Theory of Computation

Unit 1

DEFINITION OF AN AUTOMATON

An automaton is defined as a system where energy, materials and information are transformed, transmitted and used for performing some functions without direct participation of man.

- Examples are- automatic machine tools, automatic packing machines, and automatic photo printing machines.
- In computer science the term 'automaton' means 'discrete automaton'

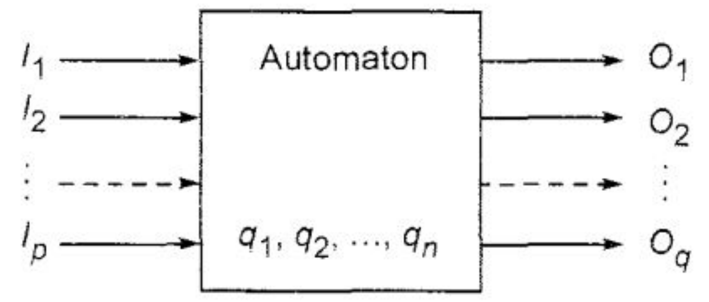


Fig. 3.1 Model of a discrete automaton.

Characteristics of Automaton

- **Input-**

At each of the discrete instants of time t1, t2, ... tm the input values I1 I2..... Ip ,each of which can take a finite number of fixed values from the input alphabet ∑, are applied to the input side of the model shown in Fig. 3.l.

- **Output-**

O1, O2,....., Oq are the outputs of the model, each of which can take a finite number of fixed values from an output O.

Characteristics of Automaton

- **States-**

At any instant of time the automaton can be in one of the states q1, q2,...., qn

- **State relation-**

The next state of an automaton at any instant of time is determined by the present state and the present input.

- **Output relation-**

The output is related to either state only or to both the input and the state. It should be noted that at any instant of time the automaton is in some state. On 'reading' an input symbol, the automaton moves to a next state which is given by the state relation.

- An automaton in which the output depends only on the input is called an **automaton without a memory**.
- An automaton in which the output depends on the states as well is called **automaton with a finite memory**.
- An automaton in which the output depends only on the states of the machine is called a **Moore machine**.
- An automaton in. which the output depends on the state as well as on the input at any instant of time is called a **Mealy machine**.

FINITE AUTOMATON

Analytically, a finite automaton can be represented by a 5-tuple

(Q, $\sum$, δ, q_0, F) where

Q : Finite set of states.

Σ : set of Input Symbols or called the input alphabet.

δ : Transition Function.

q_0 : Initial state where $q_0 \epsilon Q$

F : set of Final States where $F \subseteq Q$.

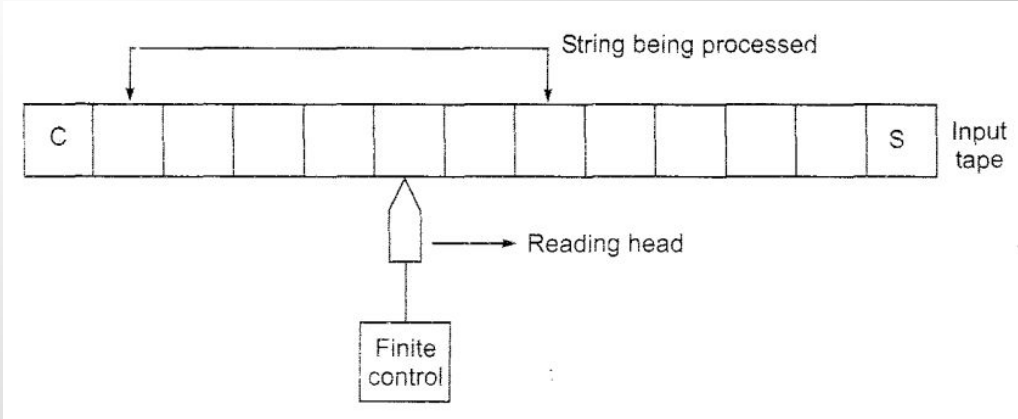


Fig. 3.4 Block diagram of a finite automaton.

TRANSITION SYSTEMS

A transition graph or a transition system is a finite directed labelled graph in which each vertex (or node) represents a state and the directed edges indicate the transition of a state and the edges are labelled with input/output.

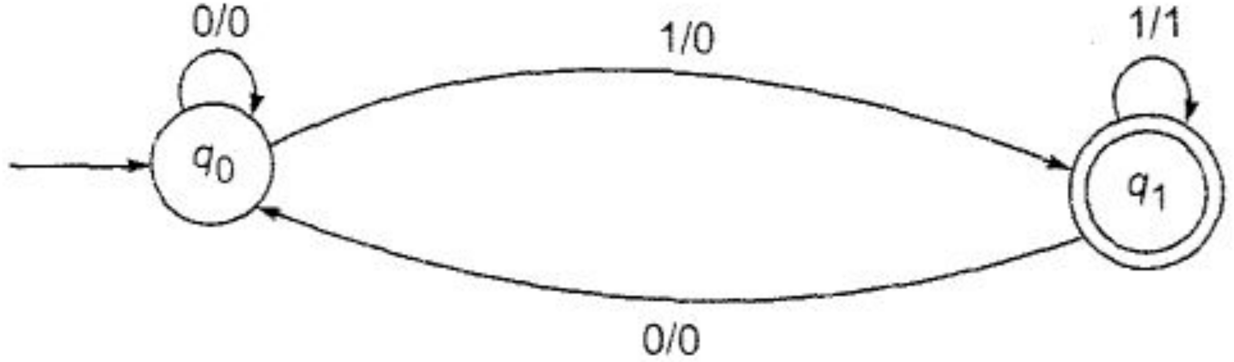


Fig. 3.5 A transition system.

Definition 3.1:

A transition system is a 5-tuple $(\boldsymbol{Q}, \Sigma, \boldsymbol{\delta}, q_0, F)$ where

$\boldsymbol{Q}$: Finite set of states.

Σ : set of Input Symbols or called the input alphabet.

q_0 : Initial state where $q0 \in \boldsymbol{Q}$

F : set of Final States where F $\subseteq \boldsymbol{Q}$

δ : Transition Function and a finite subset of $\boldsymbol{Q} \times \Sigma^* \times \boldsymbol{Q}$

If (q_1, w, q_2) is in δ, it means that the graph starts at the vertex q_1, goes along a set of edges, and reaches the vertex q_2.

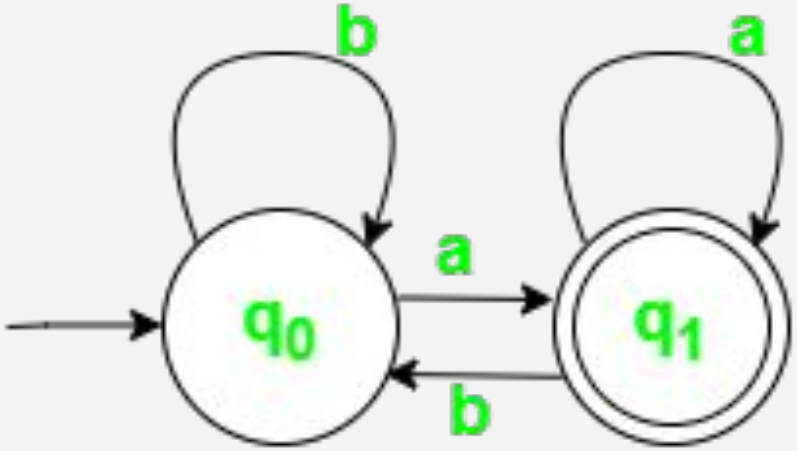
The concatenation of the label of all the edges thus encountered is w.

Definition 3.2:

A transition system accepts a string w in ∑* if

(i) there exists a path which originates from some initial state, goes along the arrows, and terminates at some final state; and

(ii) the path value obtained by concatenation of all edge-labels of the path is equal to *w*.



∑={a,b}

L= set of strings ending with a

Strings accepted are,

a, aa, aaa, aaaa, aaaaa, ba, bba, bbbaa, aba, abba, aaba, abaa

Strings not accepted are,

ab, bb, aab, abbb

Properties of transition functions

- Property 1:

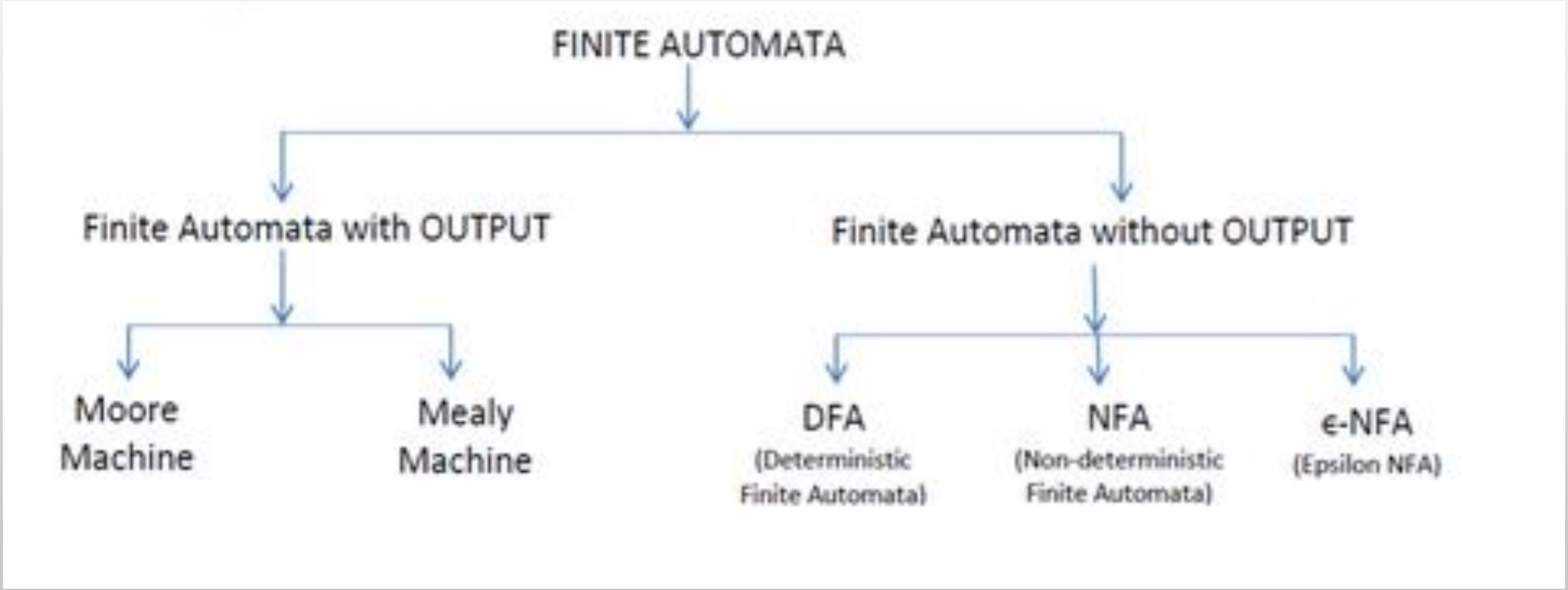
$$\delta(q,\Lambda) = q.$$

It means the state of a system can be changed by an input symbol.

- Property 2:
  For all strings w and input symbol a,
  $\delta(q, aw) = \delta(\delta(q,a),w)$
  $\delta(q, wa) = \delta(\delta(q,w), a)$

It means the state after the automaton consumes or reads the first symbol of a string aw and the state after the automaton consumes a prefix of the string wa.

FINITE AUTOMATA CLASSIFICATION


FINITE AUTOMATA
Finite Automata with OUTPUT
Finite Automata without OUTPUT
Moore Machine
Mealy Machine
DFA (Deterministic Finite Automata)
NFA (Non-deterministic Finite Automata)
ϵ-NFA (Epsilon NFA)

DETERMINISTIC FINITE STATE MACHINES (DFA)

A deterministic finite automaton (DFA) is a 5-tuple ($\boldsymbol{Q}$, $\sum$, $\boldsymbol{\delta}$, $\boldsymbol{q}_0$, F) where

$\boldsymbol{Q}$: Finite set of states.

Σ : set of Input Symbols or called the input alphabet.

