THEORY OF COMPUTATION

UNIT - 1

K CHAIRMADURAI ME,.. AP
Department of Computer Science and Engineering
Adhiparasakthi Engineering College , Melmaruvathur

Unit I Automata Fundamentals

Syllabus:

- **Introduction to Formal Proof**
  - **Additional forms of Proof**
  - **Inductive Proofs**
  - **Finite Automata**
  - **Deterministic Finite Automata**
  - **Non-deterministic Finite Automata**
  - **Finite Automata with Epsilon Transitions**

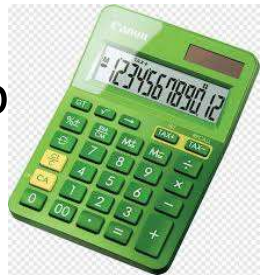
Introduction



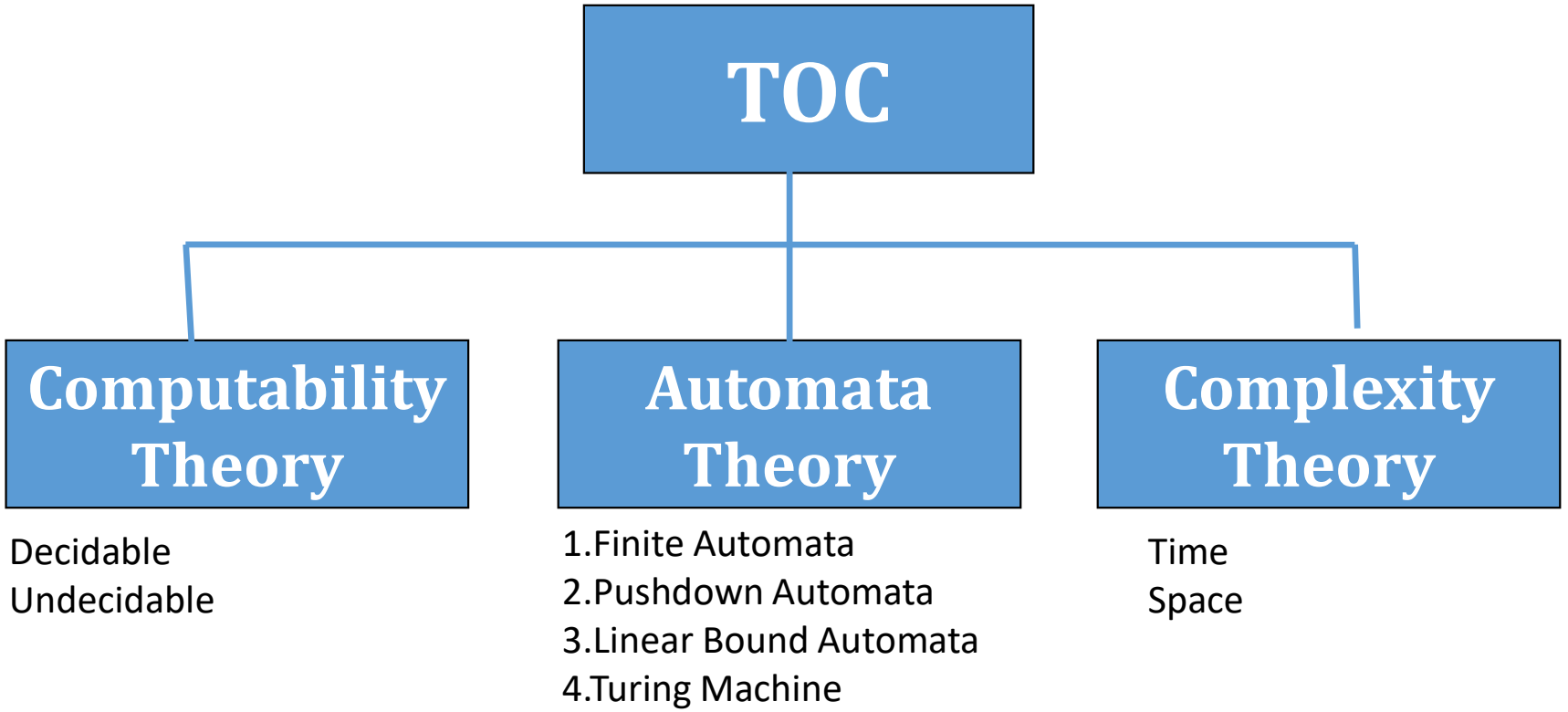
- Very important fundamental course of Computer Science.

- The **theory of computation** is the branch that deals with how efficiently problems can be solved on a model of computation, using an algorithm.

- A **model of computation** is a model which describes how an output o a mathematical function is computed given an input.



Introduction



Complexity theory

"What makes some problems computationally hard and others easy?"

- Computer problems come in different varieties; some are easy, and some are hard.
  - For example, the sorting problem is an easy one.
  - The scheduling problem seems to be much harder than the sorting problem.

Computability theory

- Certain basic problems cannot be solved by computers.
  - One example, the problem of determining whether a mathematical statement is true or false.
- The theories of computability and complexity are closely related.
  - In complexity theory, the objective is to classify problems as *easy ones* and *hard ones*;
  - Whereas in computability theory, the classification of problems is by those that are *solvable* and those that are not.

Automata theory

- Automata theory deals with the definitions and properties of mathematical models of computation.
  - These models play a role in several applied areas of computer science.
    - ✓ One model, called the *finite automaton*, is used in text processing, compilers, and hardware design.
    - ✓ Another model, called the *context-free grammar*, is used in programming languages and artificial intelligence.

