages can be accepted by DFA, NFA and ε-NFA.
- Pumping lemma is most powerful technique to show that certain language is not regular.

The concept of Pumping lemma is

1) Taking a substring from a given string.
2) Pumping it for certain number of times.
3) Is it possible to generate another string of the same language.

## Theorem

1) Let L be a regular language
2) Then there exists a constant 'n' such that for every string w in L such that $|w| \geq n$,
3) We can break w into three strings, $w = xyz$, such that
   1. $|y| > 0$
   2. $|xy| \leq n$
   3. for all $i \geq 0$, the string $xy^iz$ is also in L.

## Procedure to apply lemma

1) Assume L is Regular. Assume 'n' - number of states in the DFA accepting L.
2) choose any string w in L with $|w| >= n$.
3) Partition the string w with xyz with $|xy| <= n$ and $|y| > 0$.
4) Find the value of i, such that $xy^iz$ is not in L.

1) Prove that the language $L = \{a^i b^i \mid i \geq 1\}$ is not regular.

Soln

Given $L = \{a^i b^i \mid i \geq 1\}$

1) Assume L is regular, and
2) Let n be the number of states in the DFA accepting lemma L. (pumping length)
3) Let $w = a^n b^n$ $\quad |a^n b^n| >= 2n > n$
4) By Pumping lemma

$w = xyz$ with $|xy| <= n$ and $y >= 0$

There will be three cases

I) y contains some number of a's only, $y = a^k$
II) y contains some number of b's only, $y = b^m$
III) y contains both a's and b's, $y = a^k b^m$

case I) Let $Y = a^K$

$W = XYZ$

$W = a^{n-K} \cdot a^K \cdot b^n$ $(X = a^{n-K}, Y = a^K, Z = b^n)$

considering $i = 0$

$XY^iZ = XY^0Z = a^{n-K} \cdot b^n \notin L$, as $(n-K) \neq n$

case II) Let $Y = b^m$

$W = XYZ$

$W = a^n b^m b^{n-m}$ $(X = a^n, Y = b^m, Z = b^{n-m})$

considering $i = 0$

$XY^iZ = XY^0Z = a^n b^{n-m} \notin L$, as $(n-m) \neq n$

case III) Let $Y = a^K b^m$

$W = XYZ$

$W = a^{n-K} a^K b^m b^{n-m}$ $(X = a^{n-K}, Y = a^K b^m, Z = b^{n-m})$

considering $i = 0$,

$XY^iZ = XY^0Z = a^{n-K} b^{n-m} \notin L$,

for all the above 3 cases $W = XY^iZ \notin L$

This is the contradiction to the assumption made in step 1.

Therefore L is not a Regular language.

2) Prove that the language $L = \{0^n 1^{2n} \mid n \geq 1\}$ is not regular.

Soln

Given $L = \{0^n 1^{2n} \mid n \geq 1\}$

1) Assume L is regular, and

2) Let c be the number of states in the DFA accepting L. (pumping length)

3) Let $w = 0^c 1^{2c}$ $\quad |0^c 1^{2c}| \geq 3c > c$

4) By pumping lemma

$w = xyz$ with $|xy| \leq c$ & $y \geq 0$

There will be three cases.

I) y contains some number of 0's only, $y = 0^k$

II) y contains some number of 1's only, $y = 1^m$

III) y contains both 0's and 1's, $y = 0^k 1^m$.

Case I) Let $y = 0^k$

$w = xyz$

$w = 0^{c-k} \underbrace{0^k}_{y} 1^{2c}$ $\quad (x = 0^{c-k},\ y = 0^k,\ z = 1^{2c})$

considering $i = 0$

$xy^iz = xy^0z = 0^{c-k} 1^{2c} \notin L$

Case II) Let $y = 1^m$

$w = 0^c \underbrace{1^m}_{y} 1^{2c-m}$

considering $i = 0$

$xy^iz = xy^0z = 0^c 1^{2c-m} \notin L$

Case III) Let $Y = 0^k 1^m$

$w = 0^{c-k} \cdot \underbrace{0^k \cdot 1^m}_{y} \cdot 1^{2c-m}$

considering $i = 0$

$xy^iz = xy^0z = 0^{c-k} 1^{2c-m} \notin L$

for all the above cases $w = xy^iz \notin L$

This is the contradiction to the assumptions made in step(1)

Therefore L is not a Regular language.

3) Prove that the language $L = \{a^n b a^n \mid n \geq 1\}$ is not regular.

Soln 1) Assume L is regular

2) Let M be the number of states in the DFA accepting L.

3) Let $w = a^m b a^m$, $|a^m b a^m| = 2m+1 > m$

4) By Pumping lemma, w can be written as, $w = xyz$ with $|xy| \leq m$ and $|y| > 0$

The cases are as follows.

Case(i) y contains some number of a's alone in first part, $y = a^i$ $1 \leq i \leq m$

Case(ii) y contains some number of a's alone in second part $y = a^i$, $1 \leq i \leq m$

case(iii) y contains some number of a's from first part and also b, $y = a^i b$, $1 \leq i \leq m$

case(iv) y contains b and some number of a's from Second part $y = ba^i$, $1 \leq i \leq m$

case(v) Y contains some a's from first part, along with b and also some a's from part second.

$$y = a^i b a^j \quad , \quad 1 \leq i, j \leq m$$

case(i) Let $y = a^i$

$$w = \underbrace{a^{m-i}}_{x} \underbrace{a^i}_{y} \underbrace{b a^m}_{z}$$

Considering $i = 0$

$$xy^iz = xy^0z = a^{m-i} b a^m \notin L$$

case(ii) Let $y = a^i$

$$w = \underbrace{a^m b}_{x} \underbrace{a^i}_{y} \underbrace{a^{m-i}}_{z}$$

Considering $i = 0$

$$xy^iz = xy^0z = a^m b a^{m-i} \notin L$$

case(iii) Let $y = a^i b$

$$w = \underbrace{a^{m-i}}_{x} \underbrace{a^i b}_{y} \underbrace{a^m}_{z}$$

considering $i = 0$

$$xy^iz = xy^0z = a^{m-i} a^m \notin L$$

case(iv) Let $y = ba^i$

$$w = \underbrace{a^m}_{x} \underbrace{b a^i}_{y} \underbrace{a^{m-i}}_{z}$$

Considering $i = 0$

$$xy^iz = xy^0z = a^m a^{m-i} \notin L$$

Case(v) Let $y = a^i b a^j$

$$w = \underbrace{a^{m-i}}_{x} \underbrace{a^i b a^j}_{y} \underbrace{a^{m-j}}_{z}$$

considering $i = 0$

$$xy^iz = xy^0z = a^{m-i} a^{m-j} \notin L$$

Case (vi) $y = b$

$$w = xyz = \underset{x}{a^m}\ \underset{y}{b}\ \underset{z}{a^m}$$

Considering $i = 0$,

$$xy^iz = xy^0z = a^m b^m \notin L$$

for all the above cases

$$w = xyz = xy^iz \notin L$$

This is the contradiction to the assumption we made in step (1), Hence L is not a regular language.

4) Show that $L = \{ww \mid w \in \{a,b\}^*\}$ is not regular.

Soln 1) Assume L is regular

Let n be a number of states in DFA accepting L

2) Let $P = \underbrace{a^n b}_{w_1}.\underbrace{a^n b}_{w_2}$ $\quad |a^n b\, a^n b| = 2n+1 > n$

3) By Pumping lemma, w can be written as

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

The cases are as follows

case(i) : y contains some number of a's from $w_1$,

$y = a^i$

case(ii) : y contains some number of a's from $w_2$,

$y = a^j$

case(iii) : y contain some number a along with b in $w_1$ and some number of a's in $w_2$

$y = a^i b\, a^j$

case (i) $y = a^i$

$$w = xyz = \underbrace{a^{n-i}}_{x} \cdot \underbrace{a^i}_{y} \cdot \underbrace{b\, a^n b}_{z}$$

Considering $i = 0$

$$xy^iz = xy^0z = a^{n-i} \cdot b \cdot a^n b \notin L$$

case (ii) $y = a^j$

$$w = \underbrace{a^n \cdot b}_{x}\ \underbrace{a^j}_{y}\ \underbrace{a^{n-j} b}_{z}$$

considering $i = 0$

$$xy^iz = xy^0z = a^n \cdot b\, a^{n-j} b \notin L$$

case (iii) $y = a^i b a^j$

$$w = \underbrace{a^{n-i}}_{x} \cdot \underbrace{a^i\, b\, a^j}_{y}\ \underbrace{a^{n-j} b}_{z}$$

Considering $i = 0$

$$xy^iz = xy^0z = a^{n-i} \cdot a^{n-j} b \notin L$$

for all the above cases

$$w = xyz = xy^iz \notin L$$

This is the contradiction to the assumption we made in step (1)

Hence, L is not a regular language.

5) Show that the language containing set of all balanced parenthesis is not regular.

Soln i) Assume L is regular.

2) Let n be the number of states in DFA accepting L.

3) Let $w = (^n .)^n$

By Pumping lemma.