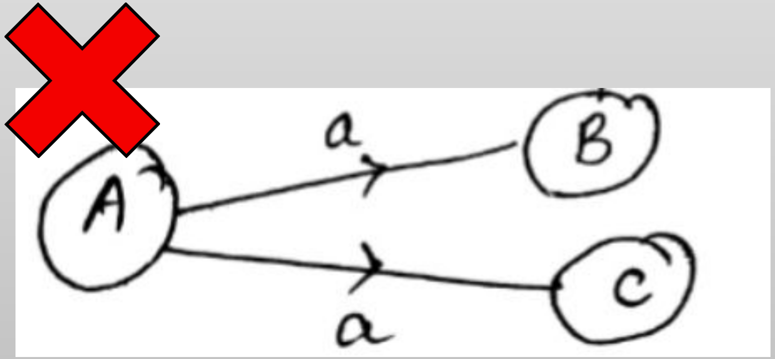
$\boldsymbol{q}$0 : Initial state where $\boldsymbol{q}0 \ \boldsymbol{\epsilon} \ \boldsymbol{Q}$

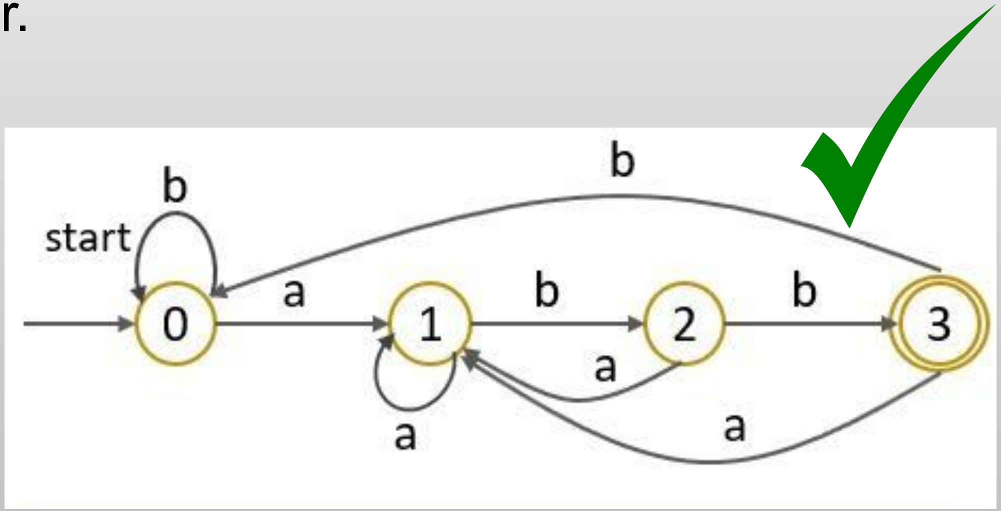
F : set of Final States where F $\subseteq$ $\boldsymbol{Q}$

δ : Transition Function, defined as δ : Q X Σ --> Q

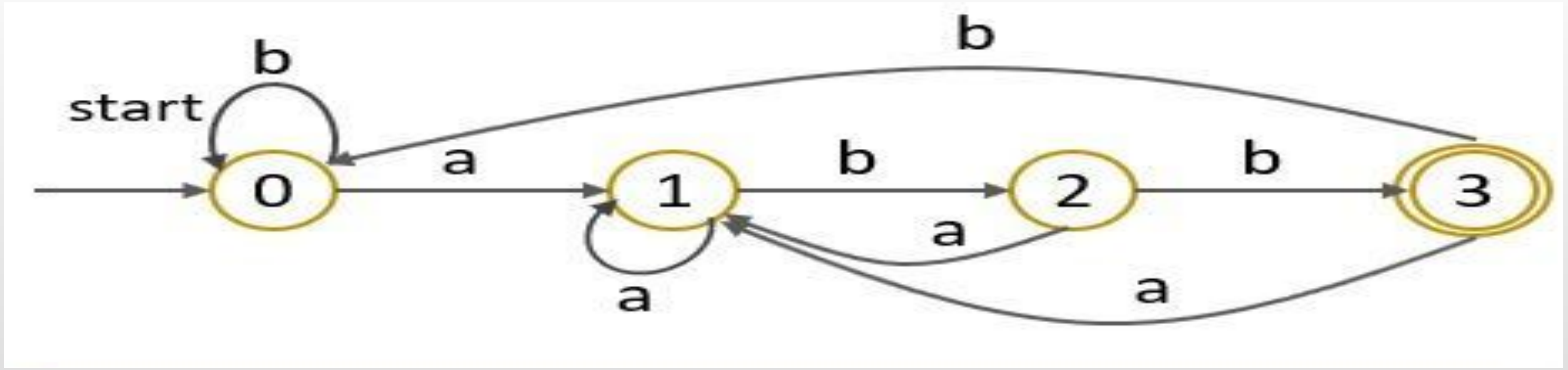
DETERMINISTIC FINITE STATE MACHINES (DFA)

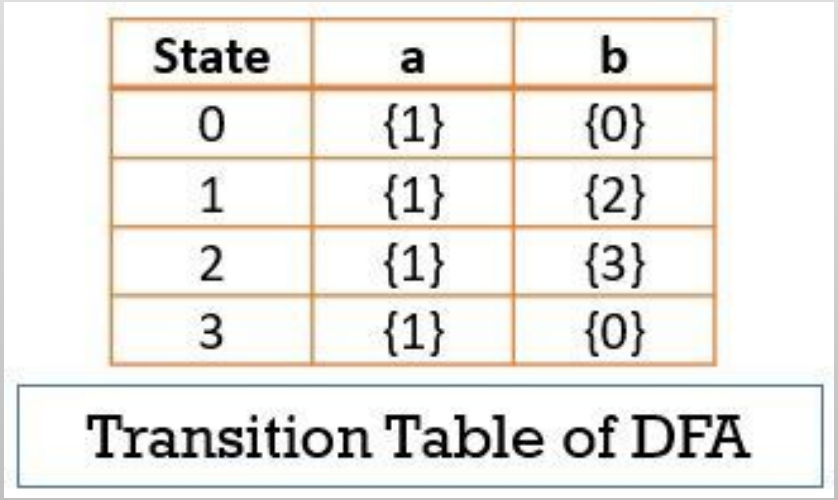
- For a particular input character, the machine goes to one state only.
- A transition function is defined on every state for every input symbol.
- Also in DFA null (or ε) move is not allowed, i.e., DFA cannot change state without any input character.





DETERMINISTIC FINITE STATE MACHINES (DFA)





State	a	b
0	{1}	{0}
1	{1}	{2}
2	{1}	{3}
3	{1}	{0}

Transition Table of DFA

DETERMINISTIC FINITE STATE MACHINES (DFA)

1. Construct a DFA which accept a language of all strings ending with ‘a’.

Given: Σ = {a,b}, q = {q0}, F={q1}, Q = {q0, q1}

L = {a, aa, aaa, aaaa, aaaaa, ba, bba, bbbaa, aba, abba, aaba, abaa}...(hint)

2. Construct a DFA which accept a language of all strings starting with ‘b’.

Given: Σ = {a,b}, q = {q0}, F={q1}, Q = {q0, q1,q2}

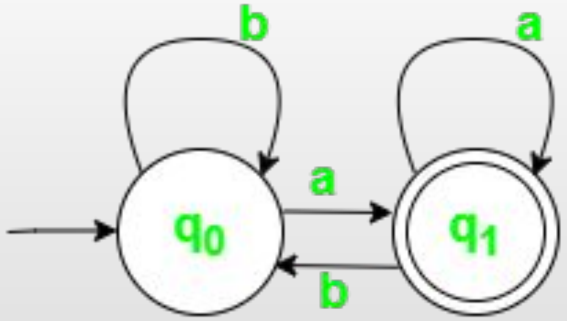
L = {b, bb, bbb, ba, bba, bbbaa, bba, bbbab, bbba, baab, babab}...(hint)

3. Draw a DFA for the language accepting strings containing 0

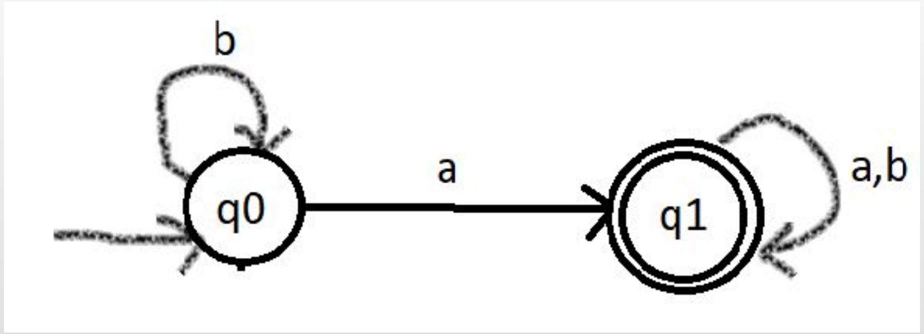
4. Draw a DFA for the language accepting strings starting and ending with same character over input alphabets ∑ = {0, 1}

DETERMINISTIC FINITE STATE MACHINES (DFA) soln

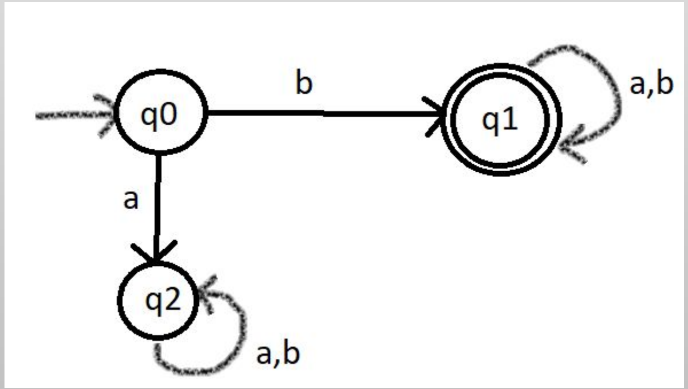
1.



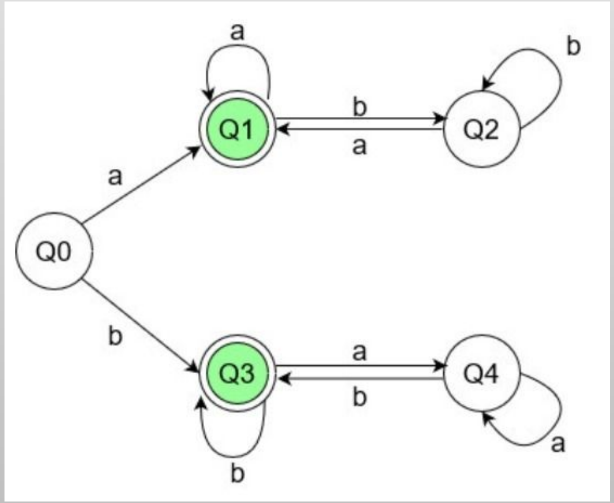
3.



2.



4.



NON-DETERMINISTIC FINITE STATE MACHINES (NFA)

A nondeterministic finite automaton (NDFA) is a 5-tuple ($\boldsymbol{Q}$, ∑, δ, q_0, F) where

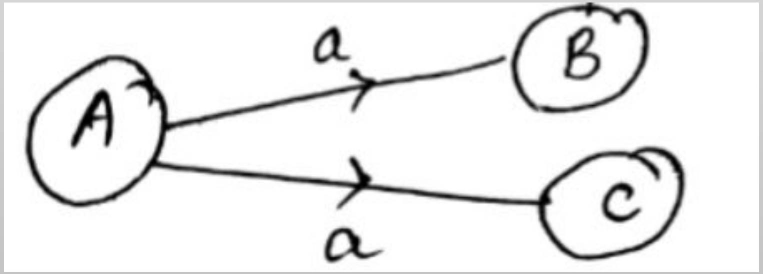
$\boldsymbol{Q}$: Finite set of states.

Σ : set of Input Symbols or called the input alphabet.

$q0$: Initial state where $q0 \in \boldsymbol{Q}$

F : set of Final States where F $\subseteq \boldsymbol{Q}$

δ : Transition Function and a mapping from $\boldsymbol{Q}$ x Σ into $2^{\boldsymbol{Q}}$ which is the power set of $\boldsymbol{Q}$, the set of all subsets of $\boldsymbol{Q}$



NON-DETERMINISTIC FINITE STATE MACHINES (NFA)

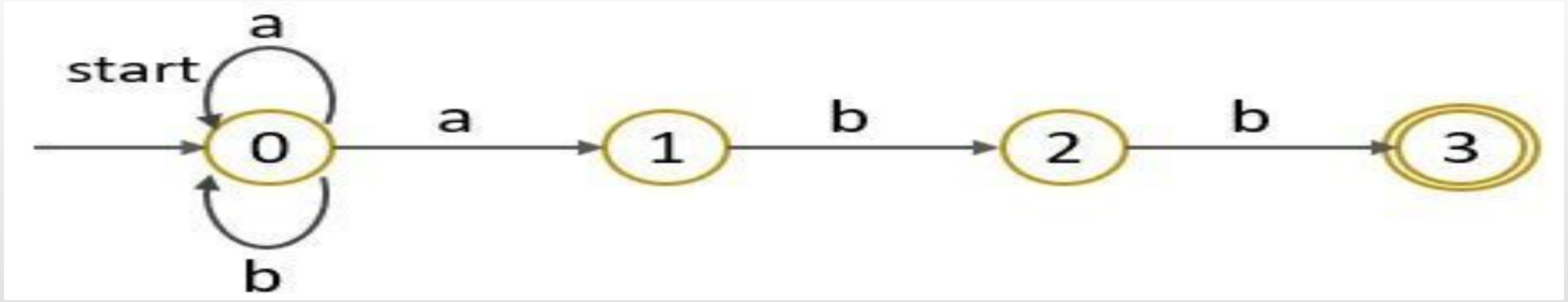
- NFA is more of a theoretical concept.
- All real machines are DFA

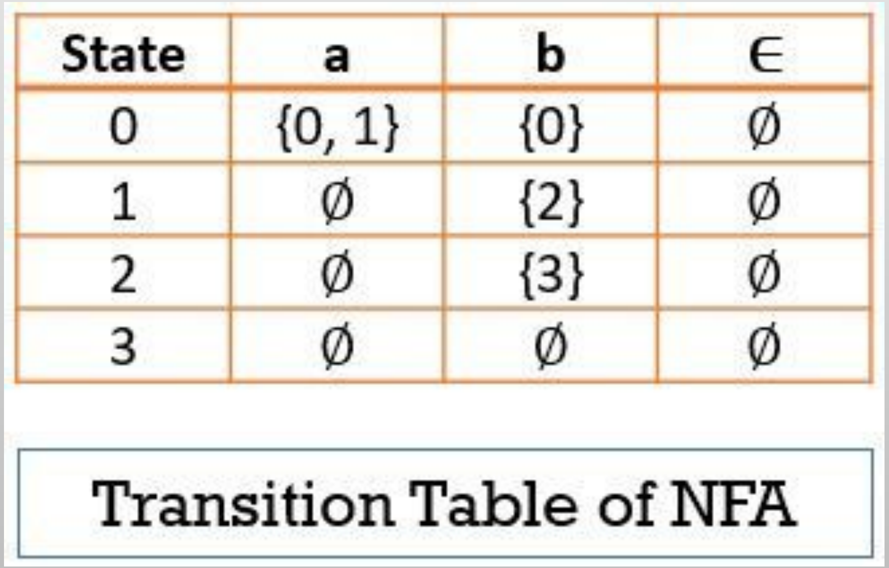
NFA is similar to DFA except following additional features:

- Null (or ε) move is allowed i.e., it can move forward without reading symbols.
- Ability to transmit to any number of states for a particular input.

If the number of states in the NFA is N then, its DFA can have maximum 2^N number of states.