Theory of Computation: A Historical Perspective

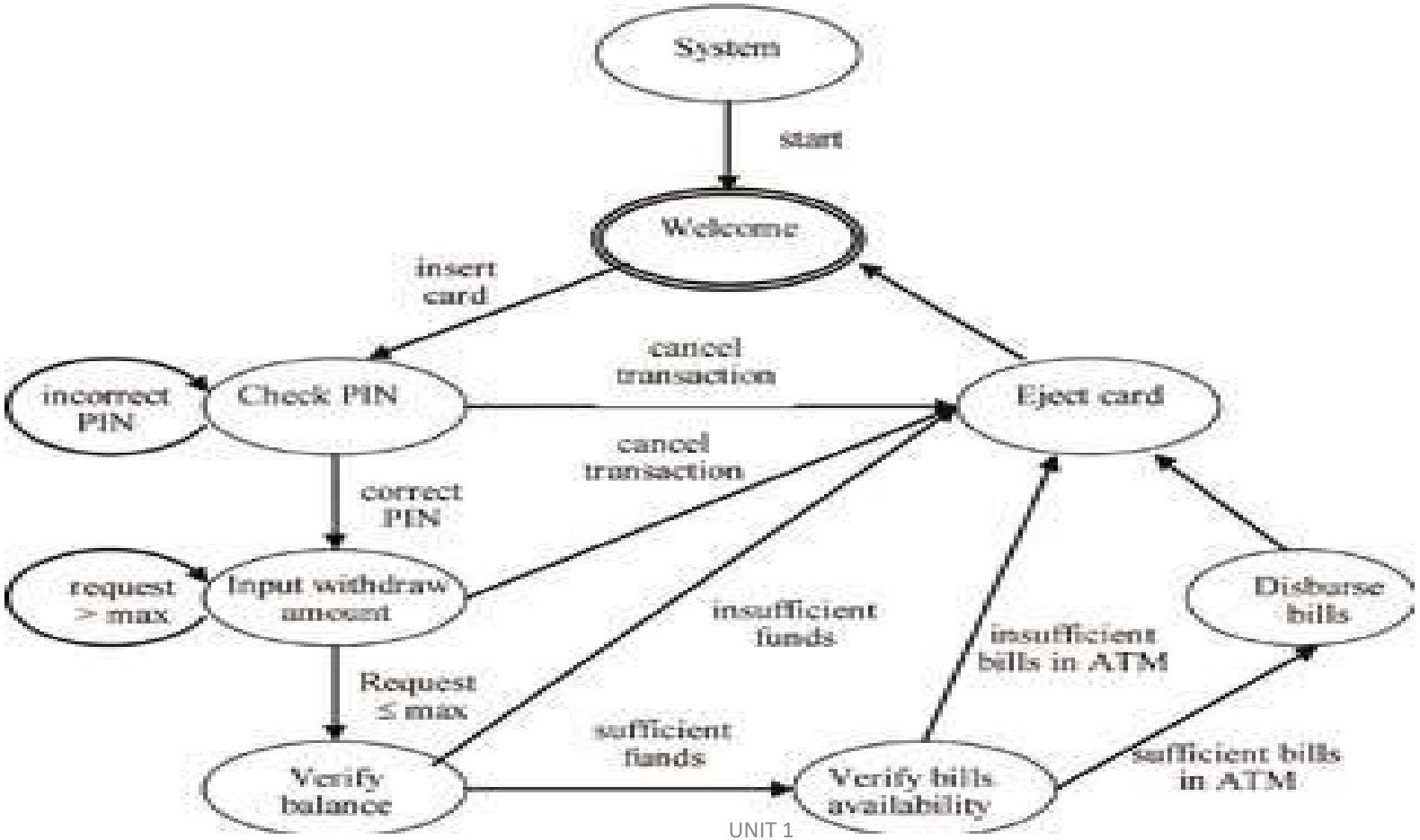
1930s	• Alan Turing studies Turing machines • Decidability • Halting problem
1940-1950s	• "Finite automata" machines studied • Noam Chomsky proposes the "Chomsky Hierarchy" for formal languages
1969	Cook introduces "intractable" problems or "NP-Hard" problems
1970-	Modern computer science: compilers, computational & complexity theory evolve

Alan Turing (1912-1954)



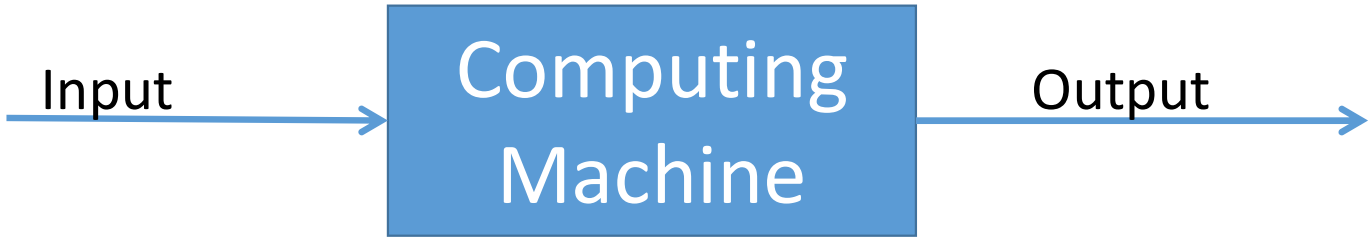
Father of Modern Computer Science
English mathematician
Studied abstract machines called ***Turing machines*** even before computers existed