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

The cases are as follows.

case (i) : y may contain some number of open parenthesis, $y = (^i$

case (ii) : y may contain some number of closed parenthesis, $y = )^j$

case (iii) : y may contain some open parenthesis's and some closed parathesis's, $y = (^i)^j$

case (i) $y = (^i$

$w = xyz = \underbrace{(^{n-i}}_{x} \underbrace{(^i}_{y} \underbrace{)^n}_{z}$

Considering $i = 0$

$w = xy^iz = xy^0z = (^{n-i})^n \notin L$

case (ii) $y = )^j$

$w = \underbrace{(^n}_{x} \underbrace{)^j}_{y} \underbrace{)^{n-j}}_{z}$

considering $i = 0$

$w = xy^iz = xy^0z = (^n)^{n-j} \notin L$

case (iii) $Y = (^i)^j$

$$w = \underbrace{(^{n-i}}_{X} \underbrace{(^i)^j}_{Y} \underbrace{)^{n-j}}_{Z}$$

Considering $i=0$

$w = xy^iz = xy^0z = (^{n-i})^{n-j} \notin L$

for all the above cases

$w = xy^iz \notin L$

This is the contradiction to the assumption we made in step (1).
Hence L is not a regular language.

6) Show that $L = \{ a^{i!} \mid i \geq 0 \}$ is not regular.

Soln

1) Assume L is regular.
2) Let n be the number of states in DFA accepting L.
3) Let $w = a^{n!}$ $\quad |a^{n!}| \geq n! > n$

By Pumping lemma

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

$w = xyz = a^{n!}$, so $|xyz| = n!$

Let $y = a^K$, $1 \leq K \leq n$

Considering $i=2$

$xy^iz = xy^2z = xyyz = |xyz| + |y| = n! + n$

Thus $(n! + n) \neq (n+1)!$

so, $xy^2z = a^{n!} + a^n = a^{n!+n} \neq a^{(n+1)!} \notin L$

This is the contradiction to the assumptions in step (1)
Hence, L is not a Regular language.

7) Show that $L = \{a^{i^2} \mid i \geq 1\}$ is not regular

Soln

1) Assume L is regular
2) Let n be the number of states in the DFA accepting L.
3) Let $w = a^{n^2}$, $|a^{n^2}| = n^2 > n$

By pumping lemma

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

$w = xyz = a^{n^2}$, so $|xyz| = n^2$

Let $y = a^k$, $1 \leq k \leq n$

Considering $i = 2$

$|xy^iz| = |xy^2z| = |xyz| + |y|$, so $|xy^2z| > n^2$

Considering Again

$|xy^2z| = |x| + |y| + |z| + |y| = |xyz| + |y|$

$= n^2 + n$

$$\text{So, } n^2 < |xy^2z| \leq (n^2+n)$$

$$n^2 < |xy^2z| \leq (n^2+n) < (n+1)^2$$

Since the length of $xy^2z$ is strictly lies between $n^2$ and $(n^2+n)$. As per the language constraint $L = \{a^{n^2} \mid n \geq 1\}$ which is perfect square. In between square of two consecutive integer there is no square of perfect square exists.

$|xy^2z| \notin L$.

This is the contradiction made in step (1)

∴ L is not a regular language.

# Finite State Machine

- Finite state machine is a machine that takes a strings of symbol as input and changes its state accordingly.
- Finite Automata without output are DFA, NFA and $\epsilon$-NFA.
- Finite Automata with output are Mealy Machine and Moore Machine.

## Moore Machine

- Moore Machine is an FSM whose output depends on only the present state.
- A Moore Machine can be described by a 6-tuple.

$$(Q, \Sigma, \Delta, \delta, \lambda, q_0)$$

Where

$Q$ is a finite set of states

$\Sigma$ is a finite set of symbols called the input alphabet.

$\Delta$ is a finite set of symbols called the output alphabet

$\delta$ is the input transition function where

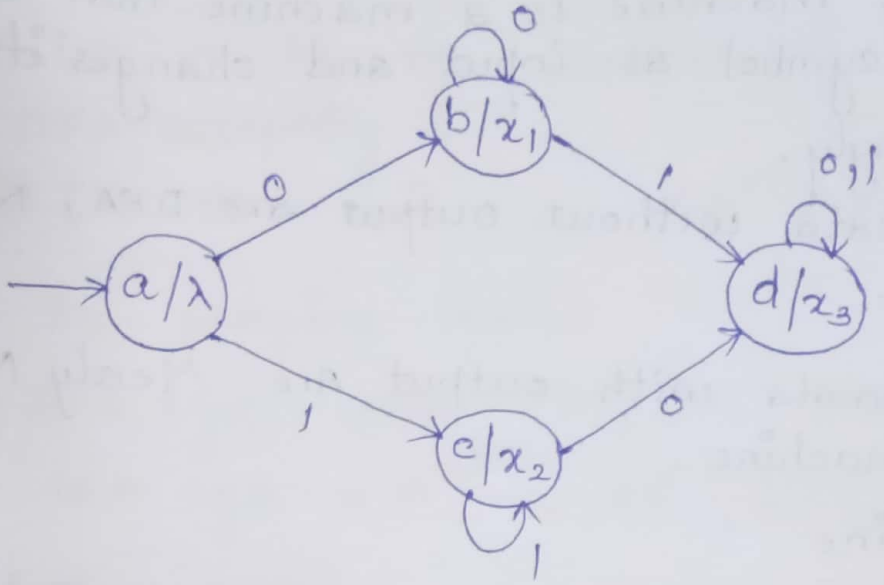
$$\delta : Q \times \Sigma \longrightarrow Q$$

$\lambda$ is the output transition function where

$$\lambda : Q \longrightarrow \Delta$$

$q_0$ is the initial state from where any input is processed ($q_0 \in Q$)

# Representation of Moore Machine




$M = (Q, \Sigma, \Delta, \delta, \lambda, q_0)$

$Q = \{a, b, c, d\}$

$\Sigma = \{0, 1\}$

$\Delta = \{\lambda, x_1, x_2, x_3\}$

$q_0 = \{a\}$

$\lambda = a \longrightarrow \lambda$

$b \longrightarrow x_1$

$c \longrightarrow x_2$

$d \longrightarrow x_3$

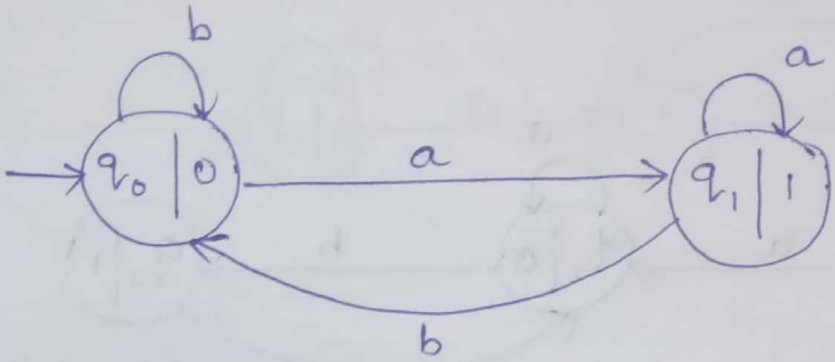
$\delta$ is defined by

| Present state | Next state | | output |
|---|---|---|---|
| | I/p = 0 | I/p = 1 | |
| → a | b | c | λ |
| b | b | d | $x_1$ |
| c | d | c | $x_2$ |
| d | d | d | $x_3$ |

1) Construct a Moore Machine that takes string over a and b input and prints 1 when number of a's occur in the string.

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

$\Delta = \{0, 1\}$




$Q = \{q_0, q_1\}$

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

$\Delta = \{0, 1\}$

$q_0 = \{q_0\}$

$\lambda = q_0 \rightarrow 0$

$q_1 \rightarrow 1$

$\delta$ is defined by

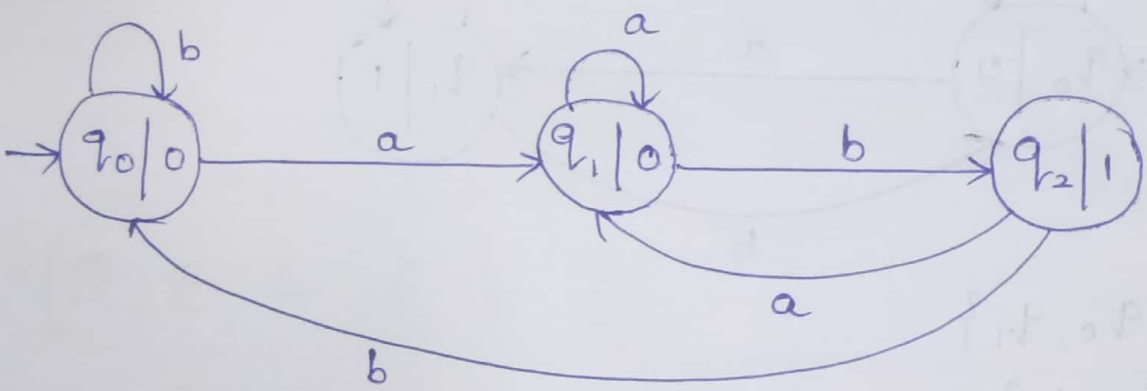
| Present state | Next state | | output |
|---|---|---|---|
| | i/p = a | i/p = b | |
| $\rightarrow q_0$ | $q_1$ | $q_0$ | 0 |
| $q_1$ | $q_1$ | $q_0$ | 1 |

2) Construct a Moore Machine that takes a string over (a, b) as input and print 1 as output for every string occurrence of 'ab' as a substring.