NON-DETERMINISTIC FINITE STATE MACHINES (NFA)





State	a	b	∈
0	{0, 1}	{0}	∅
1	∅	{2}	∅
2	∅	{3}	∅
3	∅	∅	∅

Transition Table of NFA

NON DETERMINISTIC FINITE STATE MACHINES (NFA)

1. Construct a NFA which accept a language of all strings ending with ‘0’.

Given: $\Sigma = \{0,1\}$, $q = \{q0\}$, $F=\{q1\}$, $Q = \{q0, q1\}$

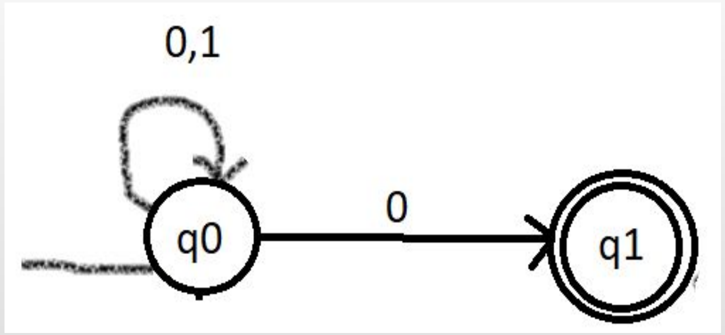
2. Construct a DFA which accept a language of all strings containing ‘0’.

Given: $\Sigma = \{0,1\}$, $q = \{q0\}$, $F=\{q1\}$, $Q = \{q0, q1\}$

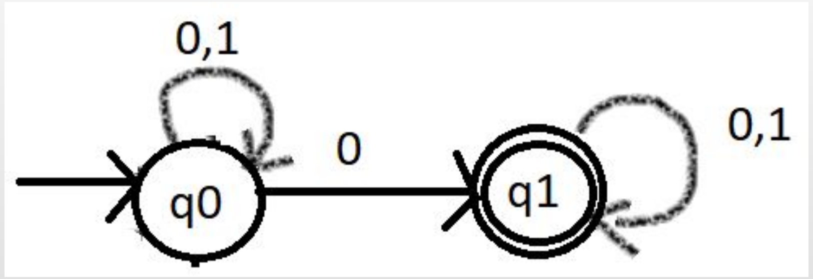
3. Draw a DFA for the language accepting strings containing ‘01’

NON DETERMINISTIC FINITE STATE MACHINES (NFA) soln

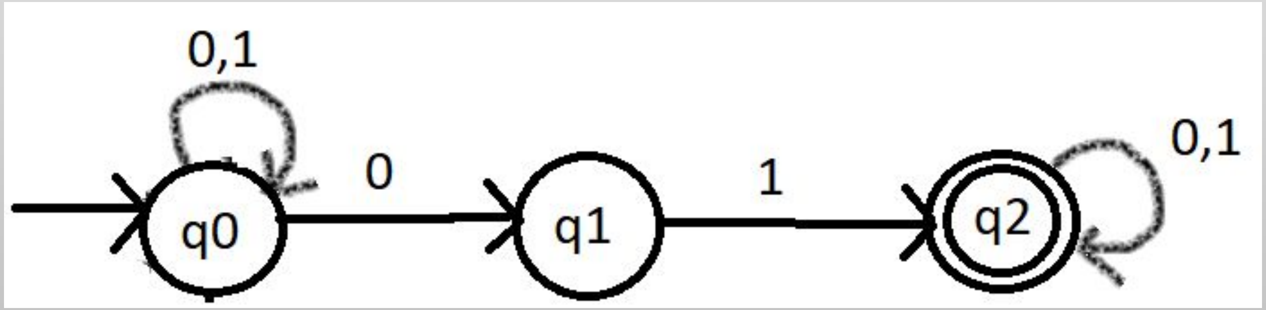
1.



3.



2.



ACCEPTABILITY OF A STRING BY A FINITE AUTOMATON

A string x is accepted by a finite automaton

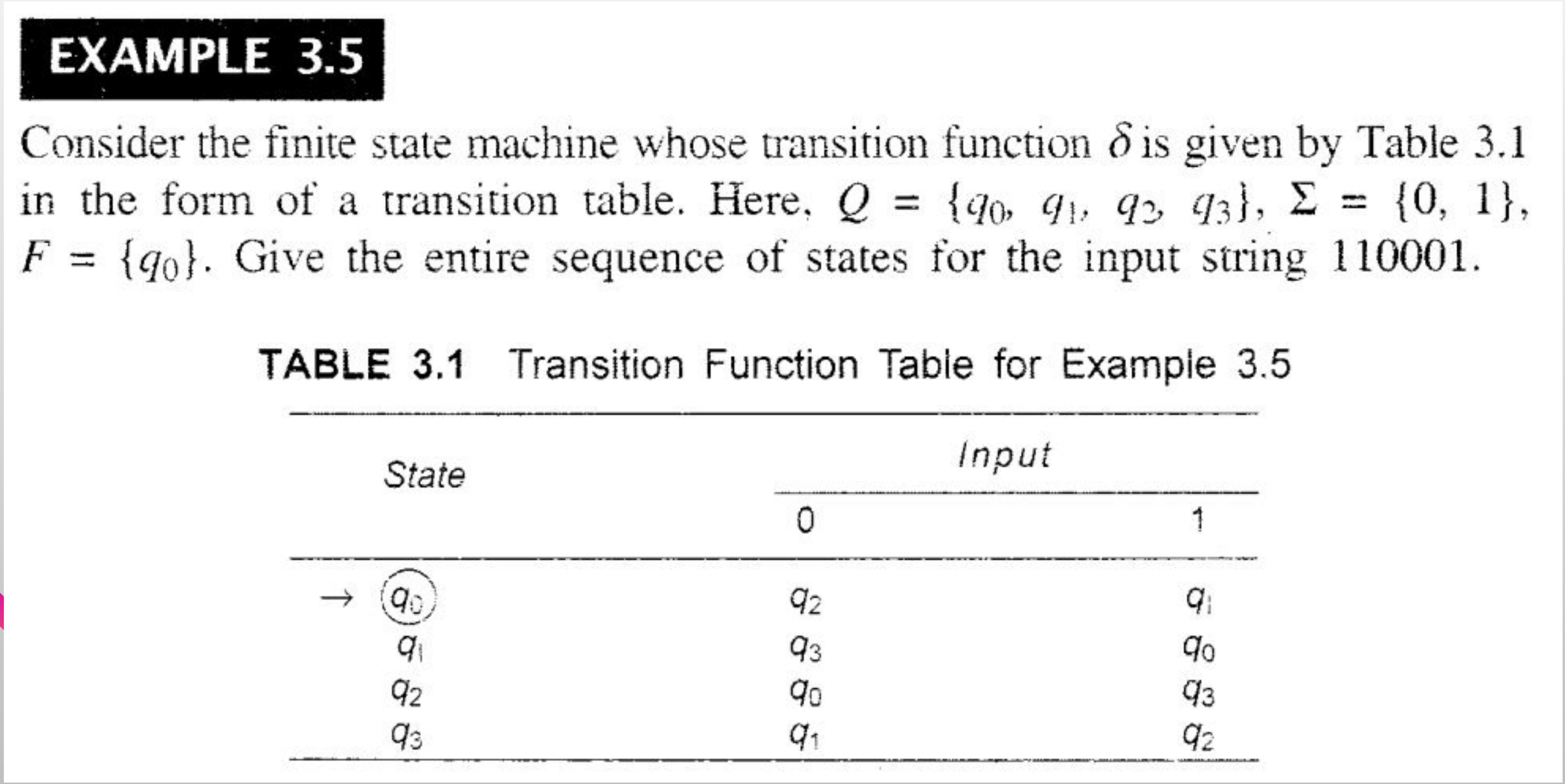
$$M = (Q, \sum, \delta, q_0, F)$$

if $\delta(q_0, x) = q$ for some $q \epsilon F$.

This is basically the acceptability of a string by the final state.

Note: A final state is also called an accepting state

ACCEPTABILITY OF A STRING BY A FINITE AUTOMATON



EXAMPLE 3.5

Consider the finite state machine whose transition function δ is given by Table 3.1 in the form of a transition table. Here, $Q = \{q_0, q_1, q_2, q_3\}$, $\Sigma = \{0, 1\}$, $F = \{q_0\}$. Give the entire sequence of states for the input string 110001.

TABLE 3.1 Transition Function Table for Example 3.5

State	*Input*	
	0	1
$\rightarrow \textcircled{q_0}$	q_2	q_1
q_1	q_3	q_0
q_2	q_0	q_3
q_3	q_1	q_2

Solution

$$
\begin{aligned}
\delta(q_0,\ \overset{\downarrow}{1}10101) &= \delta(q_1, \overset{\downarrow}{1}0101) \\
&\qquad\qquad\ \ \downarrow \\
&= \delta(q_0, \overset{}{0}101) \\
&\qquad\qquad\ \ \downarrow \\
&= \delta(q_2, 101) \\
&\qquad\qquad\ \ \downarrow \\
&= \delta(q_3, 01) \\
&\qquad\qquad\ \ \downarrow \\
&= \delta(q_1, 1) \\
&= \delta(q_0,\ \Lambda) \\
&= q_0
\end{aligned}
$$

Hence,

$$q_0 \xrightarrow{1} q_1 \xrightarrow{1} q_0 \xrightarrow{0} q_2 \xrightarrow{1} q_3 \xrightarrow{0} q_1 \xrightarrow{1} q_0$$

THE EQUIVALENCE OF DFA AND NDFA

The relation between DFA and NDFA is that:

1. A DFA can simulate the behaviour of NDFA by increasing the number of states. (In other words a DFA (Q, L, δ, qQ, F) can be viewed as an NDFA (Q, L, δ', qQ, F) by defining δ'(q, a) = {δ(q, a)}.)
2. Any NDFA is a more general machine without being more powerful.

Conversion of NFA→DFA problems

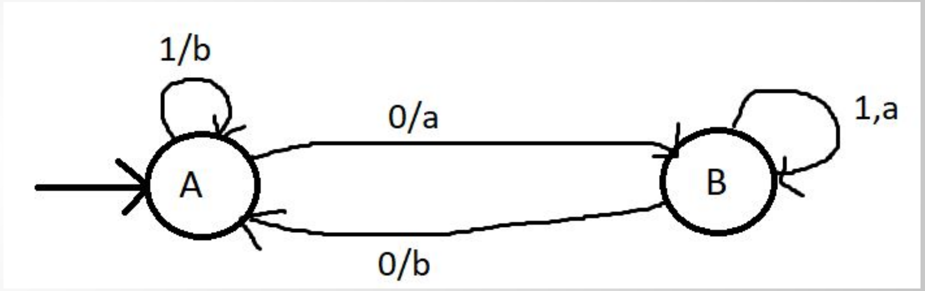
MEALY AND MOORE MODELS

Mealy Machines:

Mealy machines are also finite state machines with output value and its **output depends on the present state and current input symbol.**

It can be defined as (Q, q_0, ∑, △, δ, λ') where:

- Q is a finite set of states.
- q_0 is the initial state.
- ∑ is the input alphabet.
- △ is the output alphabet.
- δ is the transition function which maps Q × ∑ → Q.
- 'λ' is the output function that maps Q × ∑→ △.

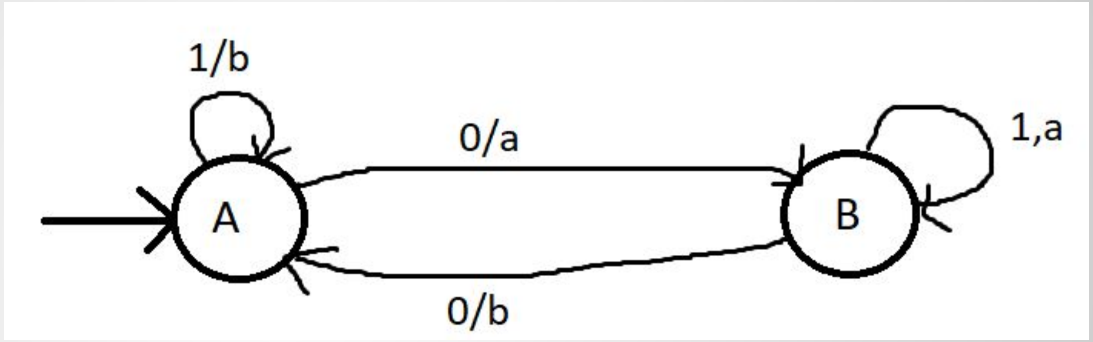


MEALY AND MOORE MODELS

Mealy Machines:

Eg. 1 0 1 0

$$\begin{array}{ccccccccc} A & \rightarrow & A & \rightarrow & B & \rightarrow & B & \rightarrow & A \\ & & b & & a & & a & & b \end{array}$$



Length of input : 4 $\Rightarrow$ n

Length of output : 4 $\Rightarrow$ n

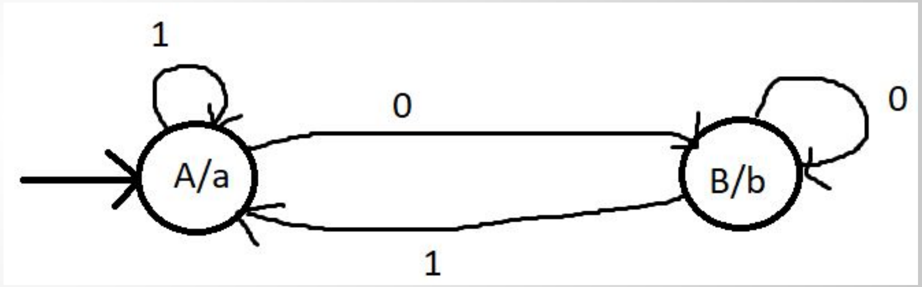
MEALY AND MOORE MODELS

Moore Machines:

They are finite state machines with output value and its **<u>output depends only on the present state</u>**.

It can be defined as (Q, q_0, ∑, △, δ, λ) where:

- Q is a finite set of states.
- q_0 is the initial state.
- ∑ is the input alphabet.
- △ is the output alphabet.
- δ is transition function which maps Q×∑ → Q.
- λ is the output function which maps Q → Q.