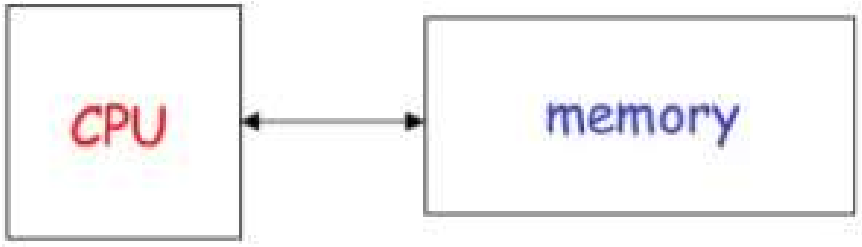
Computation Model for ATM



Computing Machine

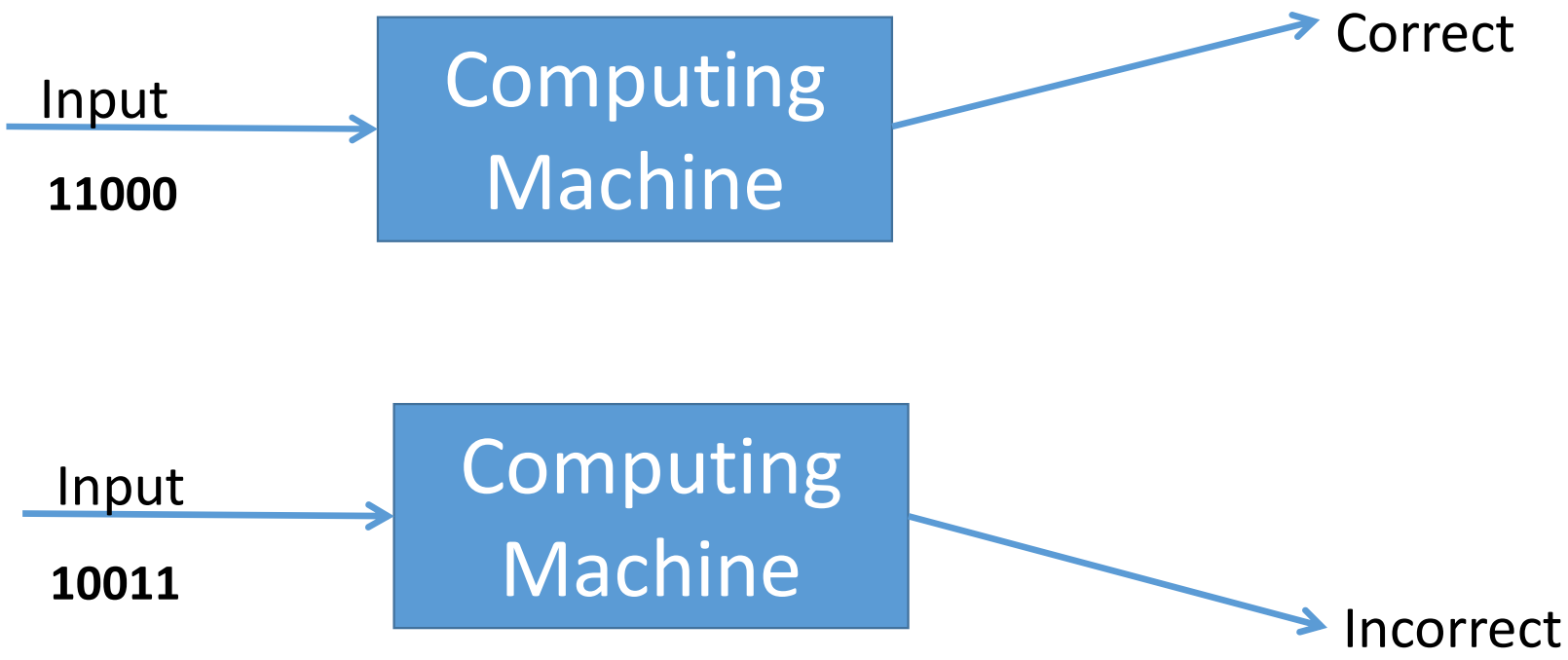
- General Model





Computing Machine

Example : Computing machine to find the input end with 0 or not:



Solvable and Unsolvable Problem

Problems can be solved by computing machine.

Problems can’t be solved by computing machine.