Soln

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

$\Delta = \{0, 1\}$




$Q = \{q_0, q_1, q_2\}$

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

$\Delta = \{0, 1\}$

$q_0 = \{q_0\}$

$\lambda = \{ q_0 \rightarrow 0$

$q_1 \rightarrow 0$

$q_2 \rightarrow 1 \}$

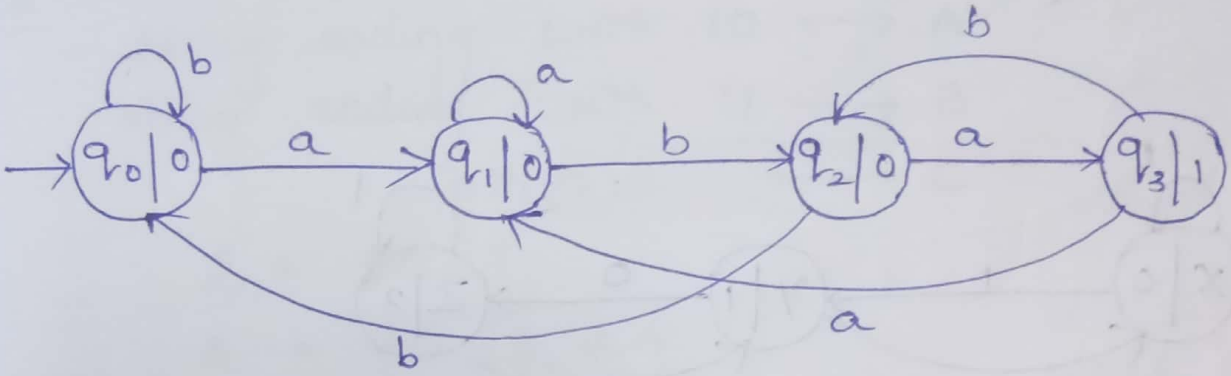
$\delta$ is defined by

| Present state | Next state | | output |
|---|---|---|---|
| | i/p = a | i/p = b | |
| → $q_0$ | $q_1$ | $q_0$ | 0 |
| $q_1$ | $q_1$ | $q_2$ | 0 |
| $q_2$ | $q_1$ | $q_0$ | 1 |

3) construct a Moore machine which found the occurrence of 'aba' in the given input string.

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

$\Delta = \{0, 1\}$




$Q = \{q_0, q_1, q_2, q_3\}$

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

$\Delta = \{0, 1\}$

$q_0 = \{q_0\}$

$\lambda = q_0 \rightarrow 0$

$q_1 \rightarrow 0$

$q_2 \rightarrow 0$

$q_3 \rightarrow 1$

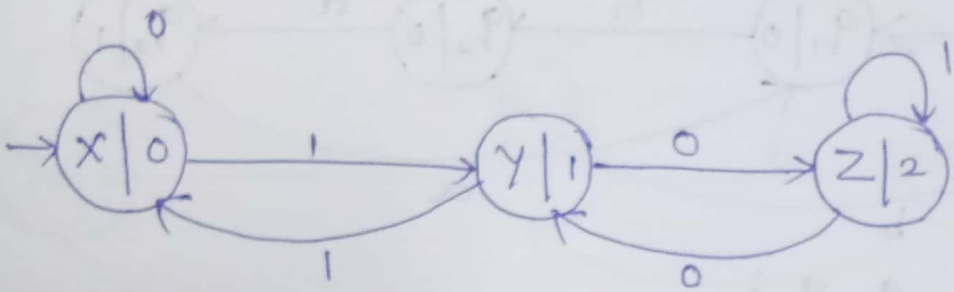
$\delta$ is defined by

| Present state | Next state | | output |
|---|---|---|---|
| | i/p = a | i/p = b | |
| $\rightarrow q_0$ | $q_1$ | $q_0$ | 0 |
| $q_1$ | $q_1$ | $q_2$ | 0 |
| $q_2$ | $q_3$ | $q_0$ | 0 |
| $q_3$ | $q_1$ | $q_2$ | 1 |

4) Construct a Moore Machine with alphabet 0 and 1. The output after reading x is a remainder where x is the number whose binary representation is divisible by 3.

Soln $\Sigma = \{0,1\}$

$\Delta = \{0,1,2\}$




$Q = \{X, Y, Z\}$

~~$\Sigma$~~ $= \{0,1\}$

$\Delta = \{0,1,2\}$

$q_0 = \{X\}$

$\lambda = X \longrightarrow 0$

$Y \longrightarrow 1$

$Z \longrightarrow 2$

$\delta$ is defined by :

| Present state | Next state | | output |
|---|---|---|---|
| | i/p = 0 | i/p = 1 | |
| → X | X | Y | 0 |
| Y | Z | X | 1 |
| Z | Y | Z | 2 |

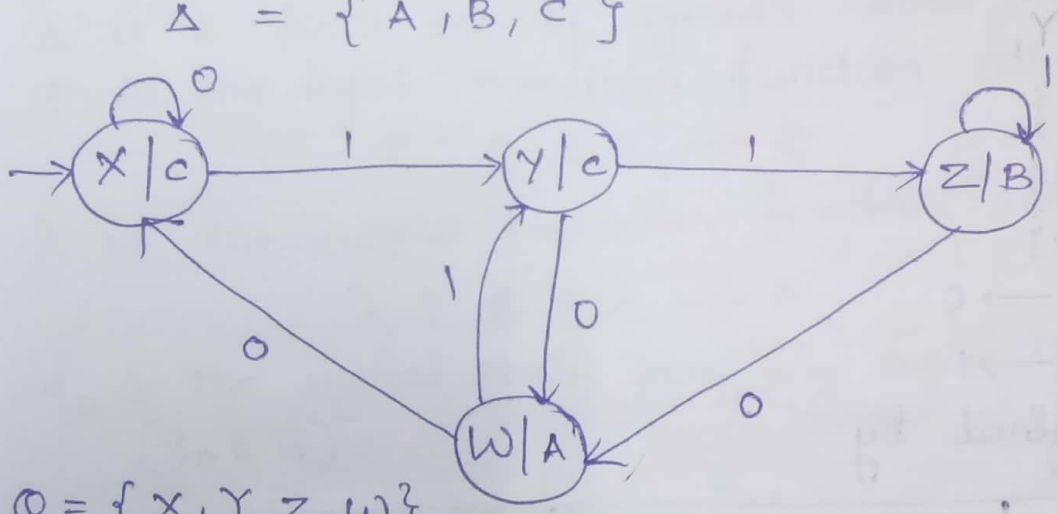
5) construct a Moore Machine that takes set of strings over 0 and 1 and that produces the A as output if the input string is ending with 10 or produce B as output if the input string is ending with 11 otherwise will produce C as output.

Soln

string ending with 10 ⟶ A
string ending with 11 ⟶ B
others ⟶ C

$\Sigma = \{0, 1\}$

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




$Q = \{X, Y, Z, W\}$

$\Sigma = \{0, 1\}$

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

$q_0 = \{X\}$

$\delta$ is defined by

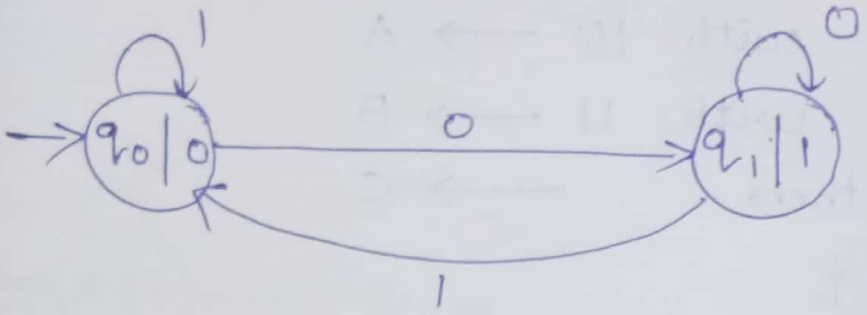
| Present state | Next state i/p=0 | Next state i/p=1 | output |
|---|---|---|---|
| → X | X | Y | C |
| Y | W | Z | C |
| Z | W | Z | B |
| W | X | Y | A |

6) Construct a Moore Machine for Producing 1's Complement for a given binary string.

Soln

$\Sigma = \{0, 1\}$

$\Delta = \{0, 1\}$




$Q = \{\ ,\ ,$

$\Sigma = \{0, 1\}$

$\Delta = \{0, 1\}$

$q_0 = \{q_0\}$

$\lambda = q_0 \longrightarrow 0$

$q_1 \longrightarrow 1$

$\delta$ is defined by

| Present state | Next state | | output |
|---|---|---|---|
| | i/p = 0 | i/p = 1 | |
| $\rightarrow q_0$ | $q_1$ | $q_0$ | 0 |
| $q_1$ | $q_1$ | $q_0$ | 1 |

# Mealy Machine

A Mealy Machine is a finite state Machine whose output depends on the present state as well as the present input.

- A Mealy Machine can be described by 6-tuple

$$(Q, \Sigma, \Delta, \delta, \lambda, q_0)$$

where

$Q$ is a finite set of states

$\Sigma$ is a finite set of symbols called the input alphabet

$\Delta$ is a finite set of symbols called the output alphabet

$\delta$ is the input transition function where

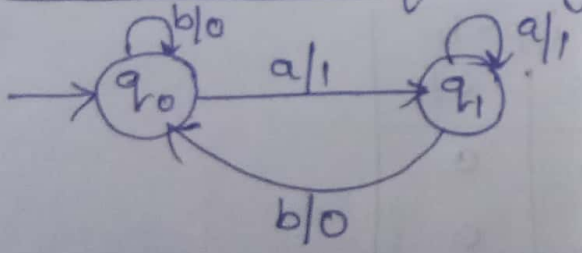
$$\delta : Q \times \Sigma \rightarrow Q$$

$\lambda$ is the output transition function where

$$\lambda : Q \times \Sigma \rightarrow \Delta$$

$q_0$ is the initial state from where any input is processed ($q_0 \in Q$).