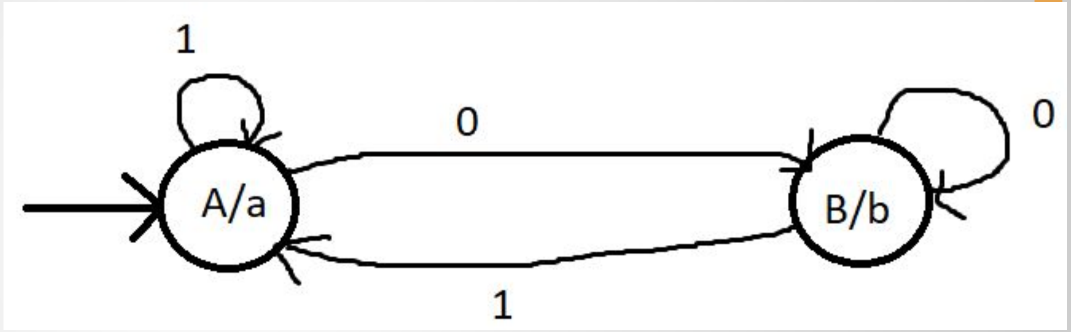


MEALY AND MOORE MODELS

Moore Machines:

Eg. 1 0 1 0

$$\begin{array}{ccccccccc} A & \rightarrow & A & \rightarrow & B & \rightarrow & A & \rightarrow & B \\ a & & a & & b & & a & & b \end{array}$$



Length of input : 4 ⇒ n

Length of output : 5 ⇒ n+1

FORMAL LANGUAGES

Grammar

It is a finite set of formal rules for generating syntactically correct sentences or meaningful correct sentences.

Formal Definition of Grammar :

Any Grammar can be represented by 4 tuples – (V_n, ∑, P, S)

V_n => Finite Non-Empty Set of Non-Terminal Symbols.

∑ => Finite Set of Terminal Symbols.

P => Finite Non-Empty Set of Production Rules.

S => Start Symbol (Symbol from where we start producing our sentences or strings).

DERIVATIONS AND THE LANGUAGE GENERATED BY A GRAMMAR

Consider Grammar G1 = $(V_n, \sum, P, S)$

N = {S, A} #Set of non-terminals Symbols

T = {a} #Set of terminal symbols

P = {A->Aa, A->AAa, A->a, A->ε} #Set of all production rules

S = {A} #Start Symbol

Language generated,

L(G)= set of strings starting with a

DERIVATIONS AND THE LANGUAGE GENERATED BY A GRAMMAR

Consider Grammar G1 = $(V_n, \sum, P, S)$

V ={S, A} #Set of non-terminal symbols

∑ = {0,1} #Set of terminal symbols

P = {A->A0,A->A1,A->0,A->1,A-> ε} #Set of all production rules

S = {A} #Start Symbol

Language generated,

L(G)= set of strings starting with a or b

DERIVATIONS AND THE LANGUAGE GENERATED BY A GRAMMAR

If G = ({S}, {0}, {S → SS}, S), find the language generated by G.

V={S}

∑ = {0}

P= {S →SS}

S={S}

Language generated,

L(G)= ϕ, since the only production S -> SS in G has no terminal on the right-hand side.

DERIVATIONS AND THE LANGUAGE GENERATED BY A GRAMMAR

Let $G = (\{S, A_1, A_2\}, \{a, b\}, P, S)$, where P consists of

$S \rightarrow aA_1A_2a$,

$A_1 \rightarrow baA_1A_2b$,

$A_2 \rightarrow A_1ab$,

$aA_1 \rightarrow baa$,

$bA_2b \rightarrow abab$

Test whether w = baabbabaaabbaba is in L(G).

DERIVATIONS AND THE LANGUAGE GENERATED BY A GRAMMAR

1. If G is $S \rightarrow aS \mid bS \mid a \mid b$, find L(G).
2. Let G = ({ S, C}, {a, b}, P, S), where P consists of $S \rightarrow aCa$, $C \rightarrow aCa \mid b$. Find L(G).
3. Test whether 001100, 001010, 01010 are in the language generated by the grammar $S \rightarrow 0S1 \mid 0A \mid 0 \mid 1B \mid 1$, $A \rightarrow 0A \mid 0$, $B \rightarrow 1B \mid 1$
4. Find the language generated by the grammar $S \rightarrow AB$, $A \rightarrow A1 \mid 0$, $B \rightarrow 2B \mid 3$.

CHOMSKY CLASSIFICATION OF LANGUAGES

According to Chomsky hierarchy, grammar is divided into 4 types as follows:

- Type 0 is known as unrestricted grammar.
- Type 1 is known as context-sensitive grammar.
- Type 2 is known as a context-free grammar.
- Type 3 Regular Grammar.

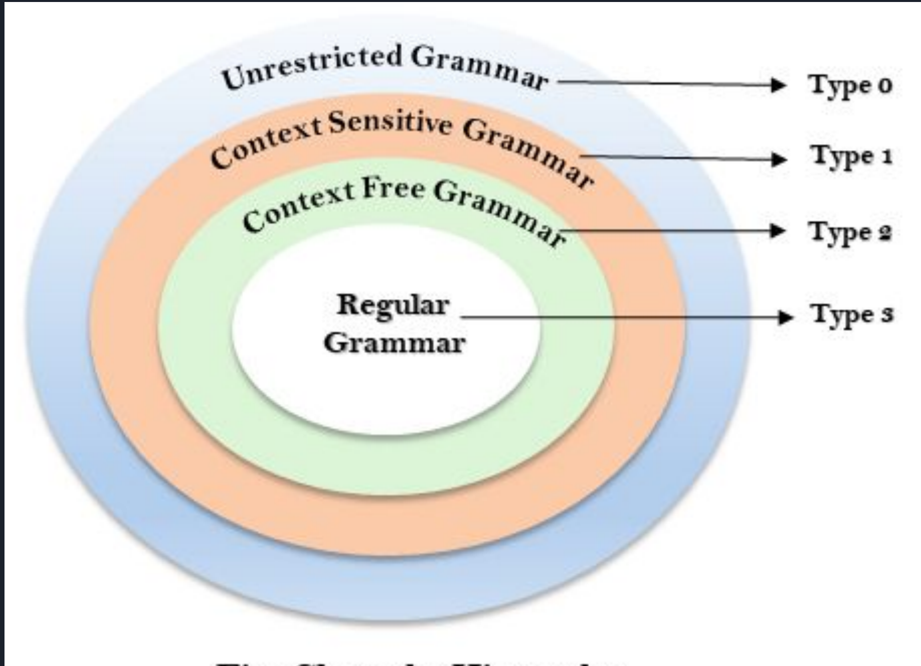


Fig: Chomsky Hierarchy

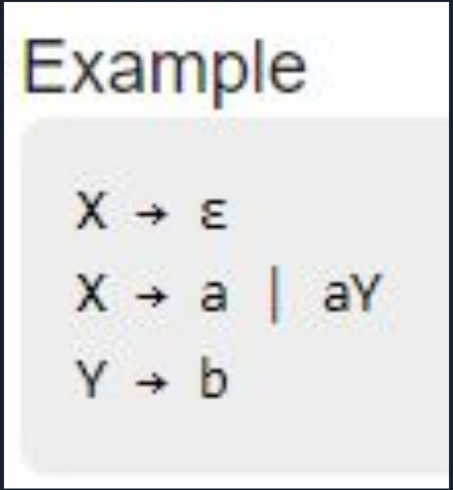
CHOMSKY CLASSIFICATION OF LANGUAGES

Type 3 - Regular Grammar

- Generate regular languages
- 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

$$X \rightarrow a \text{ or } X \rightarrow aY$$

where $X, Y \in N$ (Non terminal) & $a \in T$ (Terminal)
- The rule $S \rightarrow \varepsilon$ is allowed if S does not appear on the right side of any rule.
- Accepted by a finite-state automaton



Example

$X \rightarrow \varepsilon$

$X \rightarrow a \mid aY$

$Y \rightarrow b$

CHOMSKY CLASSIFICATION OF LANGUAGES

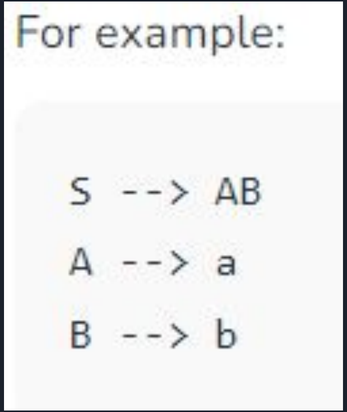
Type 2 - Context-Free Grammar(CFG)

- ➔ Generate context-free languages
- ➔ The productions must be in the form

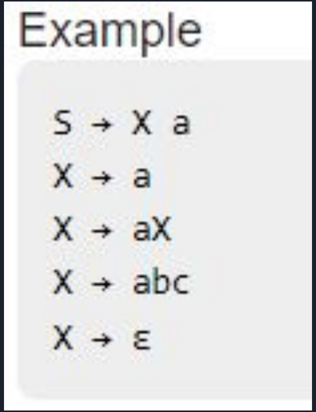
$$A \longrightarrow \gamma$$

where $A \in N$ (Non terminal) & $\gamma \in (T \cup N)^*$ (String of terminals and non-terminals).

- ➔ Recognized by a Pushdown automata


For example:

```
S --> AB
A --> a
B --> b
```


Example

```
S → X a
X → a
X → aX
X → abc
X → ε
```

CHOMSKY CLASSIFICATION OF LANGUAGES

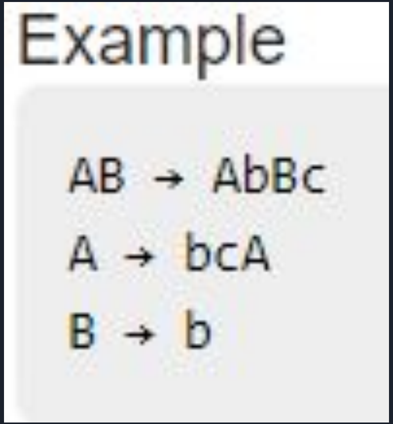
Type 1 - Context- Sensitive Grammar (CSG)

- ➔ Generate context-sensitive languages.
- ➔ The productions must be in the form

$$\alpha A \beta \rightarrow \alpha \gamma \beta$$

where $A \in N$ (Non terminal) & $\alpha, \beta, \gamma \in (T \cup N)^*$ (String of terminals and non-terminals).

- ➔ Recognized by a Linear Bound Automata


Example

$AB \rightarrow AbBc$

$A \rightarrow bcA$

$B \rightarrow b$

CHOMSKY CLASSIFICATION OF LANGUAGES

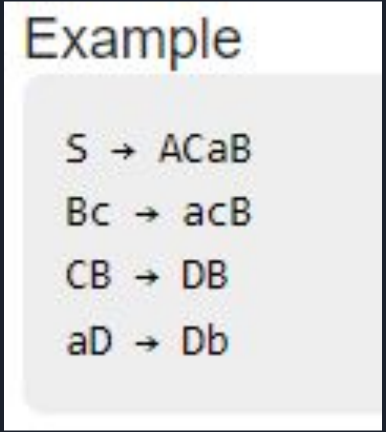
Type 0 - Unrestricted Grammar

- Generate recursively enumerable languages.
- Productions have no restrictions.
- Recognized by a Turing machine
- The productions can be in the form of

$$\alpha \longrightarrow \beta$$

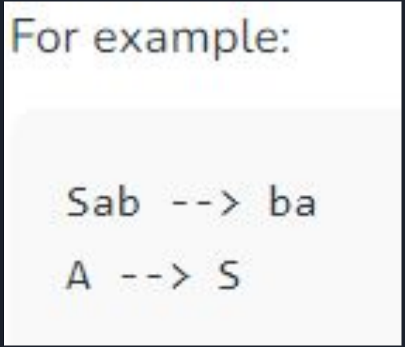
where α ⇒ a string of terminals and nonterminals with at least one non-terminal and α cannot be null.

β ⇒ a string of terminals and non-terminals.



Example

```
S  → ACaB
Bc → acB
CB → DB
aD → Db
```



For example:

```
Sab  --> ba
A  --> S
```

RECURSIVE AND RECURSIVELY ENUMERABLE SETS

- Recursive

  If L is a recursive language then –

  ➢ If $w \in L$ then a TM halts in a final state,

  ➢ If $w \notin L$ then TM halts in a non-final state.

- Recursively Enumerable(RE)

  If L is a recursive enumerable language then –

  ➢ If $w \in L$ then a TM halts in a final state,

  ➢ If $w \notin L$ then a TM halts in a non-final state or loops forever.

RECURSIVE AND RECURSIVELY ENUMERABLE SETS

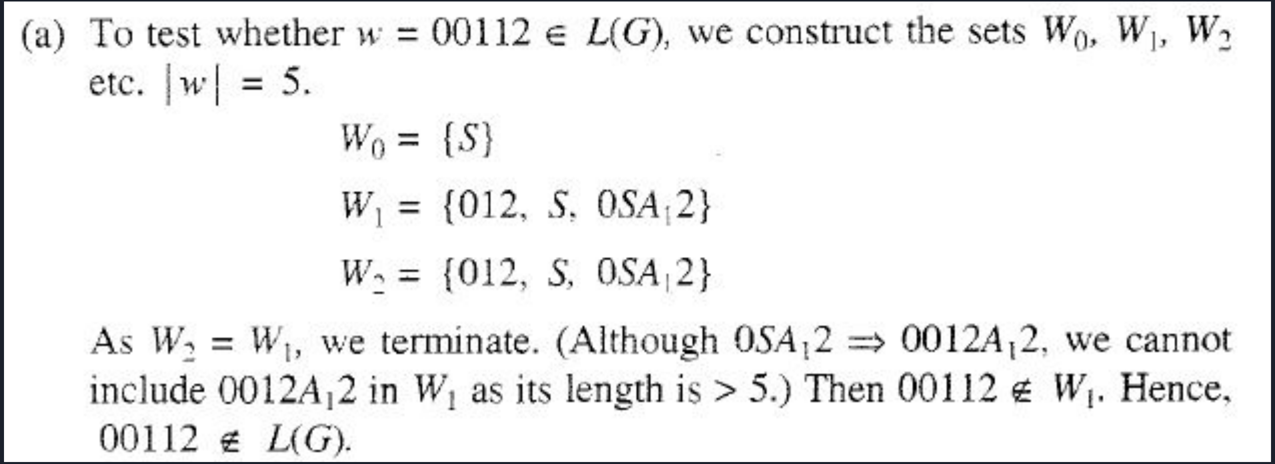
1. Consider the grammar G given by
   $S \rightarrow 0SA_1 2$,
   $S \rightarrow 012$,
   $2A_1 \rightarrow A_1 2$,
   $1A_1 \rightarrow 11$

Test whether
(a) 00112 E L(G)
(b) 001122 E L(G).
(c) Type of grammar/language?
c-> A context-sensitive grammar/language is recursive.[after check for strings in (a) and (b)]

RECURSIVE AND RECURSIVELY ENUMERABLE SETS

1. Consider the grammar G given by
   $S \to 0SA_12$,
   $S \to 012$,
   $2A_1 \to A_12$,
   $1A_1 \to 11$

Test whether
(a) 00112 E L(G)
(b) 001122 E L(G).