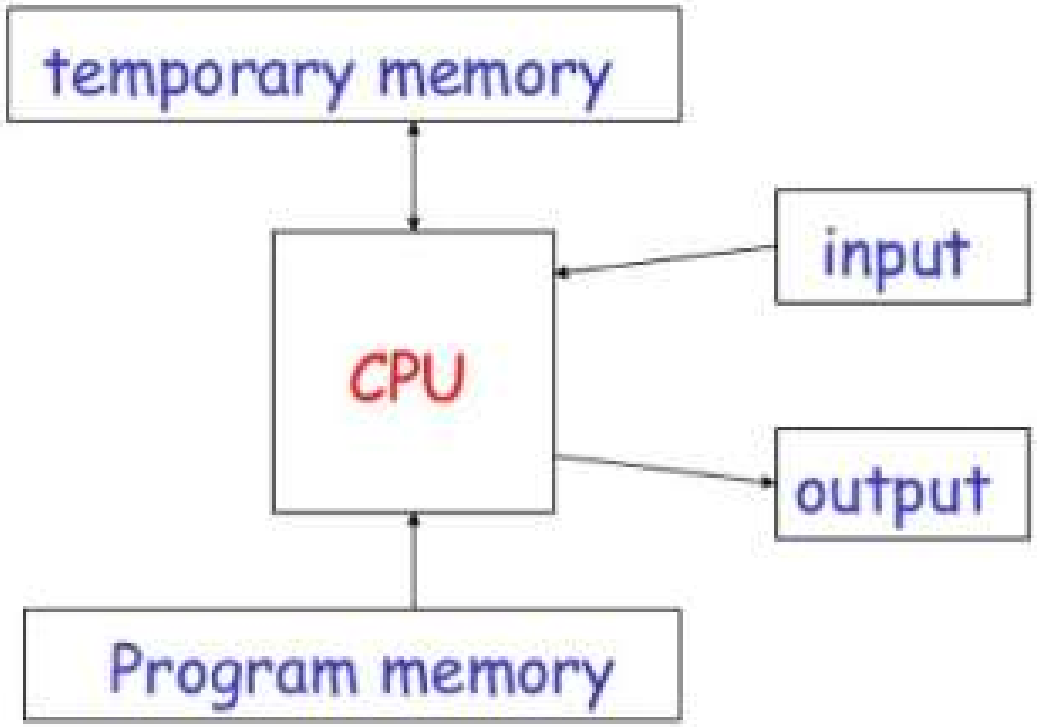


Fixing the Plug



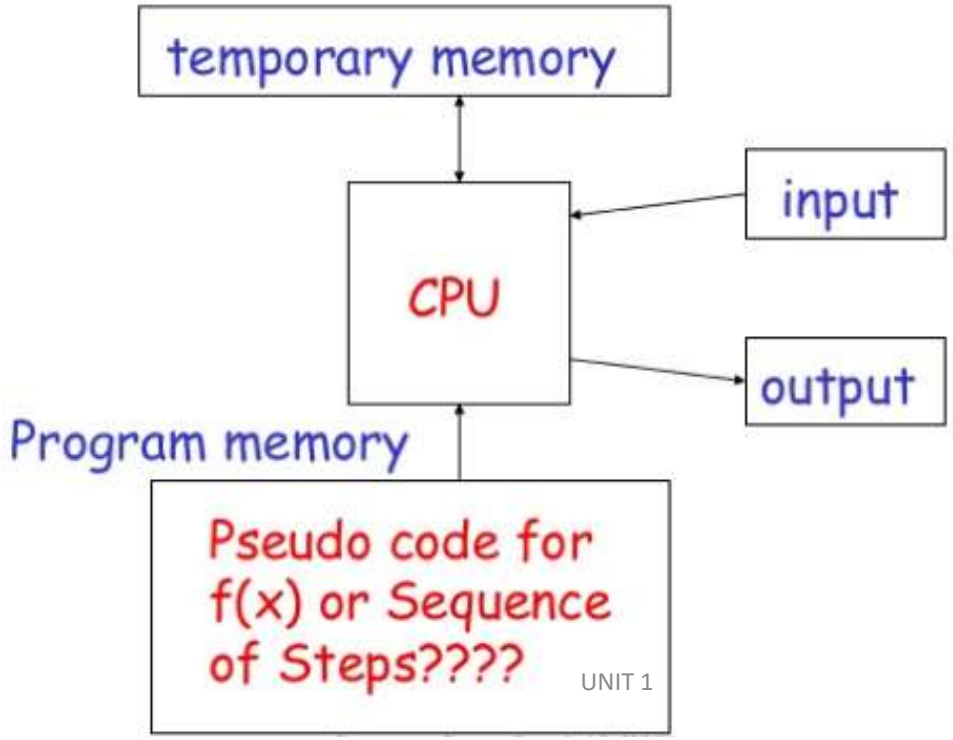
Count the Stars at the Sky

Model of Computation



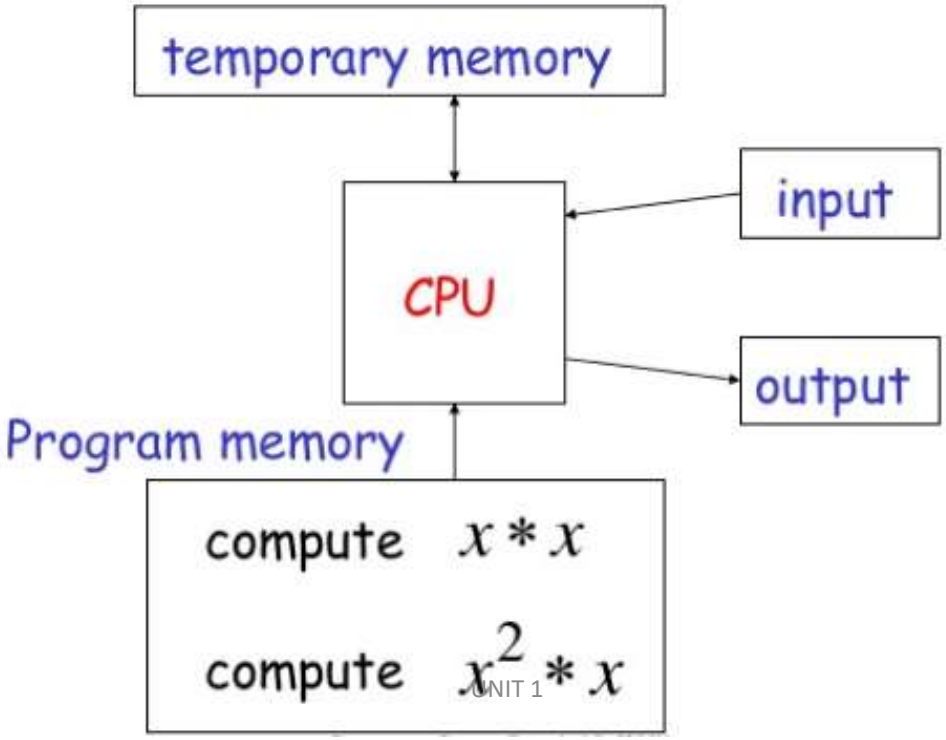
Model of Computation

Example: $f(x) = x^3$



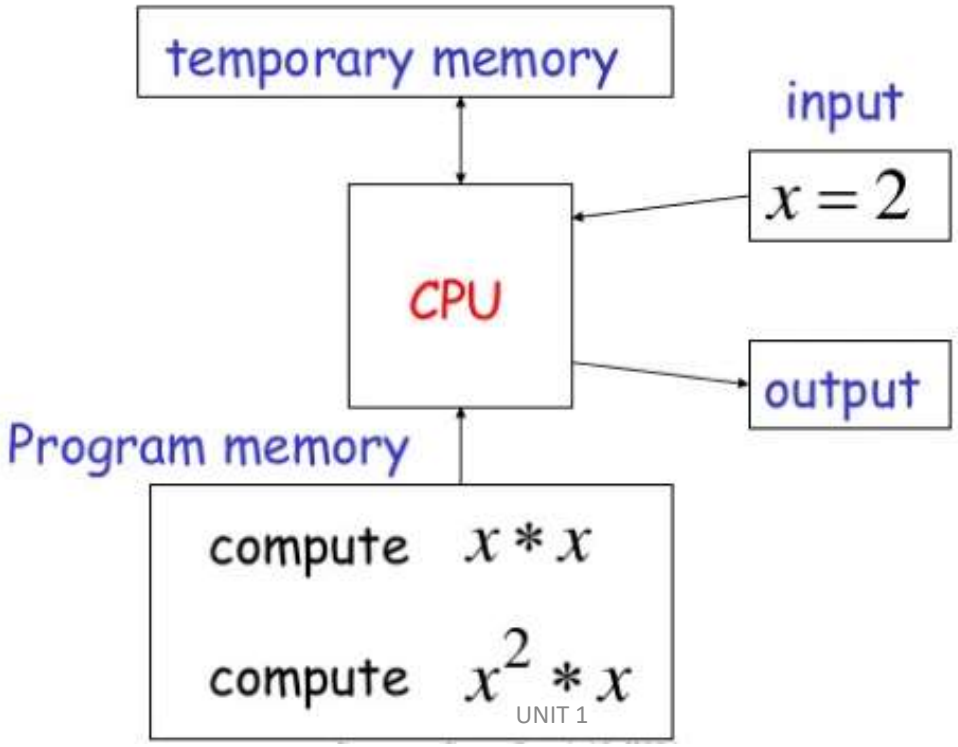
Model of Computation