## Representation of Mealy Machine




$Q = \{q_0, q_1\}$

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

$\Delta = \{0, 1\}$

$\lambda = (q_0, a) \rightarrow 1$

$(q_0, b) \rightarrow 0$

$(q_1, a) \rightarrow 1$

$(q_1, b) \rightarrow 0$

$\delta$ is defined by

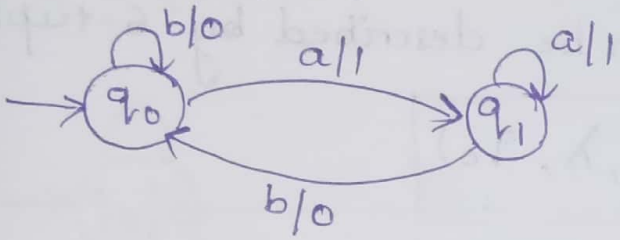
| Present state | i/p = a | | i/p = b | |
|---|---|---|---|---|
| | Next state | o/p | Next state | o/p |
| $\rightarrow q_0$ | $q_1$ | 1 | $q_0$ | 0 |
| $q_1$ | $q_1$ | 1 | $q_0$ | 0 |

1) Construct a Mealy Machine that takes string over a and b input and prints 1 when number of a's occur in the string.

Soln

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

$\Delta = \{0, 1\}$




$Q = \{q_0, q_1\}$

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

$\Delta = \{0, 1\}$

$q_0 = \{q_0\}$

$\lambda = (q_0, a) \rightarrow 1$

$(q_0, b) \rightarrow 0$

$(q_1, a) \rightarrow 1$

$(q_1, b) \rightarrow 0$

$\delta$ is defined by

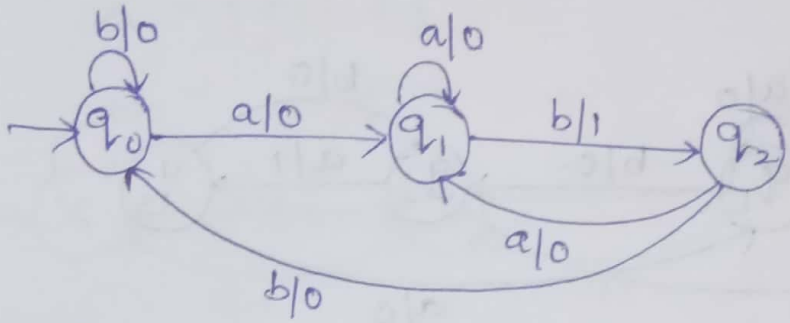
| Present state | i/p = a | | i/p = b | |
|---|---|---|---|---|
| | N/S | O/P | NS | O/P |
| $\rightarrow q_0$ | $q_1$ | 1 | $q_0$ | 0 |
| $q_1$ | $q_1$ | 1 | $q_0$ | 0 |

2) construct a Mealy Machine that takes a string over (a,b) as input and prints 1 as output for every string occurrence of 'ab' as substring.

soln

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

$\Delta = \{0, 1\}$




$Q = \{q_0, q_1, q_2\}$

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

$\Delta = \{0, 1\}$

$q_0 = \{q_0\}$

$\lambda = (q_0, a) \rightarrow 0$ , $(q_1, a) \rightarrow 0$ , $(q_2, a) \rightarrow 0$

$(q_0, b) \rightarrow 0$ , $(q_1, b) \rightarrow 1$ , $(q_2, b) \rightarrow 0$

$\delta$ is defined by

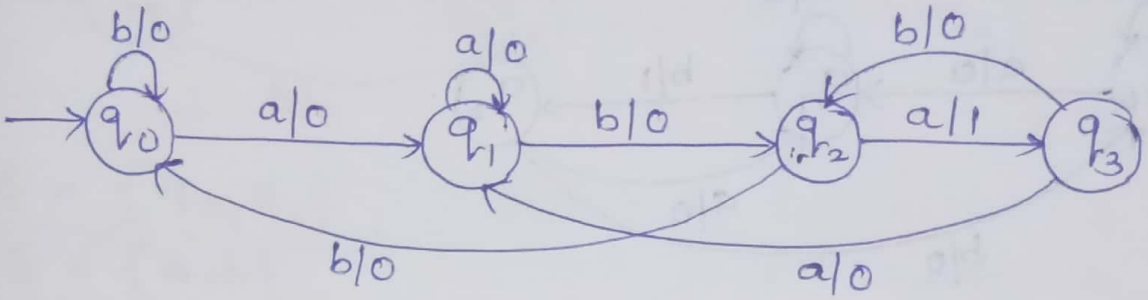
| Present state | i/p = a | | i/p = b | |
|---|---|---|---|---|
| | N.S | O/P | N/S | O/P |
| $\rightarrow q_0$ | $q_1$ | 0 | $q_0$ | 0 |
| $q_1$ | $q_1$ | 0 | $q_2$ | 1 |
| $q_2$ | $q_1$ | 0 | $q_0$ | 0 |

3) Construct a Mealy Machine which found the occurrence of 'aba' in the given string.

Soln

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

$\Delta = \{0, 1\}$




$Q = \{q_0, q_1, q_2, q_3\}$

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

$\Delta = \{0, 1\}$

$q_0 = \{q_0\}$

$\lambda = (q_0, a) \to 0$ , $(q_1, a) \to 0$ , $(q_2, a) \to 1$ ,

$(q_0, b) \to 0$ , $(q_1, b) \to 0$ , $(q_2, b) \to 0$ ,

$(q_3, a) \to 0$

$(q_3, b) \to 0$

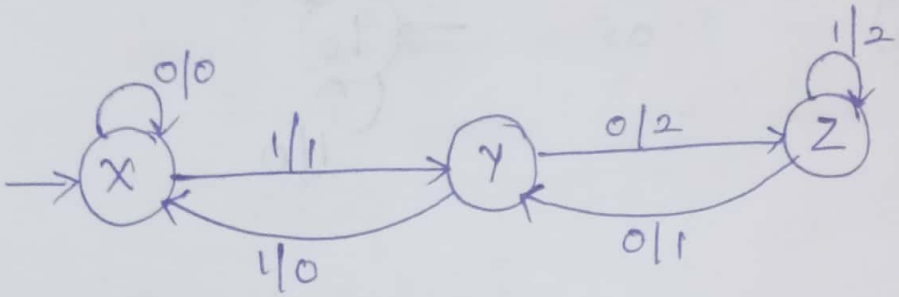
$\delta$ is defined by

| Present State | i/p = a | | i/p = b | |
|---|---|---|---|---|
| | NS | O/P | NS | O/P |
| $q_0$ | $q_1$ | 0 | $q_0$ | 0 |
| $q_1$ | $q_1$ | 0 | $q_2$ | 0 |
| $q_2$ | $q_3$ | 1 | $q_2$ | 0 |
| $q_3$ | $q_1$ | 0 | $q_2$ | 0 |

4) Construct a Mealy Machine with alphabet 0 and 1. The output after reading string X is a remainder where X is the number whose binary representation is divisible by 3.

Soln $\Sigma = \{0, 1\}$

$\Delta = \{0, 1, 2\}$




$Q = \{X, Y, Z\}$

$\Sigma = \{0, 1\}$

$\Delta = \{0, 1, 2\}$

$q_0 = \{X\}$

$\lambda = (X,0) \to 0$ , $(Y,0) \to 2$, $(Z,0) \to 1$

$(X,1) \to 1$ $(Y,1) \to 0$, $(Z,1) \to 2$

$\delta$ is defined by

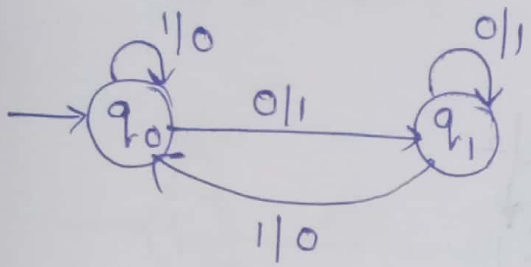
| Present state | i/p = 0 | | i/p = 1 | |
|---|---|---|---|---|
| | N/S | O/P | N/S | O/P |
| → X | X | 0 | Y | 1 |
| Y | Z | 2 | X | 0 |
| Z | Y | 1 | Z | 2 |

5) Construct a Mealy Machine for producing 1's complement of given binary string.

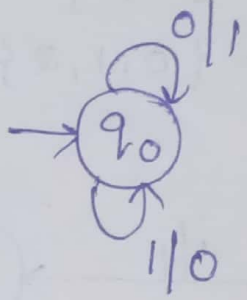
Soln

$\Sigma = \{0, 1\}$

$\Delta = \{0, 1\}$




or




$Q = \{q_0, q_1\}$

$\Sigma = \{0, 1\}$

$\Delta = \{0, 1\}$

$q_0 = \{q_0\}$

$\lambda = (q_0, 0) \rightarrow 1 \quad ; \quad (q_1, 0) \rightarrow 1$

$(q_0, 1) \rightarrow 0 \quad , \quad (q_1, 1) \rightarrow 0$

$\delta$ is defined by

| Present state | i/p = 0 | | i/p = 1 | |
|---|---|---|---|---|
| | N.S | O/P | N/S | O/P |
| $\rightarrow q_0$ | $q_1$ | 1 | $q_0$ | 0 |
| $q_1$ | $q_1$ | 1 | $q_0$ | 0 |

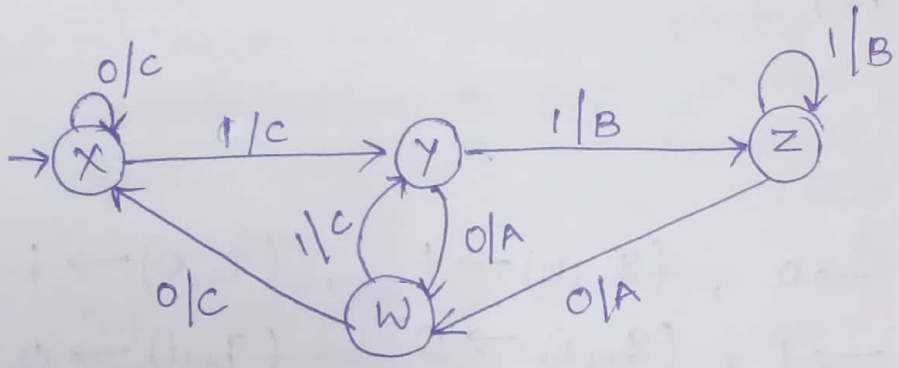
6) construct a Mealy Machine that takes a set of strings over 0 and 1 and that produce the A as o/p if the input string is ending with 10 or produce B if the input string is ending with 11 otherwise will produce C as output.