(a) To test whether $w = 00112 \in L(G)$, we construct the sets W_0, W_1, W_2 etc. $|w| = 5$.

$$W_0 = \{S\}$$
$$W_1 = \{012,\ S,\ 0SA_12\}$$
$$W_2 = \{012,\ S,\ 0SA_12\}$$

As $W_2 = W_1$, we terminate. (Although $0SA_12 \Rightarrow 0012A_12$, we cannot include $0012A_12$ in W_1 as its length is > 5.) Then $00112 \notin W_1$. Hence, $00112 \notin L(G)$.

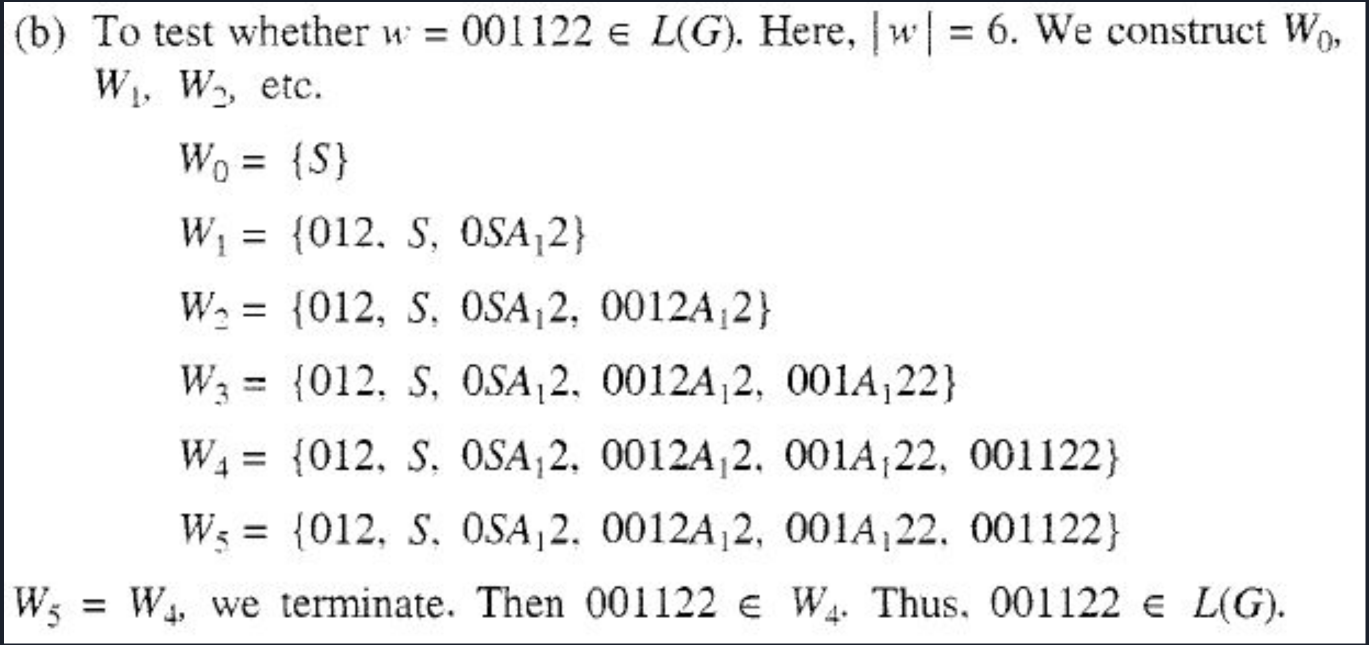
SOLUTION

RECURSIVE AND RECURSIVELY ENUMERABLE SETS

1. Consider the grammar G given by
   $S \rightarrow 0SA_1 2$,
   $S \rightarrow 012$,
   $2A_1 \rightarrow A_1 2$,
   $1A_1 \rightarrow 11$

Test whether
(a) 00112 E L(G)
(b) 001122 E L(G).

SOLUTION



(b) To test whether $w = 001122 \in L(G)$. Here, $|w| = 6$. We construct W_0, W_1, W_2, etc.

$$W_0 = \{S\}$$
$$W_1 = \{012,\ S,\ 0SA_12\}$$
$$W_2 = \{012,\ S,\ 0SA_12,\ 0012A_12\}$$
$$W_3 = \{012,\ S,\ 0SA_12,\ 0012A_12,\ 001A_122\}$$
$$W_4 = \{012,\ S,\ 0SA_12,\ 0012A_12,\ 001A_122,\ 001122\}$$
$$W_5 = \{012,\ S,\ 0SA_12,\ 0012A_12,\ 001A_122,\ 001122\}$$

$W_5 = W_4$, we terminate. Then $001122 \in W_4$. Thus, $001122 \in L(G)$.

OPERATIONS ON LANGUAGES

Kleene closure⇒∑* ⇒ the set of all possible strings of any length that can be formed using the symbols in ∑.

1. Complement
   Let L be a language over similar alphabet ∑
   The complement of L is denoted by L'
   Where L' = ∑* - L

Eg
Let A = {0, 01} , ∑ ={0, 1}
∑* ={ε,0,1,00,01,000,001,111, 0101.........}
Complement : A' = ∑* - A

OPERATIONS ON LANGUAGES

2. Union

Let L_1 and L_2 be two languages over a similar alphabet ∑

The union of L_1 and L_2 is defined as:

L_3: $A \cup B = \{w : w \in A \text{ or } w \in B\}$

I.e for every string $w \in \Sigma^*$, M accepts w ⇔ M1 accepts w or M2 accepts w

Eg.

Let $A = \{0, 01\}$ and $B = \{1, 10\}$.

union: $A \cup B = \{0, 01, 1, 10\}$

OPERATIONS ON LANGUAGES

3. Concatenation

Let L_1 and L_2 be two languages over a similar alphabet $\sum$

The concatenation of L_1 and L_2 is defined as:

L_3: $AB = \{ww' \;:\; w \in A \text{ and } w' \in B\}$

I.e where AB is the set of all strings obtained by taking an arbitrary string w in A and an arbitrary string w' in B then putting them together such that the former is to the left of the latter.

Eg.

Let A = {0, 01} and B = {1, 10}.

concatenation: AB = {01, 010, 011, 0110}

OPERATIONS ON LANGUAGES

4. Kleen closure

Let L a language over a alphabet ∑

The kleen closure of L is defined as:

$L^* = \{u_1, u_2, u_3, \ldots\ldots, u_k : k \geq 0 \text{ and } u_i \in A \text{ for all } i = 1, 2, \ldots, k\}$

Where A* is obtained by taking an infinite number of strings in A and putting them together.

Note that k cannot be zero, in this case it will correspond to an empty string ϵ and therefore ϵ ∈ A*.

Eg.

Let A = {0, 01}

kleen closure: A* = {ϵ, 0, 01, 00, 001, 010, 0101, 000, 0001, 00101, ...}

Regular Sets & Regular Grammar

TOC- UNIT 2

Regular Expressions & Regular Grammar

Regular Expressions are used to denote regular languages. An expression is regular if:

- ϕ is a regular expression for regular language ϕ.
- ε is a regular expression for regular language {ε}.
- If a ∈ Σ (Σ represents the input alphabet), a is regular expression with language {a}. I.e RE= a
- The union of two regular expressions R_1 and R_2 ,written as $R_1 + R_2$, is also a regular expression. **Eg L={a,b} then RE= a + b**

Regular Expressions & Regular Grammar

Regular Expressions are used to denote regular languages. An expression is regular if:

- The concatenation of two regular expressions R_1 and R_2, written as $R_1 R_2$, is also a regular expression.

  Eg L={00, 10} then RE = (0 + 1) 0

- The iteration (or closure) of a regular expression R written as R*, is also a regular expression.

  Eg L={ε, 0, 00,000} then RE = 0*

Regular Expressions & Regular Grammar

Regular Expressions are used to denote regular languages. An expression is regular if:

- If R is a regular expression, then (R) is also a regular expression.

  Eg: L={a} then RE= a or (a)

Regular Expressions & Regular Grammar

Order of evaluation of regular expressions

- Parenthesis ⇒ (R)
- Iteration (closure) ⇒ R*
- Concatenation ⇒ R1 R2
- Union ⇒ R1 + R2

Precedence order decreases from top to bottom

The parentheses used in Rule 5 influence the order of evaluation of a regular expression.

In the absence of parentheses the hierarchy of operations as follows: iteration (closure). concatenation, and union.

Regular Expressions & Regular Grammar

- **Regular Grammar :**

A grammar is regular if it has rules of form A -> a or A -> aB or A -> ε where ε is a special symbol called NULL.

- **Regular Languages :**

A language is regular if it can be expressed in terms of regular expression.

- **Regular Sets :**

Any set represented by a regular expression is called a regular set.

Exercise 1

Describe the following sets by regular expressions:

1) {101}
2) {abba},
3) {01, 10}
4) {Λ. ab}
5) {abb. a, b, bba}
6) {Λ, 0, 00, 000.... }
7) {1, 11 111,...... }.

Exercise 1 - Solution

Describe the following sets by regular expressions:

1) {101} ⇒ {101} is represented by 101

{1} and {0} are represented by 1 and 0 respectively.
101 is obtained by concatenating 1 0 and 1
So {101} is represented by 101.

2) {abba} ⇒ abba represents {abba}

3) {01, 10} ⇒ {01, 10} represented by 01 + 10
As {01, 10} is the union of {01} and {10}, we have 01 + 10

Exercise 1 - Solution

Describe the following sets by regular expressions:

4) {Λ, ab} ⇒ represented by A + ab

5) {abb. a, b, bba} ⇒ represented by abb + a + b + bba

6) {Λ, 0, 00, 000.... } ⇒ represented by 0*

7) {1, 11 111,...... } ⇒ represented by $1(1)^*$ or 1^+

Exercise 2

Describe the following sets by regular expressions:

(a) L 1 =the set of all strings of 0's and 1's ending in 00.

(b) L2 = the set of all strings of 0's and 1's beginning with 0 and ending with 1.

(c) L3 ={Λ, 11, 1111, 111111, ...}.

Exercise 2 - Solution

Describe the following sets by regular expressions:

(a) L 1 =the set of all strings of 0's and 1's ending in 00.

Solution: $(0 + 1)^* 00$.

(b) L2 = the set of all strings of 0's and 1's beginning with 0 and ending with 1.

Solution: $0(0 + 1)^* 1$

(c) L3 ={Λ, 11, 1111, 111111, ...}. **Solution:** $(11)^*$

IDENTITIES FOR REGULAR EXPRESSIONS

- ★ $\varnothing + R = R$
- ★ $\varnothing R = R\varnothing = \varnothing$
- ★ $\Lambda R = \Lambda R = R$
- ★ $\Lambda^* = \Lambda$ and $\varnothing^* = \Lambda$
- ★ $R + R = R$
- ★ $R^*R^* = R^*$
- ★ $RR^* = R^*R$
- ★ $(R^*)^* = R^*$
- ★ $\Lambda + RR^* = R^* = \Lambda + R^*R$
- ★ $(PQ)^*P = P(QP)^*$
- ★ $(P + Q)^* = (P^*Q^*)^* = (P^* + Q^*)^*$
- ★ $(P + Q)R = PR + QR$ **and** $R(P + Q) = RP + RQ$

Exercise

If R1 = $(1 + 011)^*$ which represents language of strings in which every 0 is immediately followed by at least two 1 's.