Example: $f(x) = x^3$

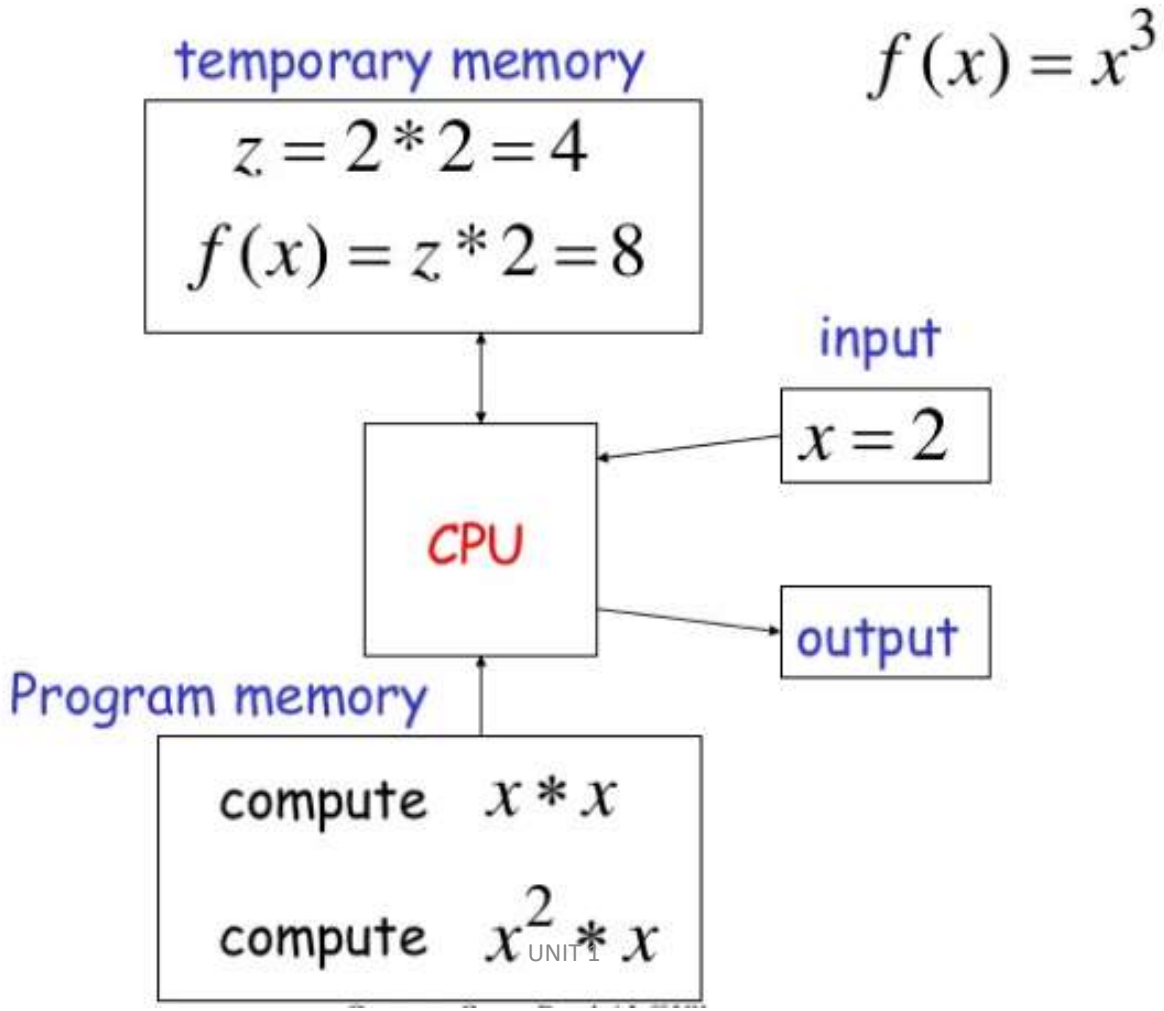


Model of Computation

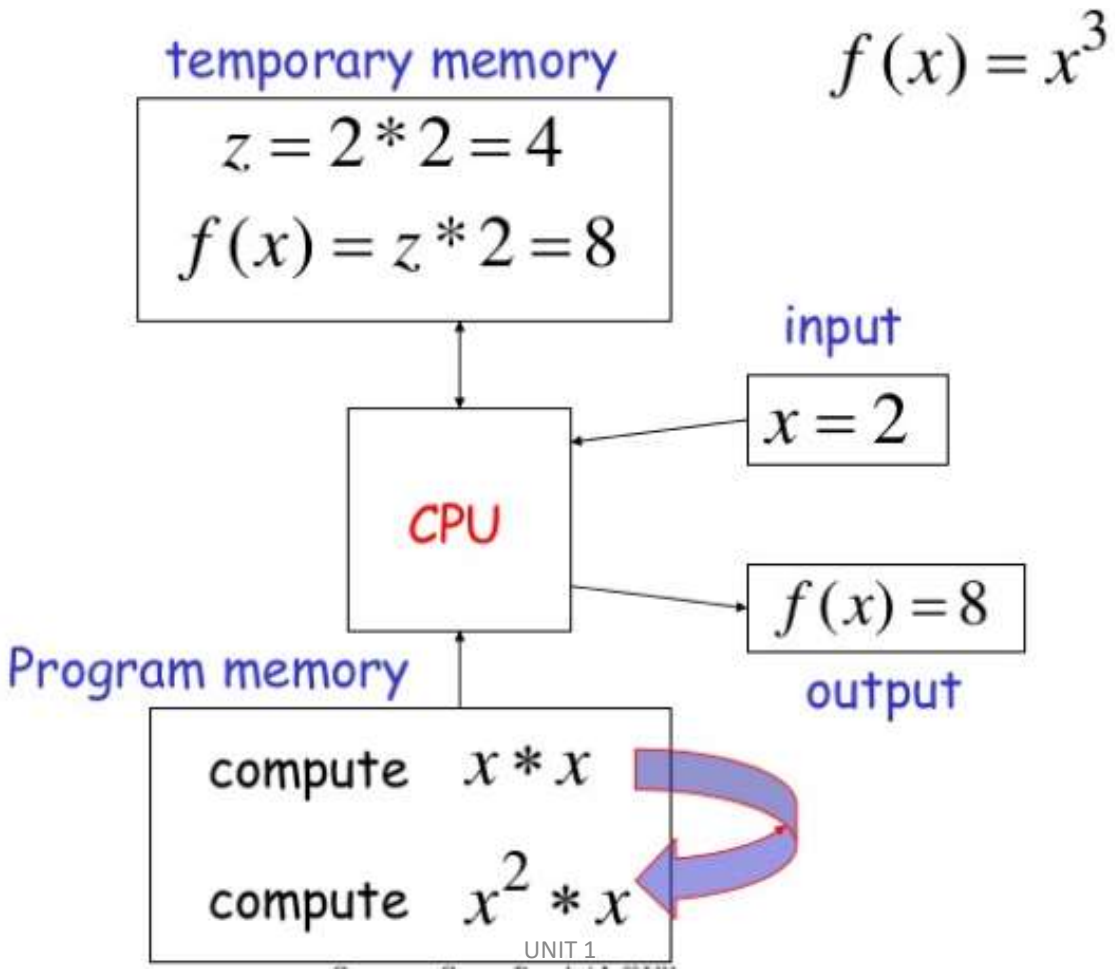
$$f(x) = x^3$$



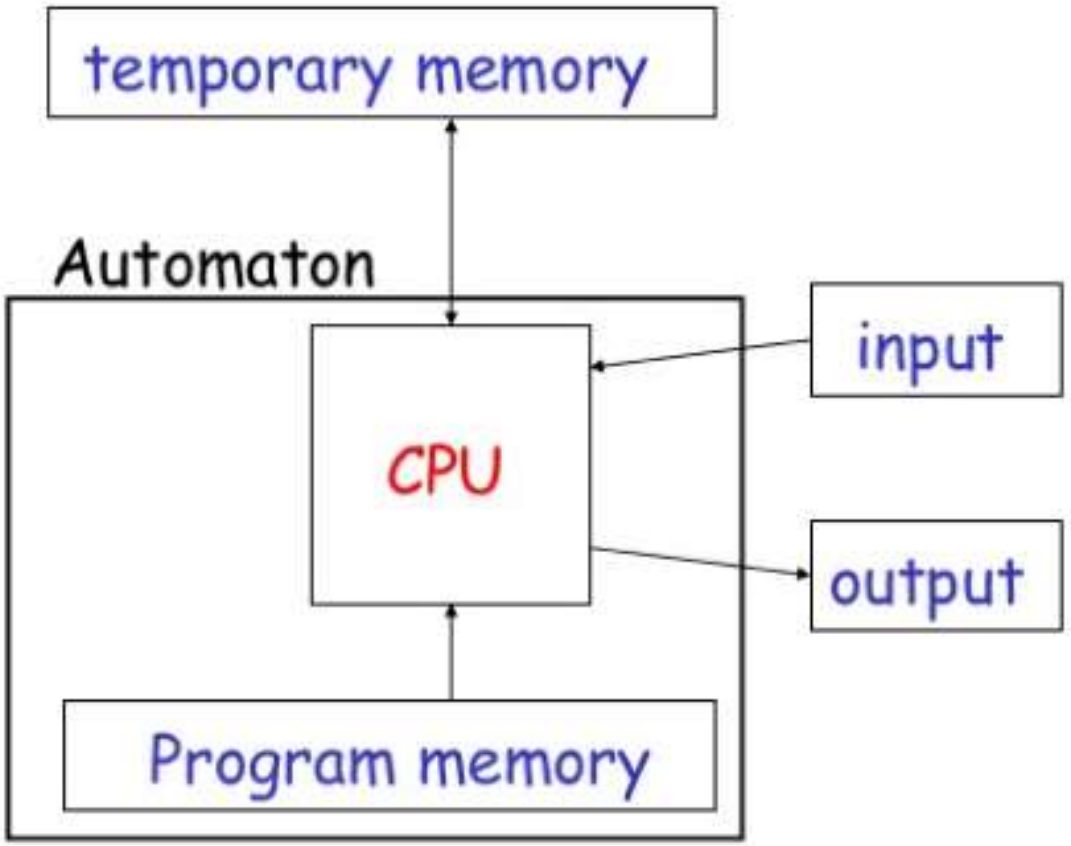
Model of Computation



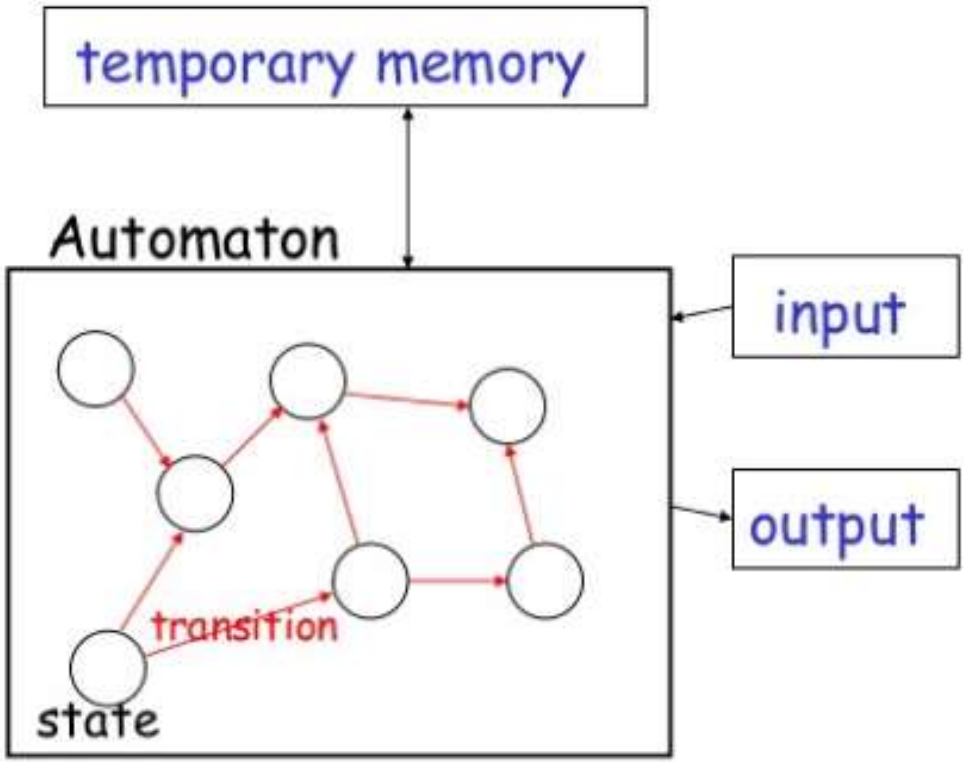
Model of Computation



Automaton


temporary memory
Automaton
input