Soln:

String ending with 10 ⟶ A
string ending with 11 ⟶ B
others ⟶ C

$\Sigma = \{0,1\}$

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




$Q = \{W, X, Y, Z\}$

$\Sigma = \{0,1\}$

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

$\lambda = (X,0) \to C, (Y,0) \to A, (Z,0) \to A, (W,0) \to C$

$(X,1) \to C, (Y,1) \to B; (Z,1) \to B, (W,1) \to 1$

$\delta$ is defined by

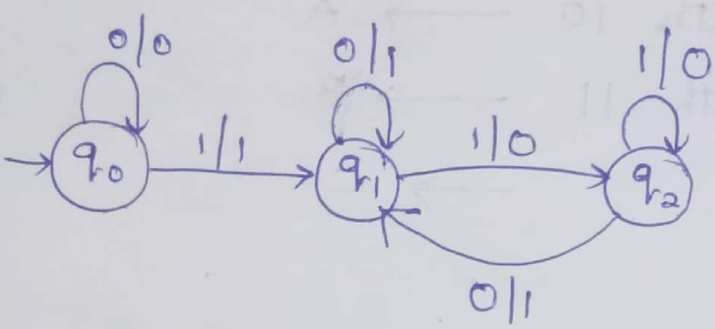
| Present state | I/p = 0 | | I/p = 1 | |
|---|---|---|---|---|
| | NS | O/P | NS | O/P |
| → X | X | C | Y | C |
| Y | W | A | Z | B |
| Z | W | A | Z | B |
| W | X | C | Y | C |

7) Construct a Mealy Machine for producing 2's complement for a given binary string.

Soln

$\Sigma = \{0,1\}$

$\Delta = \{0,1\}$




$Q = \{q_0, q_1, q_2\}$

$\Sigma = \{0,1\}$

$\Delta = \{0,1\}$

$q_0 = \{q_0\}$

$\lambda = (q_0,0) \rightarrow 0$ , $(q_1,0) \rightarrow 1$ , $(q_2,0) \rightarrow 1$

$(q_0,1) \rightarrow 1$ , $(q_1,1) \rightarrow 0$ , $(q_2,1) \rightarrow 0$

$\delta$ is defined by.

| Present state | I/p = 0 | | I/p = 1 | |
|---|---|---|---|---|
| | Ns | O/p | Ns | OP |
| $q_0$ | $q_0$ | 0 | $q_1$ | 1 |
| $q_1$ | $q_1$ | | $q_2$ | 0 |
| $q_2$ | $q_1$ | 1 | $q_2$ | 1 |

# Equivalence of Moore and Mealy Machine

## Moore Machine to Mealy Machine

### Algorithm

Input :- Moore Machine
output :- Mealy Machine

Step 1 :- Take a blank Mealy machine transition format.

step 2 :- Copy all the Moore Machine transition states into this table format.

step 3 :- Check the present states and their corresponding outputs in the Moore machine state table; if for a state $q_i$ output is m, copy it into the output columns of the Mealy machine state table wherever $q_i$ appears in the next state.

### Examples

1) Convert the following Moore Machine to Mealy Machine.

Moore Machine

| Present state | Next state | | output |
|---|---|---|---|
| | i/p = 0 | i/p = 1 | |
| A | A | B | 0 |
| B | B | C | 1 |
| C | B | C | 0 |

soln Step 1 and step 2

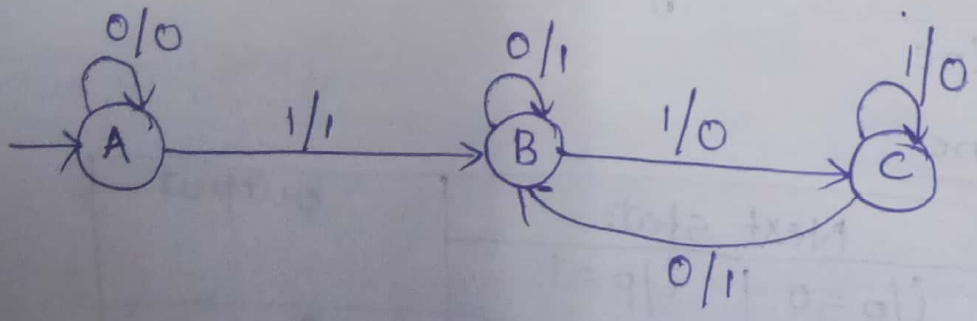
Mealy Machine Table

| Present state | I/P = 0 | | I/P = 1 | |
|---|---|---|---|---|
| | NS | O/P | NS | O/P |
| → A | A | | B | |
| B | B | | C | |
| C | B | | C | |

Step 3

| Present state | I/P = 0 | | I/P = 1 | |
|---|---|---|---|---|
| | NS | O/P | NS | O/P |
| → A | A | 0 | B | 1 |
| B | B | 1 | C | 0 |
| C | B | 1 | C | 0 |

Final Mealy Machine transition diagram is



2) Moore Machine

| Present state | Next state | | output |
|---|---|---|---|
| | I/p=0 | I/p=1 | |
| → a | d | b | 1 |
| b | a | d | 0 |
| c | c | c | 0 |
| d | b | a | 1 |

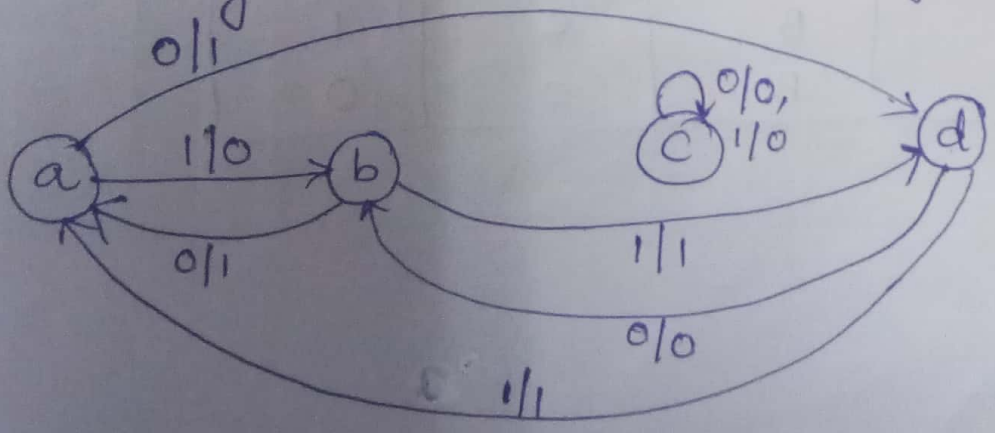
Soln. step 1 and step 2

| Present state | I/p = 0 | | I/p = 1 | |
|---|---|---|---|---|
| | NS | O/P | NS | O/P |
| → a | d | | b | |
| b | a | | d | |
| c | c | | c | |
| d | b | | a | |

Step 3

| Present state | I/p = 0 | | I/p = 1 | |
|---|---|---|---|---|
| | NS | O/P | NS | O/P |
| a | d | 1 | b | 0 |
| b | a | 1 | d | 1 |
| c | c | 0 | c | 0 |
| d | b | 0 | a | 1 |

Final Mealy Machine Transition diagram is



# Mealy Machine to Moore Machine

## Algorithm

Input :- Mealy Machine
output :- Moore Machine

step 1). calculate the number of different outputs for each state ($Q_i$) that are available in the state table of the Mealy Machine.

step 2:- If all the outputs of $Q_i$ are same, copy state $Q_i$. If it has n distinct outputs, break $Q_i$ into n states as $Q_{in}$ where n = 0, 1, 2, ....

step 3:- If the output of the initial state is 1, insert a new initial state at the beginning which gives 0 output.