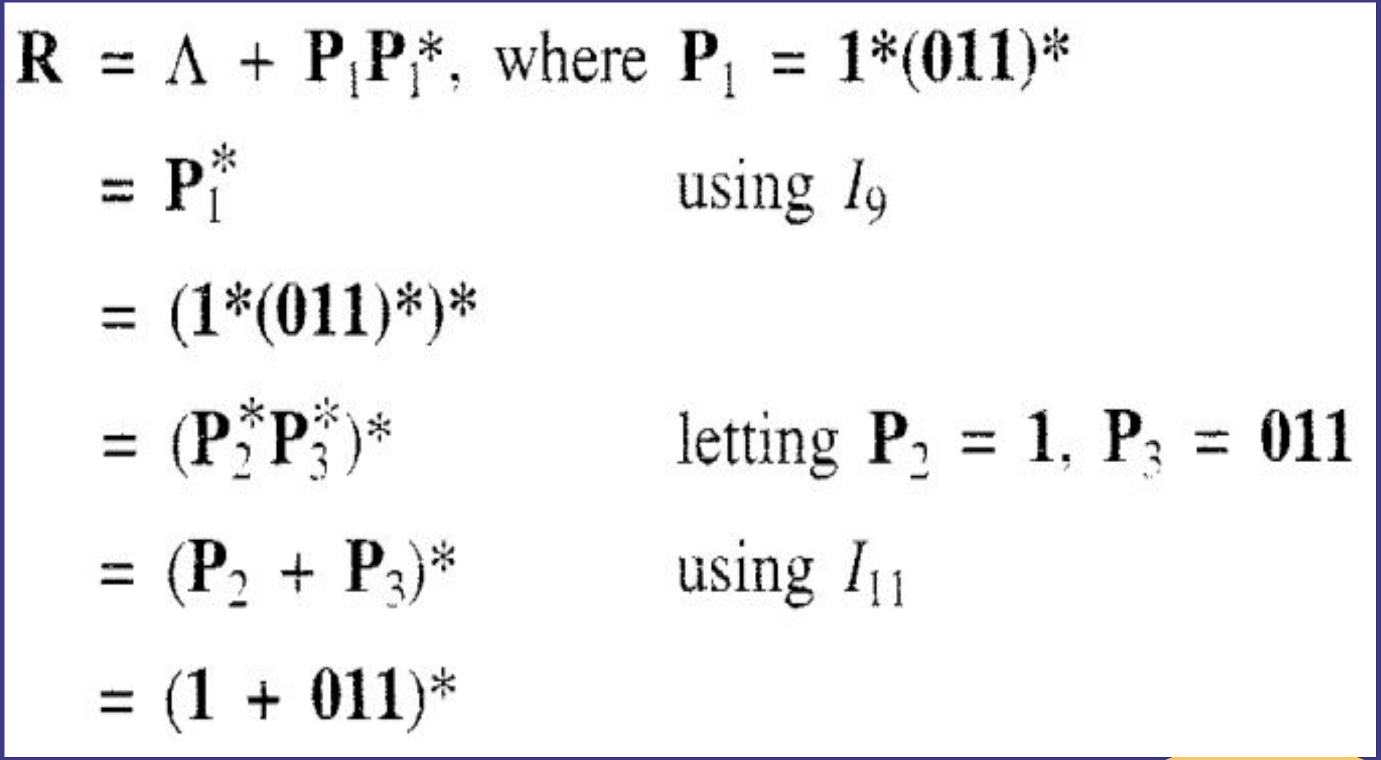
Prove that the regular expression R2 = $\Lambda + 1^*(011)^*(1^*(011)^*)^*$ also describes the same set of strings.

Exercise

If R1 = (1 + 011)* which represents language of strings in which every 0 is immediately followed by at least two 1's.

Prove that the regular expression R2 =A + 1*(011)*(1*(011)*)* also describes the same set of strings.

Solution



$$R = \Lambda + P_1 P_1^*, \text{ where } P_1 = 1^*(011)^*$$

$$= P_1^* \quad \text{using } I_9$$

$$= (1^*(011)^*)^*$$

$$= (P_2^* P_3^*)^* \quad \text{letting } P_2 = 1,\ P_3 = 011$$

$$= (P_2 + P_3)^* \quad \text{using } I_{11}$$

$$= (1 + 011)^*$$

Exercise

1. Prove that P + PQ*Q =a*bQ* where P =b + aa*b and Q is any regular expression.
2. Prove that (0*1*)* is the same as (0 + 1)*
3. Prove (1 + 00*1) + (1 + 00*1)(0 + 10*1)* (0 + 10*1) = 0*1(0 + 10*1)*
4. Prove the following identity: (a*ab + ba)*a* = (a + ab + ba)*

ARDEN'S THEOREM

Let P and Q be two regular expressions over ∑. If P does not contain ∈, then the following equation in R, namely R = Q + RP has a unique solution (i.e. one and only one solution) given by R = QP*.

ARDEN'S THEOREM

1. proof R = QP* is the solution of R = Q + RP

R = Q + RP ……(i)
Now, replacing R by R = QP*, we get,
R = Q + QP*P

Taking Q as common,
R = Q(∈ + P*P) = QP*

(As we know that ∈ + R*R = R*). Hence proved. Thus, R = QP* is the solution of the equation R = Q + RP.

ARDEN'S THEOREM

2. proof $R = QP^*$ is the unique solution of $R = Q + RP$

Let's take this equation again:

$R = Q + RP$

Now, replace R by $R = Q + RP$,

$$\begin{aligned} R &= Q + (Q + RP)P \\ &= Q + QP + RP^2 \end{aligned}$$

Again, replace R by $R = Q + RP$:-

$$\begin{aligned} R &= Q + QP + (Q + RP)\,P^2 \\ &= Q + QP + QP^2 + RP^3 \end{aligned}$$

Asst. Prof. Jesica D'cruz

ARDEN'S THEOREM

$= Q + QP + QP^2 + .. + QP^n + RP^{(n+1)}$

Now, replace R by R = QP*, we get,

$R = Q + QP + QP^2 + .. + QP^n + QP^*P^{(n+1)}$

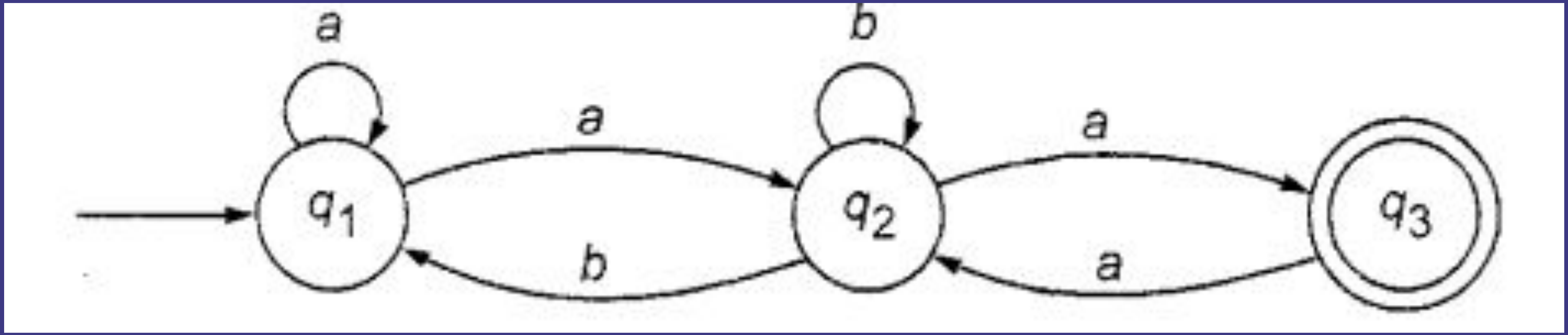
Taking Q as common,

$R = Q(\in + P + P^2 + .. + P^n + P^*P^{(n+1)} = QP^*$ [As $\in + P + P^2 + .. + P^n + P^*P^{(n+1)}$ represent the closure of P]

Thus, R = QP* is the unique solution of the equation R = Q + RP.

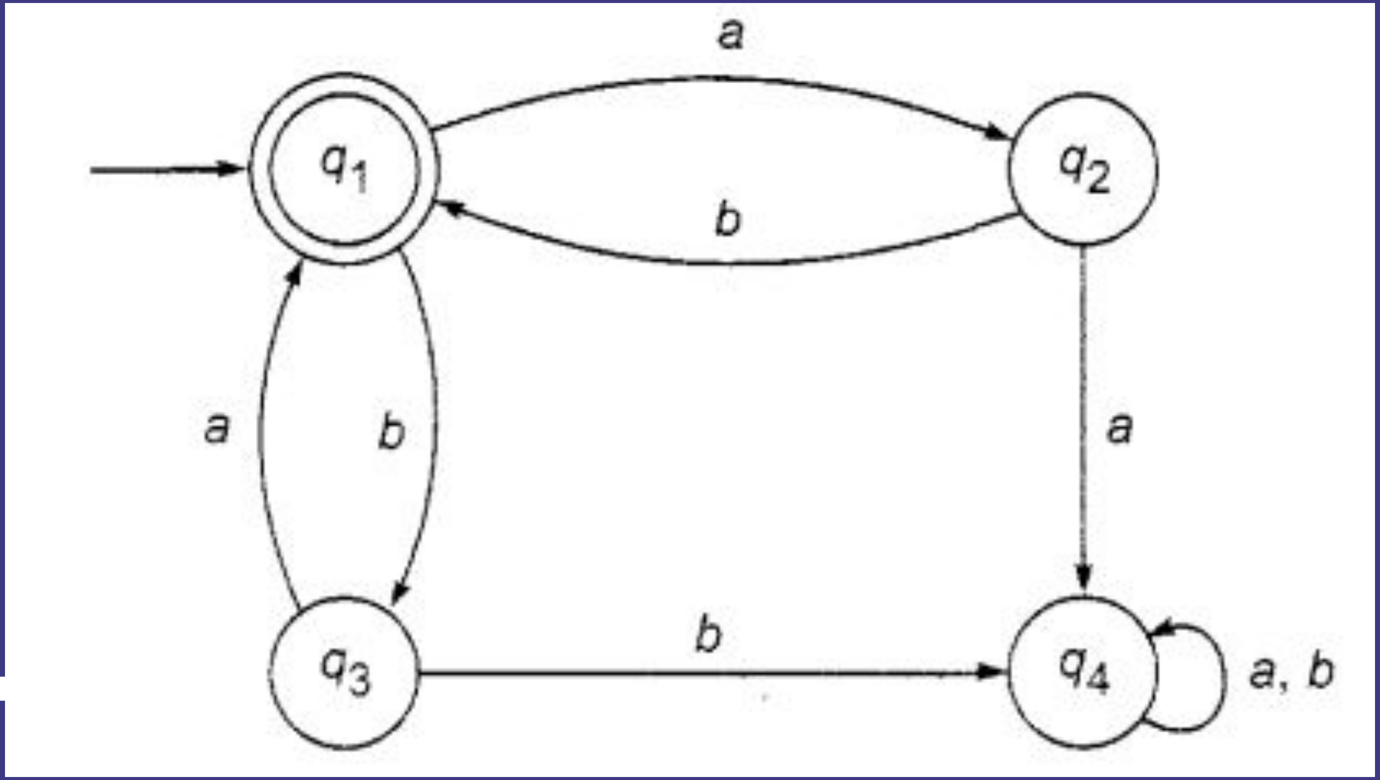
EXERCISES

1. Consider the transition system given in below figure.Prove that the strings recognized are (a + a(b + aa)*b)* a(b + aa)* a.


$$\begin{tikzcd}
\arrow[r] & q_1 \arrow[loop above, "a"] \arrow[r, bend left, "a"] & q_2 \arrow[loop above, "b"] \arrow[l, bend left, "b"] \arrow[r, bend left, "a"] & q_3 \arrow[l, bend left, "a"]
\end{tikzcd}$$

EXERCISES

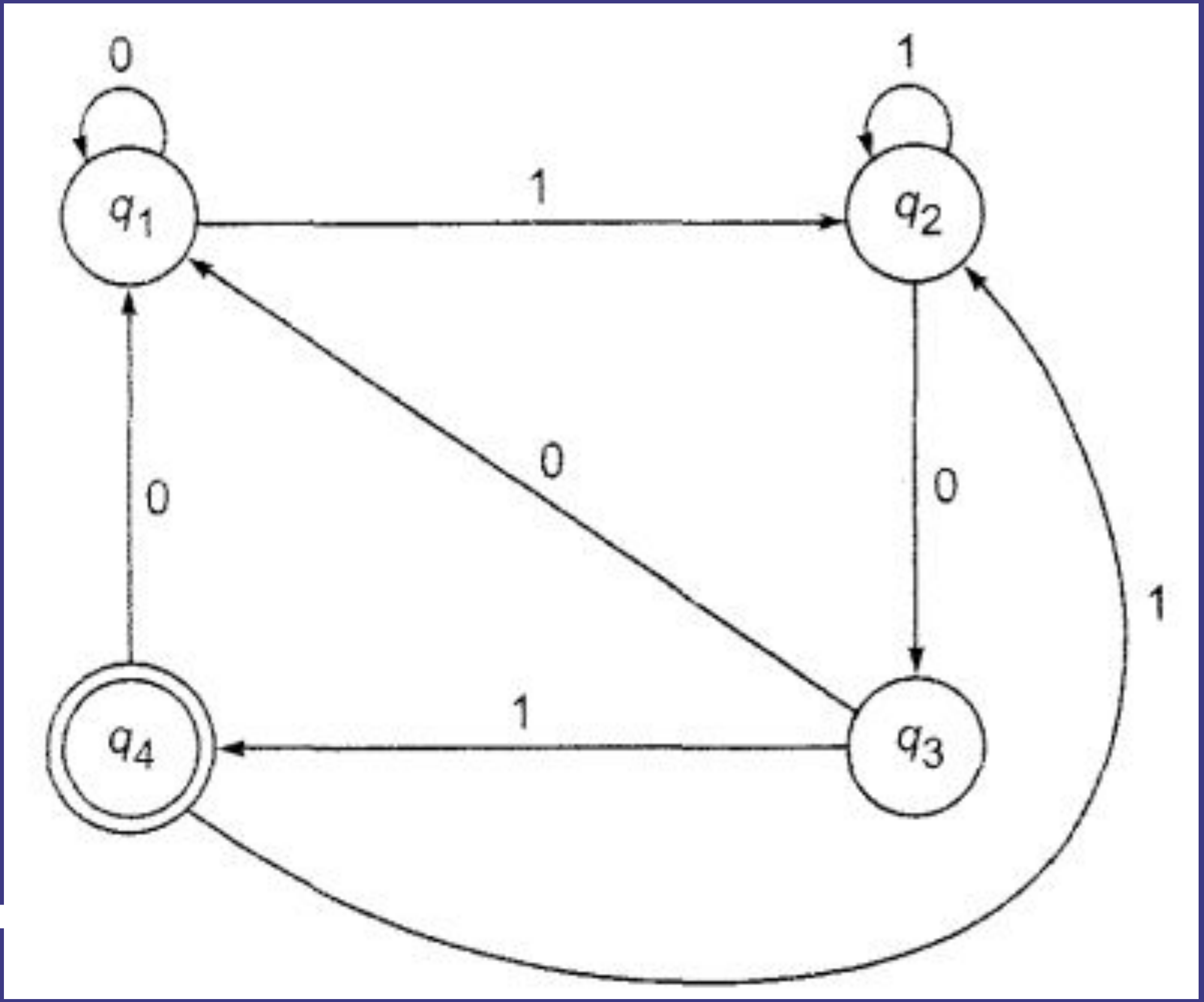
2. Prove that the finite automaton whose transition diagram is as shown in shown figure accepts the set of all strings over the alphabet {a, b} with an equal number of a's and b's, such that each prefix has at most one more a than the b's and at most one more b than the a's.


$$\begin{tikzcd}
\arrow[r] & q_1 \arrow[rr, bend left, "a"] \arrow[d, bend left, "b"] & & q_2 \arrow[ll, bend left, "b"] \arrow[d, "a"] \\
& q_3 \arrow[u, bend left, "a"] \arrow[rr, "b"] & & q_4 \arrow[loop right, "{a, b}"]
\end{tikzcd}$$

Asst. Prof. Jesica D'cruz

EXERCISES

3. Find the regular expression corresponding to Figure.



Exercise: Convert RE to Finite state automata

Construct the finite automaton equivalent to the regular expression

1. (0 + 1)*(00 + 11)(0 + 1)*
2. 10 + (0 + 11)0*1
3. (ab + c*)*b
4. (a + b)*abb.