CPU
output
Program memory

Automaton



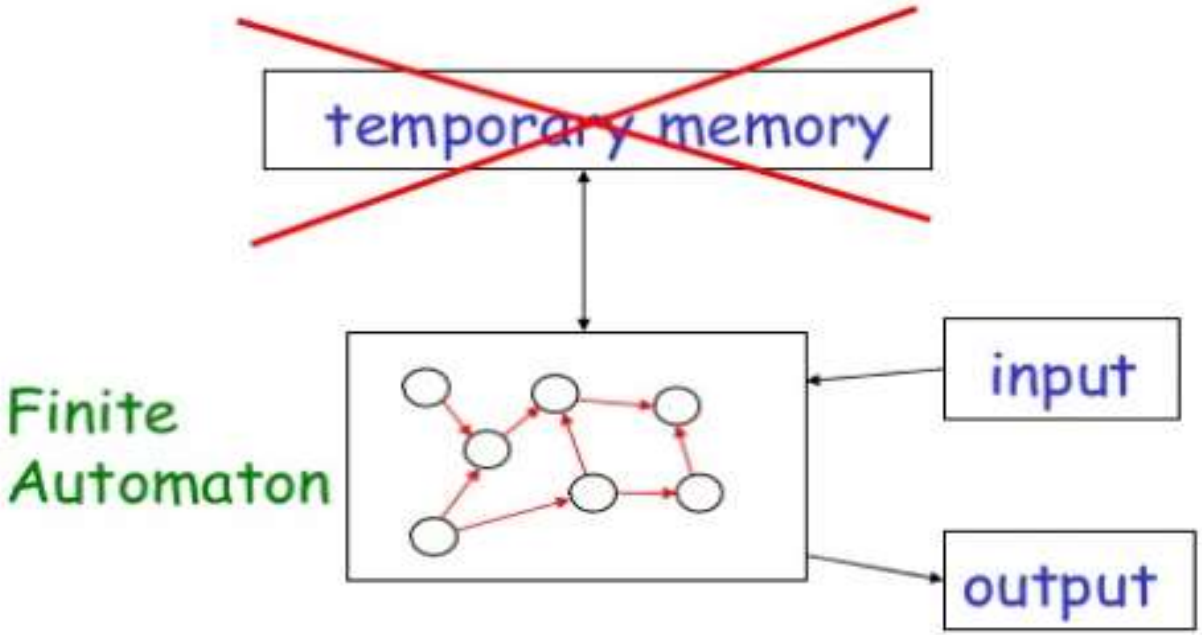
CPU + ProgramMem = States + Transitions

Different kinds of Automata

Automata are distinguised by the temporary memory

- Finite Auomata - No temporary memory
- Push Down Automata - Stack
- Turing Machine - Random Access Memory