## Examples

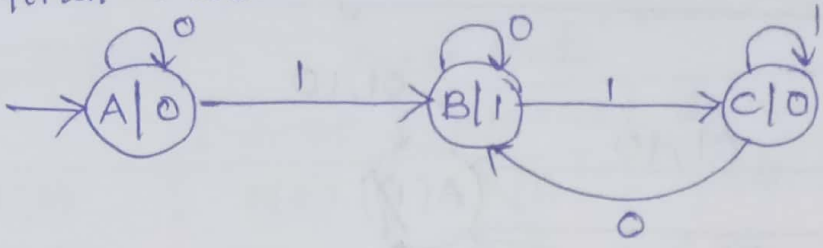
1)

| Present state | I/p = 0 | | I/p = 1 | |
|---|---|---|---|---|
| | NS | O/p | NS | O/p |
| → A | A | 0 | B | 1 |
| B | B | 1 | C | 0 |
| C | B | 1 | C | 0 |

Soln step 1 and step 2

| Present state | NS | | O/p |
|---|---|---|---|
| | I/p = 0 | I/p = 1 | |
| → A | A | B | 0 |
| B | B | C | 1 |
| C | B | C | 0 |

final Moore Machine Transition diagram

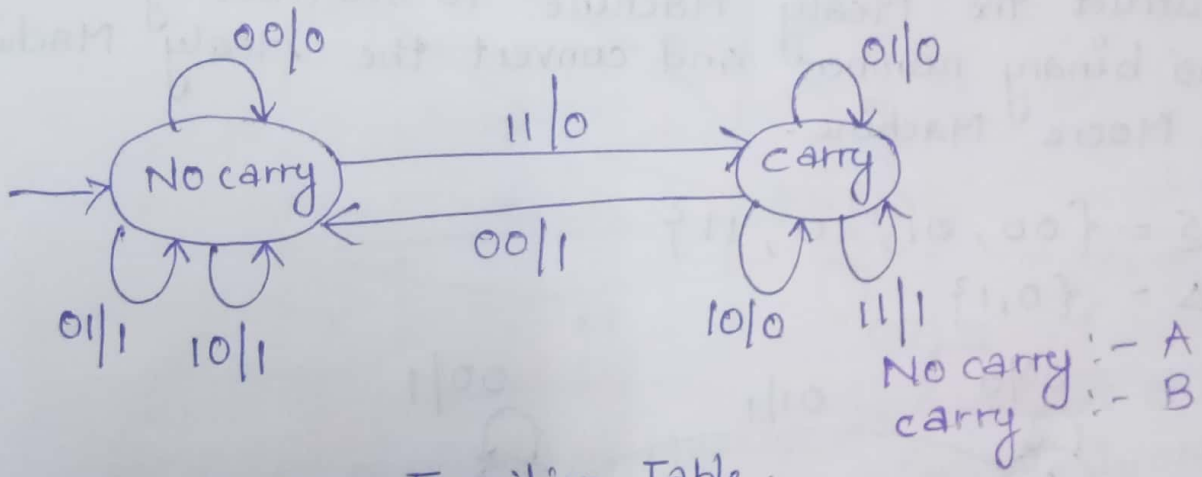



Ex 2) Determine the Mealy Machine for adding two binary numbers (Binary Adder) and find the equivalent Moore Machine.

Sol$^n$

$\Sigma = \{00, 01, 10, 11\}$

$\Delta = \{0, 1\}$



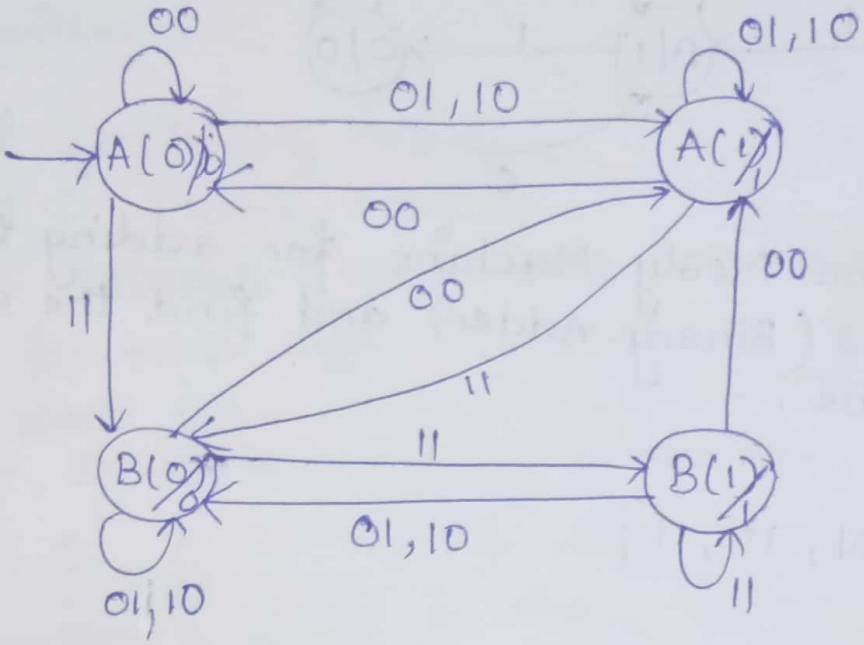

Mealy Machine Transition Table

| Present state | I/p = 00 | | I/p = 01 | | I/p = 10 | | I/p = 11 | |
|---|---|---|---|---|---|---|---|---|
| | NS | O/P | NS | O/P | NS | O/P | NS | O/P |
| → A | A | 0 | A | 1 | A | 1 | B | 0 |
| B | A | 1 | B | 0 | B | 0 | B | 1 |

Moore Machine Transition Table

| Present state | Next state | | | | O/P |
|---|---|---|---|---|---|
| | I/p=00 | I/p=01 | I/p=10 | I/p=11 | |
| →A(0) | A(0) | A(1) | A(1) | B(0) | 0 |
| A(1) | A(0) | A(1) | A(1) | B(0) | 1 |
| B(0) | A(1) | B(0) | B(0) | B(1) | 0 |
| B(1) | A(1) | B(0) | B(0) | B(1) | 1 |

## Moore Machine Transition Diagram

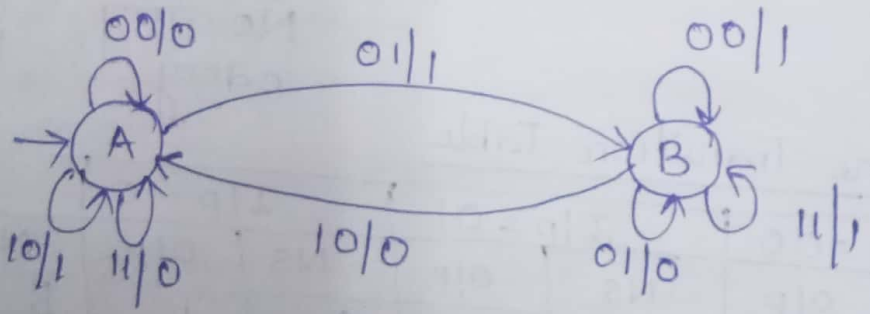



3) Construct the Mealy Machine to subtract given two binary numbers and convert the Mealy Machine to Moore Machine.

301

$\Sigma = \{00, 01, 10, 11\}$

$\Delta = \{0, 1\}$



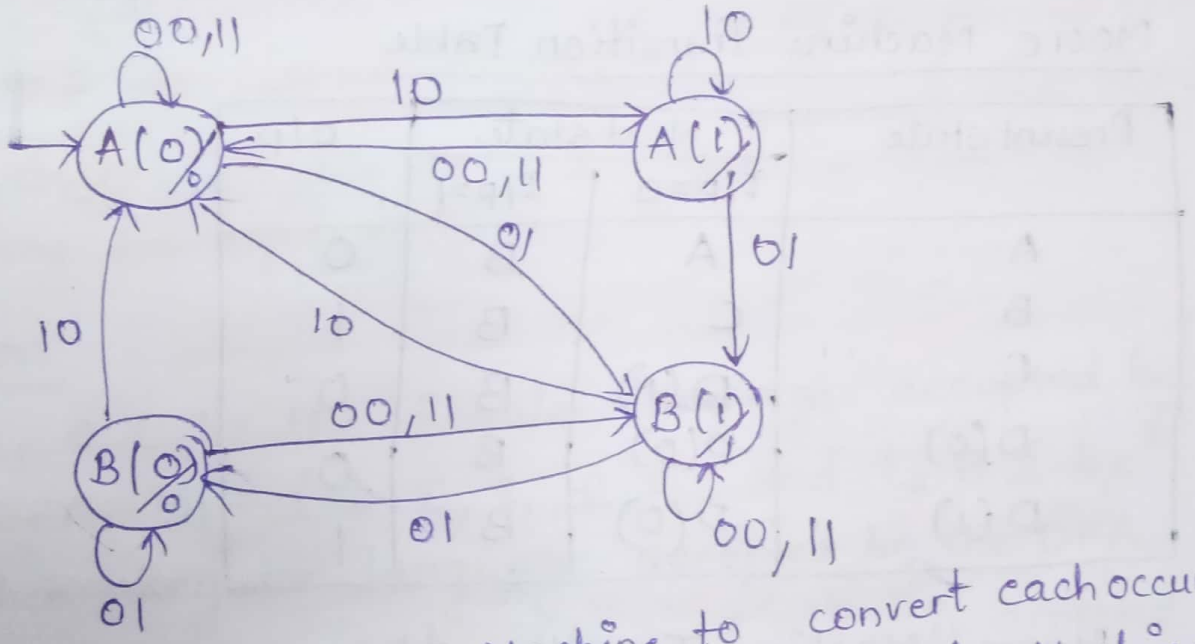

## Mealy Machine Transition Table.

| Present state | I/p = 00 | | I/p = 01 | | I/p = 10 | | I/p = 11 | |
|---|---|---|---|---|---|---|---|---|
| | NS | O/P | NS | O/P | NS | O/P | NS | O/P |
| → A | A | 0 | B | 1 | A | 1 | A | 0 |
| B | B | 1 | B | 0 | A | 0 | B | 1 |

## Moore Machine Transition Table.

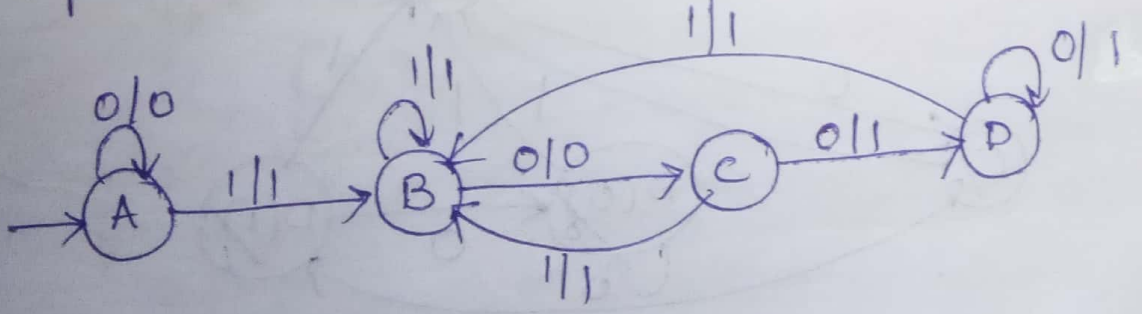
| Present state | Next state | | | | Output |
|---|---|---|---|---|---|
| | I/p = 00 | I/p = 01 | I/p = 10 | I/p = 11 | |
| A(0) | A(0) | B(1) | A(1) | A(0) | 0 |
| A(1) | A(0) | B(1) | A(1) | A(0) | 1 |
| B(0) | B(1) | B(0) | A(0) | B(1) | 0 |
| B(1) | B(1) | B(0) | A(0) | B(1) | 1 |

Moore Machine Transition diagram




4) Construct the Mealy Machine to convert each occurrence of substring 100 by 101. convert the Mealy Machine to equivalent Moore Machine.