PUMPING LEMMA

- ➔ It gives a method for pumping (generating) many substrings from a given string.
- ➔ It gives necessary condition(s) to prove a set of strings is not regular.

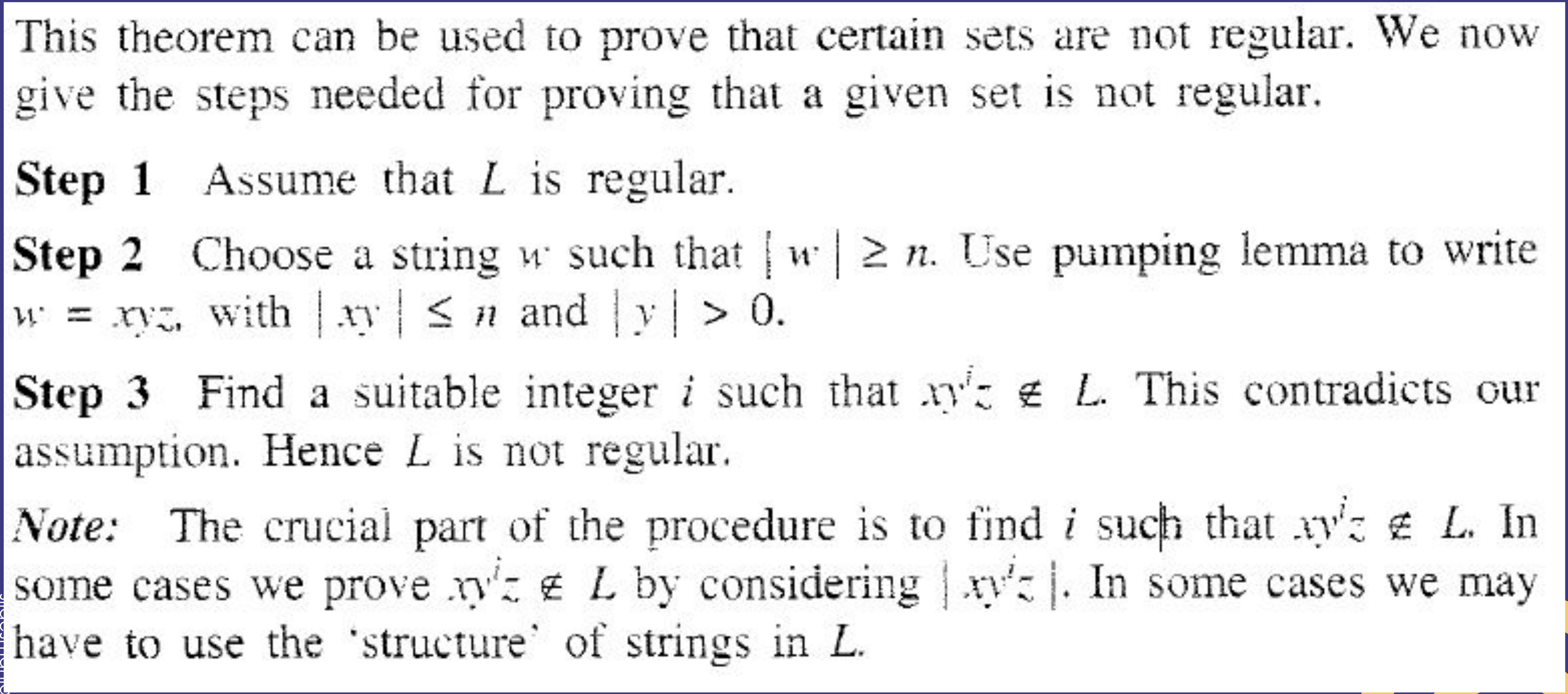
If A is a Regular Language, then A has a Pumping Length 'P' such that any string 'S" where |S| >= P may be divided into 3 parts S=xyz such that the following conditions must be true:

(1) $xy^iz \in A$ for every $i>=0$

(2) $|y|>0$

(3) $|xy| <= P$

APPLICATIONS OF PUMPING LEMMA



This theorem can be used to prove that certain sets are not regular. We now give the steps needed for proving that a given set is not regular.

Step 1 Assume that L is regular.

Step 2 Choose a string w such that $|w| \geq n$. Use pumping lemma to write $w = xyz$, with $|xy| \leq n$ and $|y| > 0$.

Step 3 Find a suitable integer i such that $xy^iz \notin L$. This contradicts our assumption. Hence L is not regular.

Note: The crucial part of the procedure is to find i such that $xy^iz \notin L$. In some cases we prove $xy^iz \notin L$ by considering $|xy^iz|$. In some cases we may have to use the 'structure' of strings in L.

REGULAR EXPRESSION TO FSA

Convert Regular expression to FSA:

- (ab + a)*(aa + b)
- (a*b + b*a)*a
- a* + (ab + a)*
- a(a + b)*ab
- a*b + b*a
- (aa + b)*(bb + a)*

CLOSURE PROPERTY FOR REGULAR SETS

Any 2 regular languages L1 & L2 are closed under:

- Union

If L1 and If L2 are two regular languages, their union L1 U L2 will also be regular.

- Intersection

If L1 and If L2 are two regular languages, their intersection L1 ∩ L2 will also be regular.

- Concatenation

If L1 and If L2 are two regular languages, their concatenation L1.L2 will also be regular.

CLOSURE PROPERTY FOR REGULAR SETS

Any 2 regular languages L1 & L2 are closed under:

- Kleene Closure

If L1 is a regular language, its Kleene closure L1* will also be regular.

- Complement

If L(G) is a regular language, its complement L'(G) will also be regular. Complement of a language can be found by subtracting strings which are in L(G) from all possible strings.

- Reversal

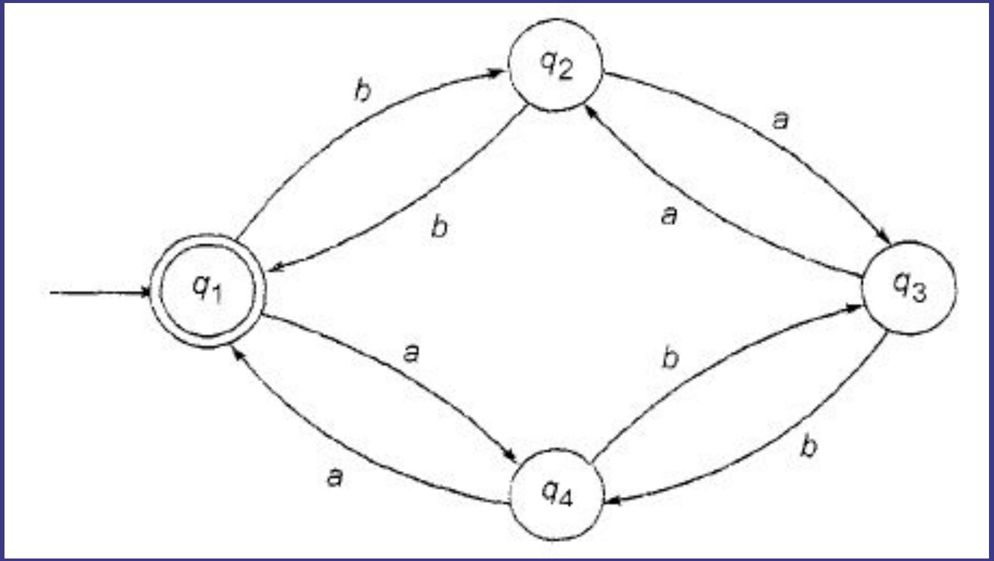
Given language L, LR is the set of strings whose reversal is in L.

Example: L = {0, 01, 100}   LR = {0, 10, 001}

CONSTRUCTION OF A REGULAR GRAMMAR FROM DFA & VICE VERSA

1. Construct a finite automaton recognizing L(G), where G is the grammar $S \rightarrow aS \mid bA \mid b$ and $A \rightarrow aA \mid bS \mid a$.

2. Construct Regular grammar from DFA



Asst. Prof. Jesica D'cruz

TURING MACHINE

UNIT 3

CONTENTS

VARIATIONS OF TURING MACHINE

NOTE: All variations of turing machine have the same power

1. **Turing Machine with Stay option:**
   If instead of moving left or right on seeing an input, the head could also stay at one position without moving anywhere i.e.

   f: Q × X --> Q × X × {Left_shift, Right_shift, Stay}

2. Turing Machine with Semi-infinite tape

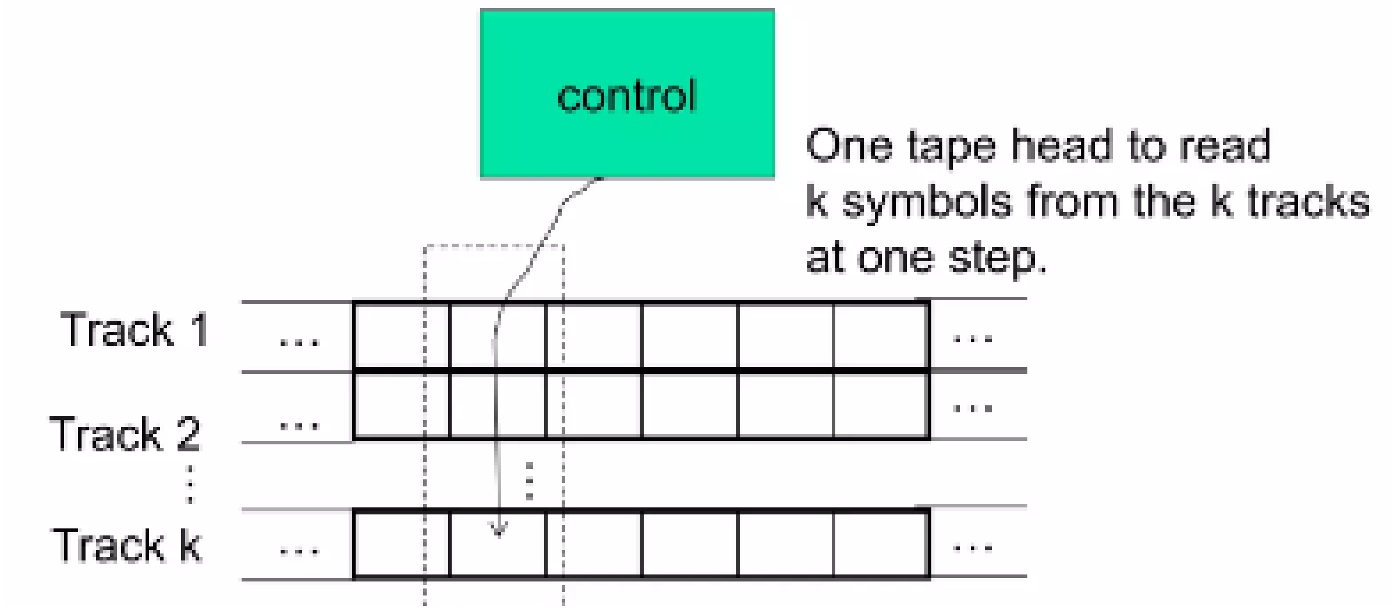
Turing machine has an infinite input tape with extends in both the directions (left and right) infinitely. So now if we restrict it to extend only in one direction and not in both the directions, i.e., we make the tape to be semi infinite. Eg. FSA

VARIATIONS OF TURING MACHINE

NOTE: All variations of turing machine have the same power

3. Multi-track TM

A k-tack Turing machine (for some k>0) has k-tracks and one R/W head that reads and writes all of them one by one. A k-track Turing Machine can be simulated by a single track Turing machine


control
One tape head to read k symbols from the k tracks at one step.
Track 1
Track 2
Track k
...