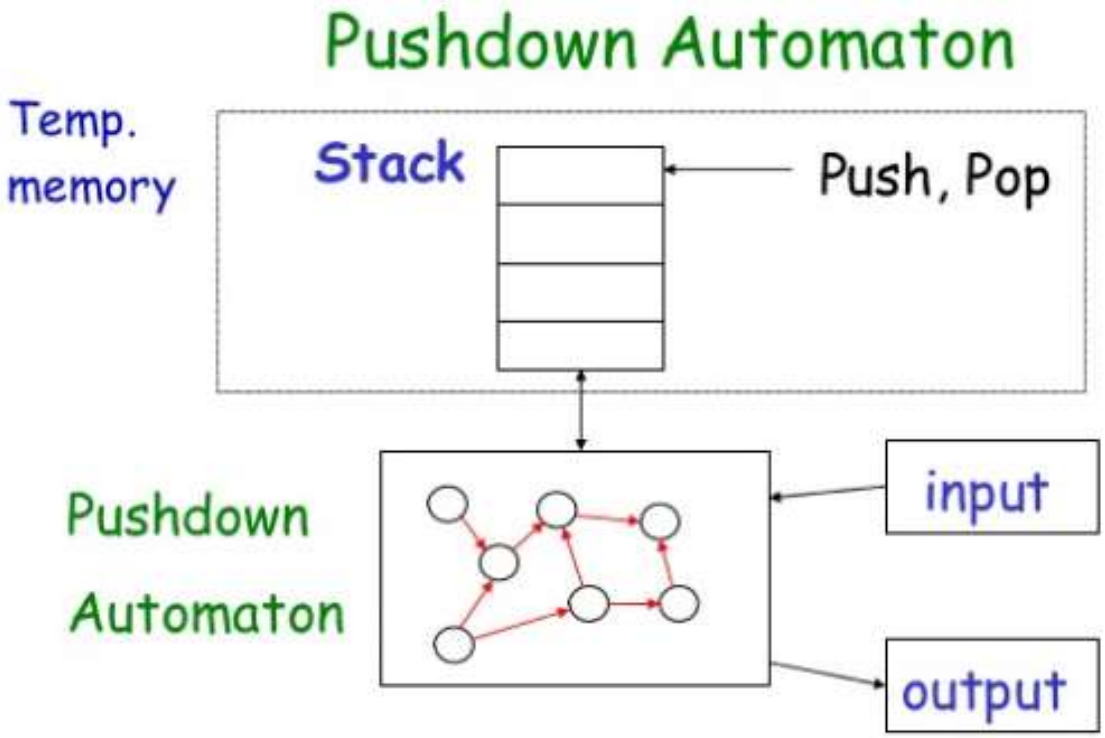
Finite Automaton



Example: Elevators, Vending Machines,

Lexical Analyzers

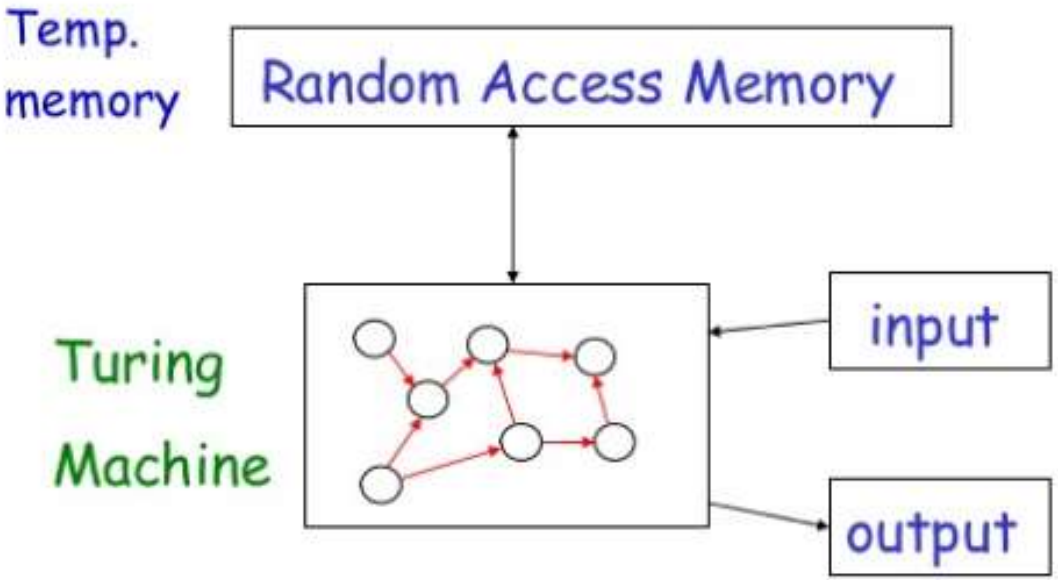
(small computing power)



Example: Parsers for Programming Languages

(medium computing power)