Sol$^{n}$



## Mealy Machine Transition Table

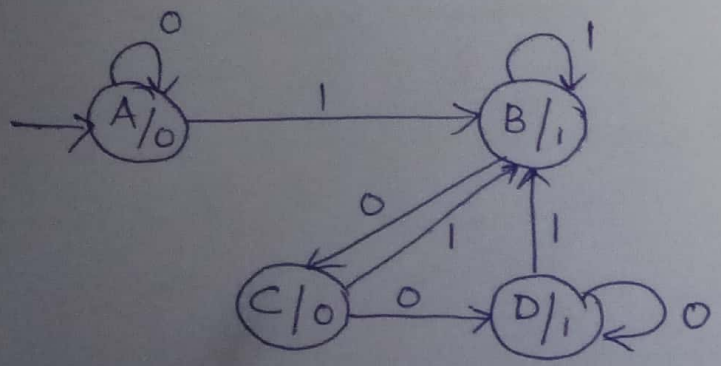
| Present state | I/p = 0 | | I/p = 1 | |
|---|---|---|---|---|
| | NS | O/P | NS | O/P |
| A | A | 0 | B | 1 |
| B | C | 0 | B | 1 |
| C | D | 1 | B | 1 |
| D | D | 1 | B | 1 |

A → 0
C → 0
B → 1
D → 1

## Moore Machine Transition Table

| Present state | Next state | | O/P |
|---|---|---|---|
| | I/p = 0 | I/p = 1 | |
| A | A | B | 0 |
| B | C | B | 1 |
| C | D | B | 0 |
| D | D | B | 1 |

## Moore Machine Transition diagram



# Closure Properties of Regular Languages

**Theorem:-** The family of Regular is closed under the following operations.

1) union
2) Intersection
3) complementation
4) Concatenation
5) closure
6) Reversal

**Proof:-** The six closure properties will be proved either through FSA or by Regular grammars and shown they are equivalent.

## 1) union

Let $L_1 \in \Sigma$ be the Regular language accepted by the DFA $M_1 = (Q_1, \Sigma, q_1, \delta_1, F_1)$ and $L_2 \in \Sigma$ be another Regular language accepted by the DFA $M_2 = (Q_2, \Sigma, q_2, \delta_2, F_2)$ then $L_1 \cup L_2$ is accepted by DFA $M_3 = (Q_3, \Sigma, q_3, \delta_3, F_3)$

where

$Q_3 = \{Q_1 \times Q_2\}$

$q_3 = [q_1, q_2]$

$\delta_3$ is defined by,

for any state $P \in Q_1$, $q \in Q_2$ and every symbol $a \in \Sigma$,

$\delta_3([Pq], a) = \delta_1(P, a) \cup \delta_2(q, a)$ and

$F_3 = \{[Pq] \mid P \in F_1 \text{ or } q \in F_2\}$

So $L_1 \cup L_2$ is also regular

2) Intersection

Let $L_1 \in \Sigma$ be the Regular Language accepted by DFA $M_1 = (Q_1, \Sigma, q_1, \delta_1, F_1)$ and $L_2 \in \Sigma$ be another language accepted by DFA $M_2 = (Q_2, \Sigma, q_2, \delta_2, F_2)$ then $L_1 \cap L_2$ is accepted by DFA $M_3 = (Q_3, \Sigma, q_3, \delta_3, F_3)$ where

$Q_3 = \{Q_1 \times Q_2\}$

$q_3 = [q_1 q_2]$

$\delta_3$ is defined by,

for any state $P \in Q_1$, $q \in Q_2$ and $a \in \Sigma$

$$\delta([Pq], a) = \delta_1(P, a) \cup \delta_2(q, a)$$

$$F_3 = \{[Pq] \mid P \in F_1 \text{ and } q \in F_2\}$$

3) complementation

Let $L_1 \in \Sigma$ be the Regular language accepted by DFA $M_1 = (Q_1, \Sigma, q_1, \delta_1, F_1)$ and the complement of $L_1'$ is accepted by the DFA $M = (Q, \Sigma, q_1, \delta, (Q - F))$

Hence $L_1'$ is also Regular.

4) Concatenation

This property can be proved by the Regular grammar.. Let $L_1 \in \Sigma$ be the Regular Language generated by the right linear grammar $G = (V_1, \Sigma, S_1, P_1\}$ and

$L_2 \in \Sigma$ be another Regular language generated by the right linear grammar $G_2 = (V_2, \Sigma, S_2, P_2)$ then the right linear grammar $G_3 = (V_3, \Sigma, S_3, P_3)$ is constructed as below satisfying the requirement that $L(G_3) = L(G_1) . L(G_2)$ where

$V_3 = \{V_1 \cup V_2\}$

$S_3 = S_1$

$P_3 = \{P_1 \cup P_2\} \cup S_3 \longrightarrow S_1 S_2$

clearly, $L(G_3) = L(G_1) \cdot L(G_2)$ because any deviation starting from $S_1$ derives a string $w \in L_1$ and for $G_3$, $S_1 \rightarrow w S_2$. hence if $S_2 \longrightarrow {}^* w'$ by $G_2$ then $S_1 \longrightarrow {}^* ww'$ by $G_3$. so $L_1 L_2$ is also Regular.

5) Closure

Let $L_1 \in \Sigma$ be the Regular language accepted by the Non-Deterministic FA $M = (Q, \Sigma, q_0, \delta, F)$ then clearly the closure of $L_1$ (i.e $L_1^*$) is accepted by the NFA $M' = (Q, \Sigma, q_0, \delta', F)$ where $\delta'$ is defined by for every $q_1 \in F$ in M, $\delta'$ includes $\delta(q_1, \epsilon) = q_0$. clearly $M'$ accepts $L_1^*$. so $L_1^*$ is also regular.

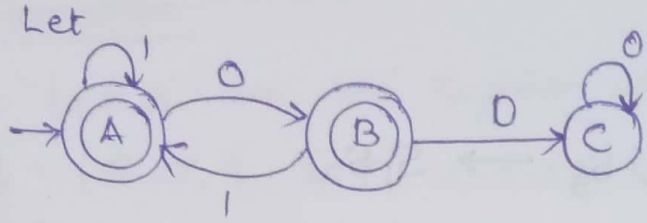
6) Reversal

Let $L_1 \in \Sigma$ be the regular language accepted by NFA $M = (Q, \Sigma, q_0, \delta, q_f)$ then the reversal automaton $M' = (Q, \Sigma, q_f, \delta', \{q_0\})$ where $\delta'$ is defined as,

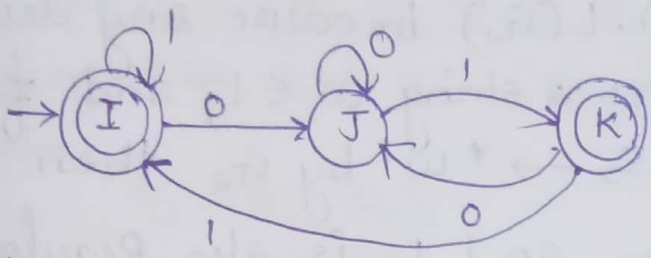
$\delta'(q, a) = p$ iff $\delta(p, a) = q$, for any $p, q \in Q$ & $a \in \Sigma$

one can see that if $w \in L(M)$ then $w^R \in L(M)$ as in the modified automaton $M'$, each transition takes a backward movement on $w$.