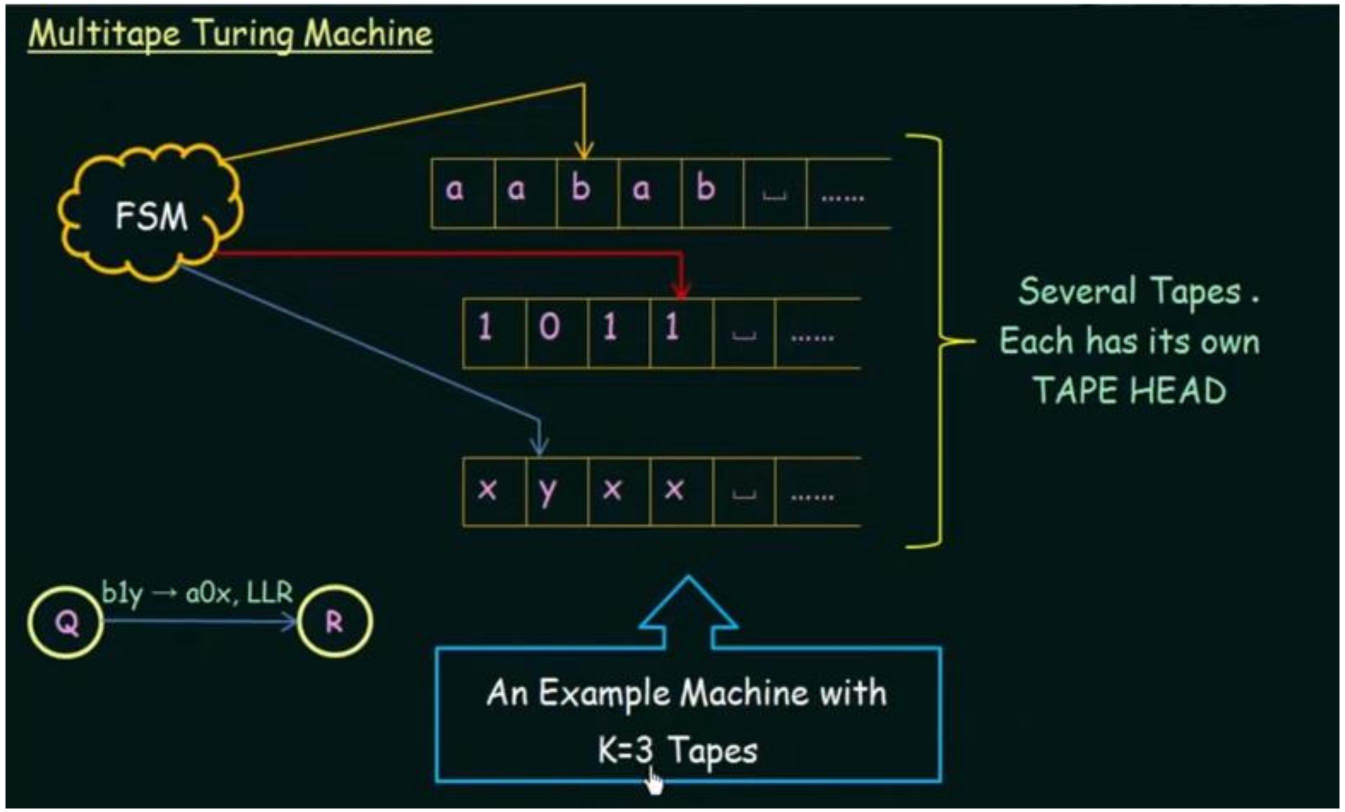
VARIATIONS OF TURING MACHINE

NOTE: All variations of turing machine have the same power

4. Multi tape

A Turing machine with several tapes we call it a multi tape Turing machine.

Every tape's have their own Read/Write head

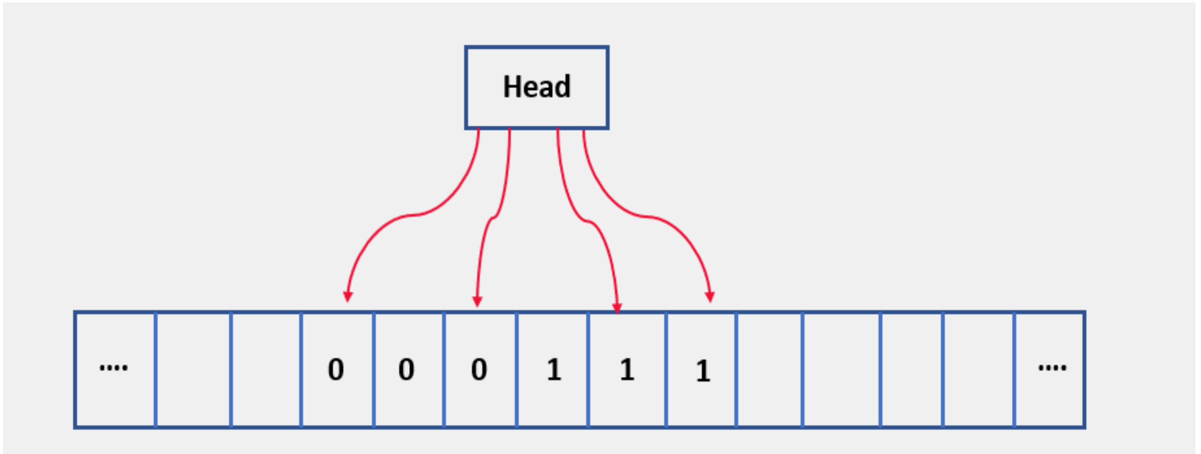


VARIATIONS OF TURING MACHINE

NOTE: All variations of turing machine have the same power

5. Multi head

- A multi-head Turing machine contains two or more heads to read the symbols on the same tape.
- In one step all the heads sense the scanned symbols and move or write independently.
- Multi-head Turing machine can be simulated by single head Turing machine.



Multi-head Turning Machine

DECIDABILITY & UNDECIDABILITY

- **DECIDABLE LANGUAGE**

1. A problem with two answers (Yes/No) is decidable if the corresponding language is recursive. In this case, the language L is also called decidable.
2. All recursive languages are Turing decidable or decidable languages

- **PARTIALLY DECIDABLE**

1. All Recursively enumerable languages are partially decidable and Turing recognizable

- **UNDECIDABLE LANGUAGE**

1. A problem language is undecidable if it is not decidable.
2. No Turing machine can be designed for that language

DECIDABLITY TABLE

S.NO	PROBLEM	REGULAR	DCFL	CFL	CSL	REC	RE
1.	Is w ∈ L ? Membership Problem	D	D	D	D	D	UD
2.	Is L = ϕ ? Emptiness Problem	D	D	D	UD	UD	UD
3.	Is L = Finite or not ? Finiteness Problem	D	D	D	UD	UD	UD
4.	Is L1=L2? Equivalence Problem	D	UD	UD	UD	UD	UD
5.	Is L1 ^ L2=ϕ ? Intersection Empty Problem	D	UD	UD	UD	UD	UD
6.	Is Σ* ? Totality / Completeness Problem	D	D	UD	UD	UD	UD
7.	Is L1 $\underline{C}$ L2 ? Subset roblem	D	UD	UD	UD	UD	UD
8.	L1 ^ L2= Finite ? Co-Finiteness Problem	D	D	UD	UD	UD	UD
9.	Is (Σ*-L) = Finite ? Co-Finiteness Problem	D	D	UD	UD	UD	UD
10.	Is L = Regular ? Regularity Problem	D	D	UD	UD	UD	UD
11.	Ambiguity Problem	D	UD	UD	UD	UD	UD
12.	Is complement of a language is of same type or not ?	D	D	UD	D	D	UD

NOTE :

Regular – Decidable for all problems

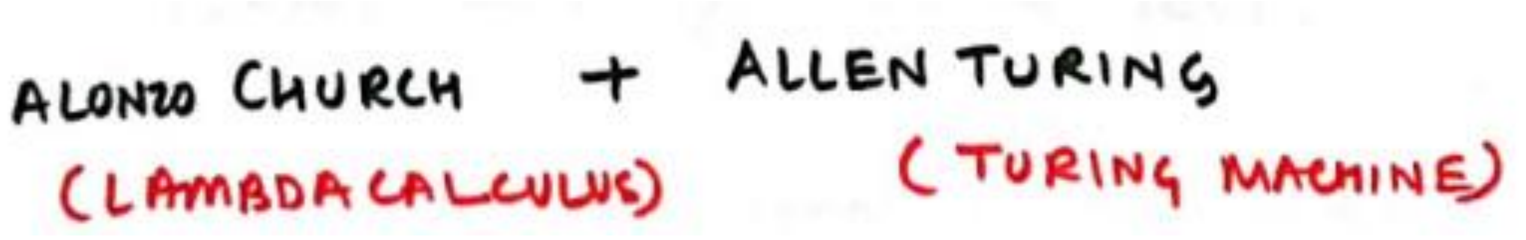
REL – Undecidable problems

CSL & Recursive – Decidable for Membership & Complement of language

CFL – MEF

CHURCH’S TURING THESIS

- Originates from meaning of "COMPUTABLE"?



ALONZO CHURCH + ALLEN TURING

(LAMADA CALCULUS) (TURING MACHINE)

- Turing machines(precise definition) and λ-calculus(intutive notions) are equivalent models of computation.
- Both were meant to say that any machine/algorithm that is recognized by their machines they are computable.
- But as a standard way computable means problems that can be computable by Turing machine

UNIVERSAL TURING MACHINE

1. A Turing machine is said to be universal Turing machine if it can accept:

- The input data, and
- An algorithm (description) for computing.

2. This is precisely what a general-purpose digital computer does. A digital computer accepts a program written in high level language.

3. Thus, a general purpose Turing machine will be called a universal Turing machine if it is powerful enough to simulate the behavior of any digital computer, including any Turing machine itself.

4. Notation of universal turing machine

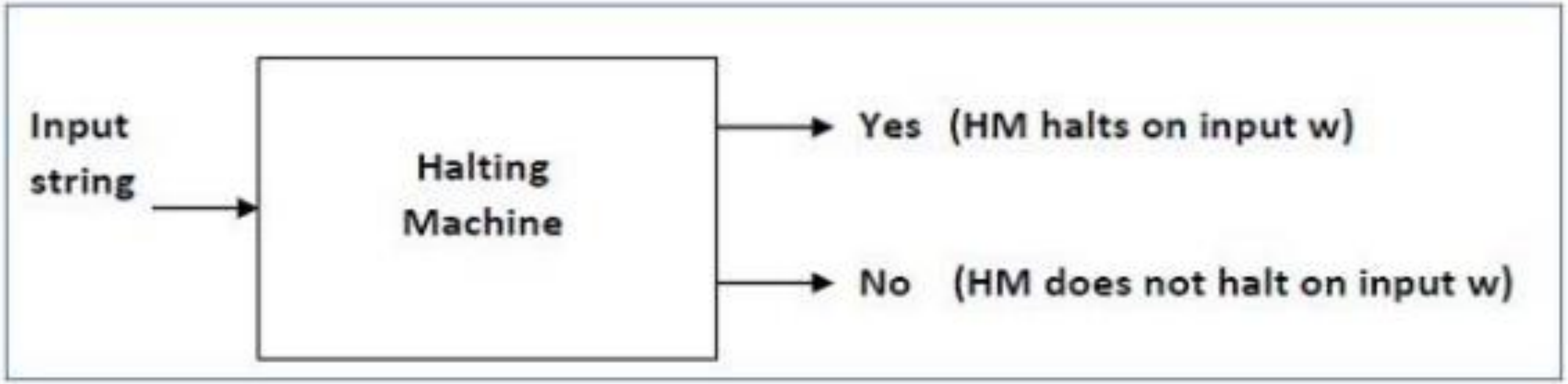
U={M, ∑ | where M =>description of turing machine, and ∑ =>input symbol }

HALTING PROBLEM

Input – A Turing machine and an input string **w**.

Problem – Does the Turing machine finish computing of the string **w** in a finite number of steps? The answer must be either **yes or no.**

Proof – At first, we will assume that such a Turing machine exists to solve this problem and then we will show it is contradicting itself. We will call this Turing machine as a **Halting machine** that produces a 'yes' or 'no' in a finite amount of time. If the halting machine finishes in a finite amount of time, the output comes as 'yes', otherwise as 'no'. The following is the block diagram of a Halting machine –

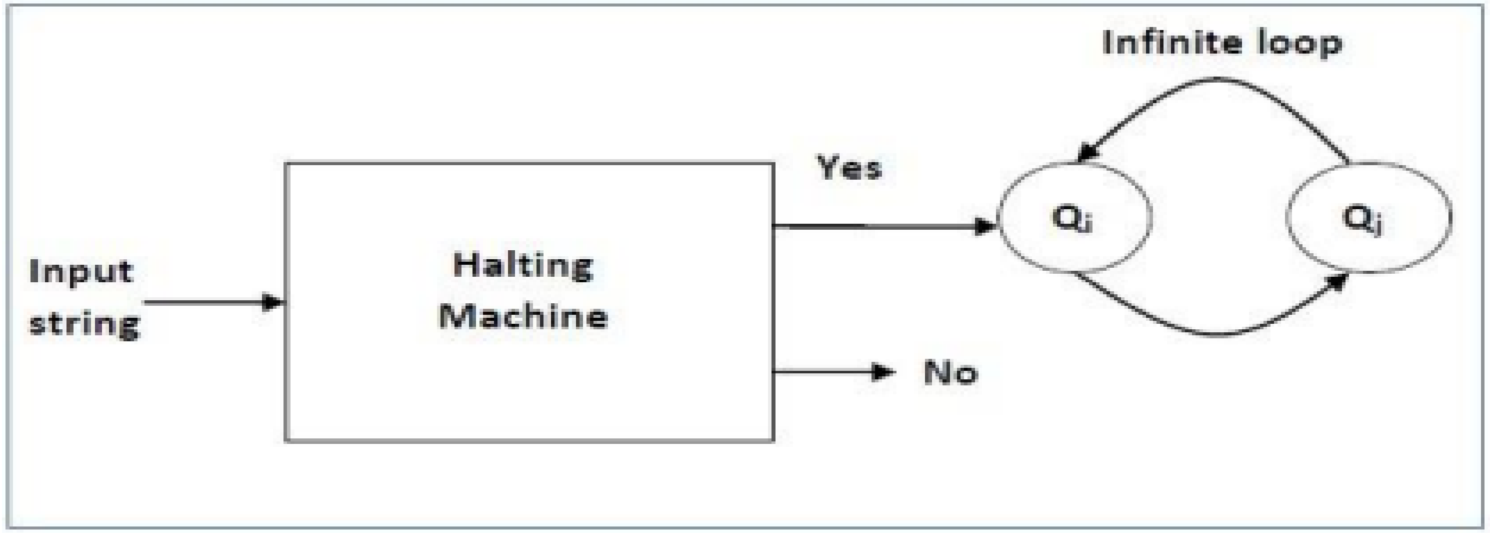


HALTING PROBLEM

Now we will design an **inverted halting machine (HM)'** as –

- If **H** returns YES, then loop forever.
- If **H** returns NO, then halt.

The following is the block diagram of an 'Inverted halting machine' –



Further, a machine **(HM)₂** which input itself is constructed as follows –

- If (HM)₂ halts on input, loop forever.
- Else, halt.

Here, we have got a contradiction. Hence, **the halting problem is undecidable.**