1) Let




be $M_1$ recognizing $L_1$




be $M_2$ recognizing $L_2$

find $L_1 \cup L_2$, $L_1 \cap L_2$ and $L_1 - L_2$

Soln

Thus the DFA $M = (Q, \Sigma, \delta, q_0, F)$

$Q = Q_1 \times Q_2$

$= \{[AI], [AJ], [AK], [BI], [BJ], [BK], [CI], [CJ], [CK]\}$

$\Sigma = \{0, 1\}$

$q_0 = \{[AI]\}$

$\delta$ is defined by

$\delta([AI], 0) = \delta_1(A, 0) \cup \delta_2(I, 0) = [BJ]$

$\delta([AI], 1) = \delta_1(A, 1) \cup \delta_2(I, 1) = [AI]$

$\delta([AJ], 0) = \delta_1(A, 0) \cup \delta_2(J, 0) = [BJ]$

$\delta([AJ], 1) = \delta_1(A, 1) \cup \delta_2(J, 1) = [AK]$

$\delta([AK], 0) = \delta_1(A, 0) \cup \delta_2(K, 0) = [BJ]$

$\delta([AK], 1) = \delta_1(A, 1) \cup \delta_2(K, 1) = [AI]$

$\delta([BI], 0) = \delta_1(B, 0) \cup \delta_2(I, 0) = [CJ]$

$\delta([BI], 1) = \delta_1(B, 1) \cup \delta_2(I, 1) = [AI]$

$\delta([BJ], 0) = \delta_1(B, 0) \cup \delta_2(J, 0) = [CJ]$

$\delta([BJ], 1) = \delta_1(B, 1) \cup \delta_2(J, 1) = [AK]$

$\delta([BK], 0) = \delta_1(B, 0) \cup \delta_2(K, 0) = [CJ]$

$\delta([BK], 1) = \delta_1(B, 1) \cup \delta_2(K, 1) = [AI]$

$\delta([CK], 0) = \delta_1(C, 0) \cup \delta_2(K, 0) = [CJ]$

$\delta([CK], 1) = \delta_1(C, 1) \cup \delta_2(K, 1) = [CI]$

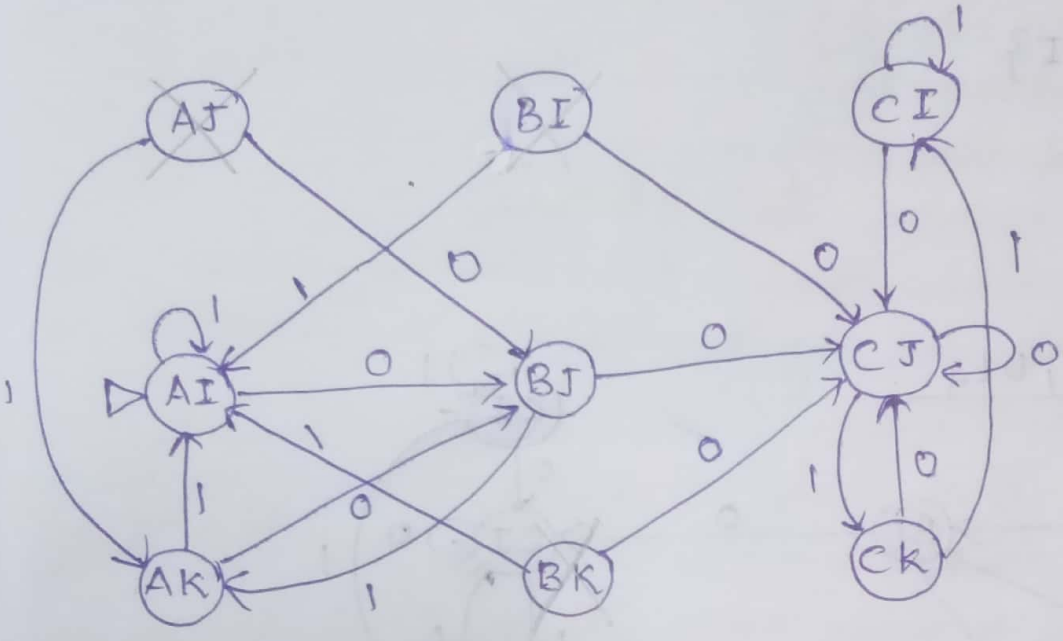
$\delta([CI],0) = \delta_1(C,0) \cup \delta_2(I,0) = [CJ]$

$\delta([CI],1) = \delta_1(C,1) \cup \delta_2(I,1) = [CI]$

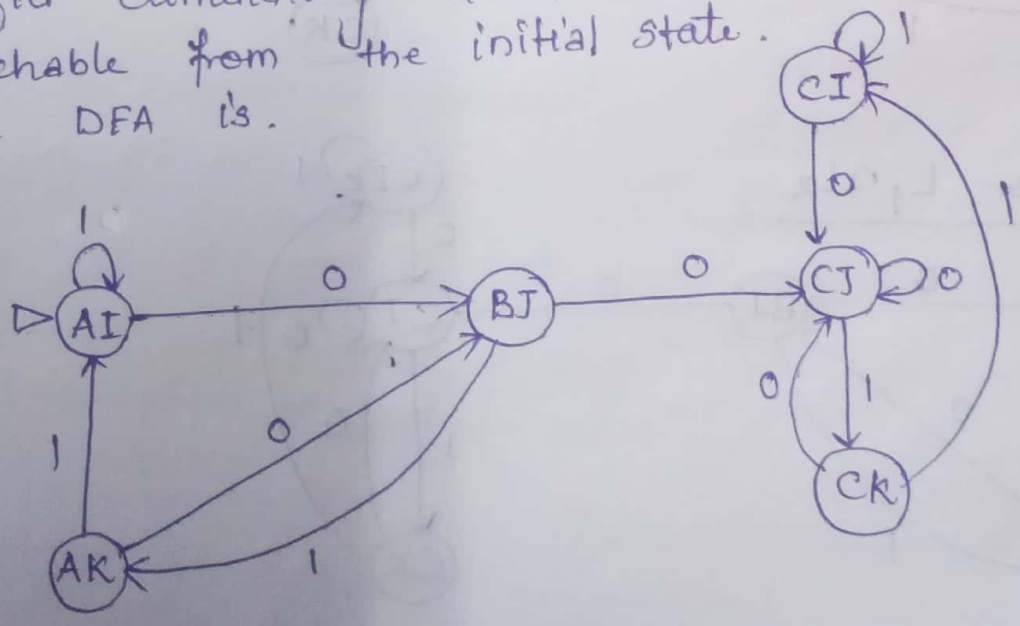
$\delta([CJ],0) = \delta_1(C,0) \cup \delta_2(J,0) = [CJ]$

$\delta([CJ],1) = \delta_1(C,1) \cup \delta_2(J,1) = [CK]$

Thus the DFA is




After Eliminating dead states which are not reachable from the initial state. The DFA is.



for $L_1 \cup L_2$

$F = \{ AI, BJ, AK, CK, CI \}$

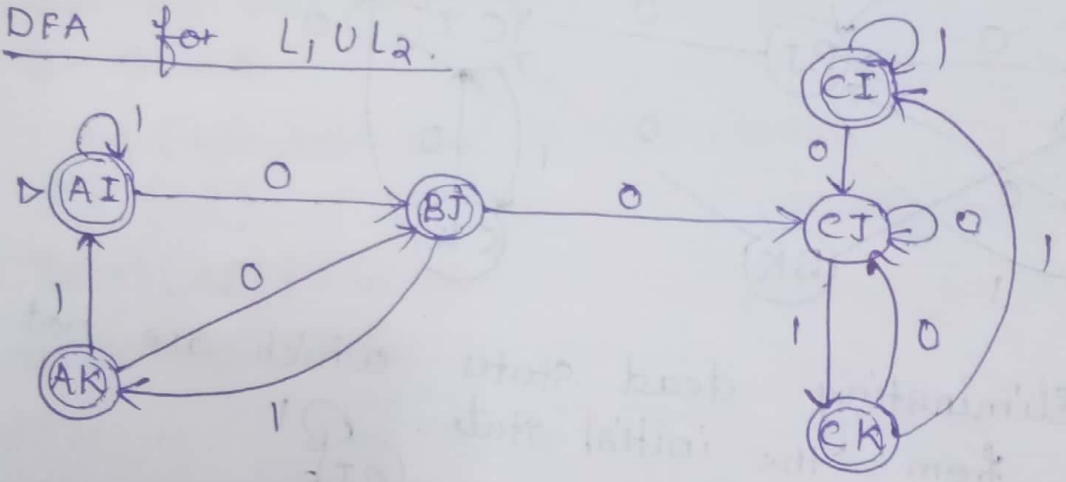
for $L_1 \cap L_2$

$F = \{ AK, AI \}$

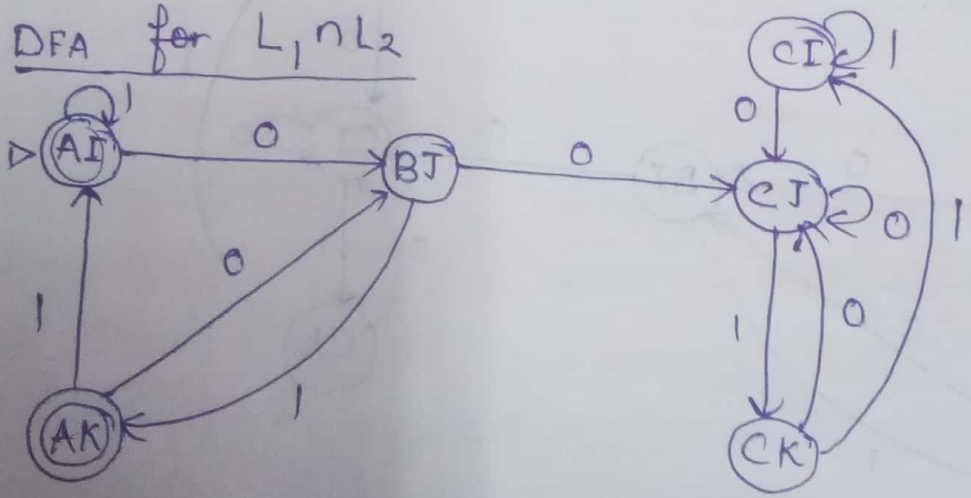
for $L_1 - L_2$

$F = \{ BJ \}$

DFA for $L_1 \cup L_2$.




DFA for $L_1 \cap L_2$



DFA for $L_1 - L_2$