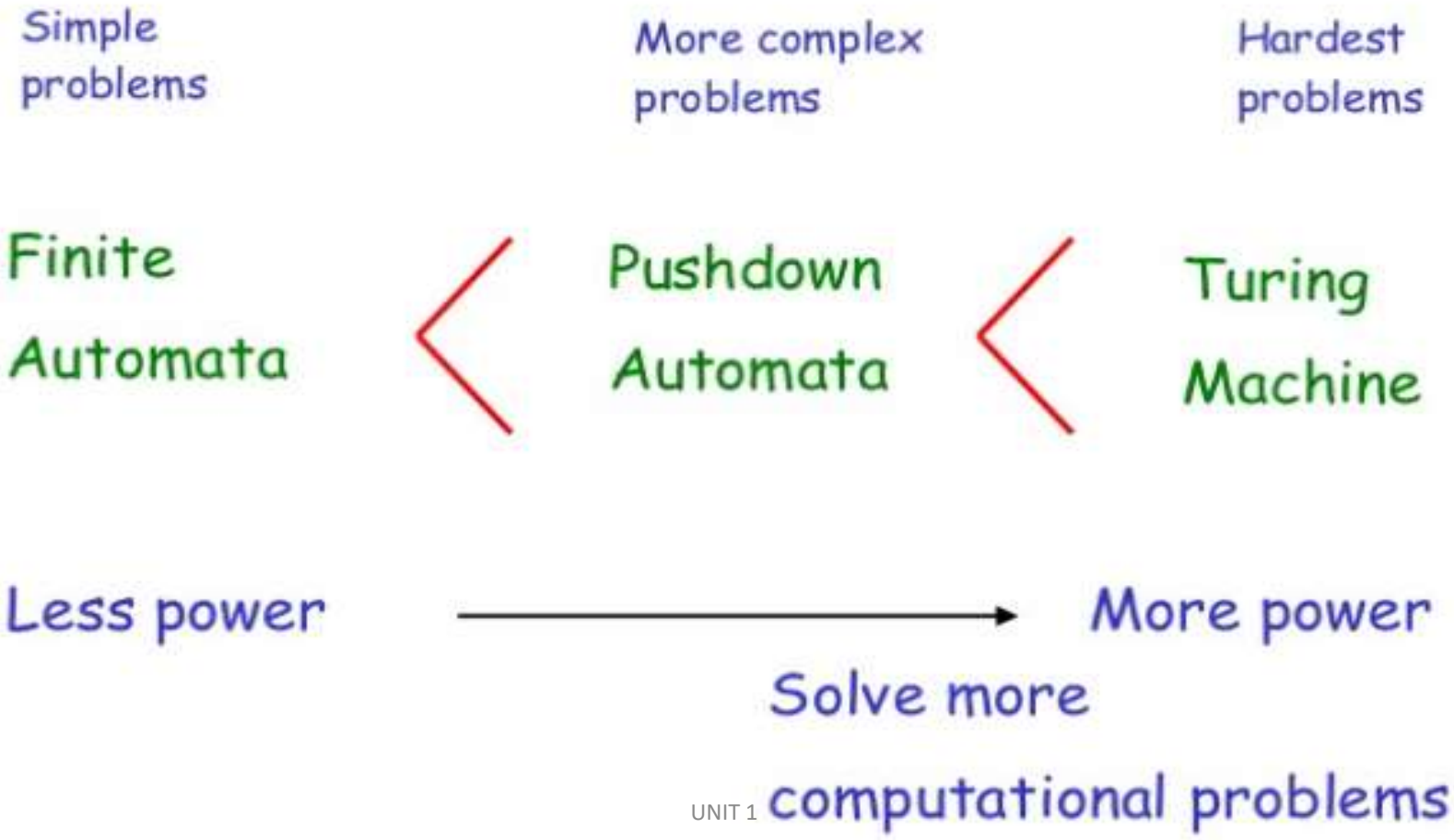
Turing Machine



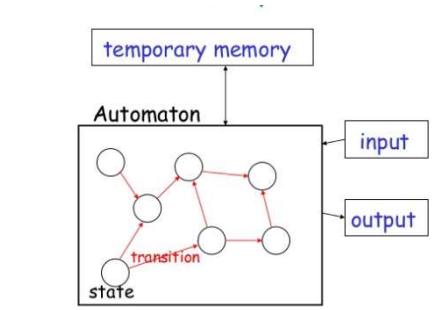
Examples: Any Algorithm

(highest known computing power)

Power of Automata



Automata Theory



CPU + ProgramMem = States + Transitions

- Automata theory (also known as Theory of Computation) is a theoretical branch of Computer Science and Mathematics, which mainly deals with the logic of computation with respect to simple machines, referred to as automata.
- Automata is the kind of machine which takes some string as input and this input goes through a finite number of states and may enter in the final state.

Basic Concepts of Automata Theory

Symbols

Symbol is the smallest building block, which can be any letter, numbers or any picture.

a,b,c.....z

0,1,2.....9

A,B.....Z

#,$,@.....&

Ж☺.....

Alphabet :(**Σ**)

Alphabets are non empty set of symbols, which are always ***finite***.

Σ ={0,1} Binary Alphabet

Σ ={a,b} Lower case letter set

Σ ={0,1,2} Digits

Σ ={$,&,@}

Σ ={Ж, ☺ }

String:(**W**)

String is a ***finite*** sequence of symbols from some alphabet.

Example:

Σ ={0,1,2} w= 0,1, 2, 01, 02, 110, 210, 111,.....

01101 is a string from Σ ={0,1}

aa, bb, ab, ba are strings from Σ ={a,b}

- *The Empty string is ε (or "epsilon") the string with zero occurences of symbols.*
- *The length of a string is the number of positions for symbols in the string. The length of astring w is written as* /w/
- **/00111/ = 5**
- **$/\varepsilon/ = 0$**

Power of an Alphabet Σ^k

If Σ is an alphabet , the power of an alphabet Σ^k is the set of all strings of a certain length k from that alphabet by using an exponential notation.

Example: Σ:{0,1}

Σ^0=all set strings with length 0 $\Sigma^0=\{\varepsilon\}$

Σ^1=all set strings with length 1 $\Sigma^1=\{0,1\}$

Σ^2=all set strings with length 2 $\Sigma^2=\{01,10,00,11\}$

Σ^3=all set strings with length 2 $\Sigma^3=\{000,111,100,101,110,011,010,001\}$

.

Σ^n=all set strings with length n

Power of an Alphabet Σ^k

Example: Σ:{0,1,2}

Σ^0=all set strings with length 0 $\Sigma^0=\{\varepsilon\}$

Σ^1=all set strings with length 1 $\Sigma^1=\{0,1,2\}$

Σ^2=all set strings with length 2 $\Sigma^2=\{00,01,02,10,11,12,20,21,22\}$

.

.

Σ^n=all set strings with length n

Power of an Alphabet Σ^k

Σ^* = Kleene Closure , denotes the set of all strings over the alphabet,

$$\sum = \{a, b\} \qquad \sum^* = \{\in, a, b, aa, bb, ab, ba, aaa, aab, ...\}$$

Σ^+ = Positive Closure, denotes the set of all strings over the alphabet, except null string.

Cardinality: Number of strings in a set

$$\Sigma^n = 2^n$$

$\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup ..... \Sigma^n$ = {ε} U {0,1}.....

$\Sigma^+ = \Sigma^1 \cup \Sigma^2 \cup ..... \Sigma^n$ = {0,1} U {00,11,01,10}.....

$\Sigma^+ = \Sigma^* - \{\varepsilon\}$

Concatenation of Strings

- Let x and y be strings. Then xy denotes the concatenation of x and y that is, the string formed by making a copy of x and following it by a copy of y.

Example:

x=110 y= 101

then xy = 110101

yx= 101110

Language: (L)

Any set of strings over an alphabet Σ is called a language.
Language is a *set of strings*, chosen from some Σ*.

(or)

Language is a subset of Σ*.

Σ:{a, b}
L1={Set of all strings of length 2}
L1={ab, aa, bb, ba}
L2={Set of all strings end with 'a'}
L2={a, ba, bba, aba, aa, aaa.....}

Language (L)

- **Example:**
- **Construct a language contain a sub string of “ab” over Σ:{a, b}**

  L={ab,aab,bab,bbab,aba…..}
- **Construct a language starts with “b” over Σ:{a, b}**

  L={b,ba,bab,bbab,baa…..}
- **Construct a language contains odd number of “b” over Σ:{a, b}**

  L={ab,aab,ab,bbab,aba…..}
- **Construct a language contains even number of “a” over Σ:{a, b}**

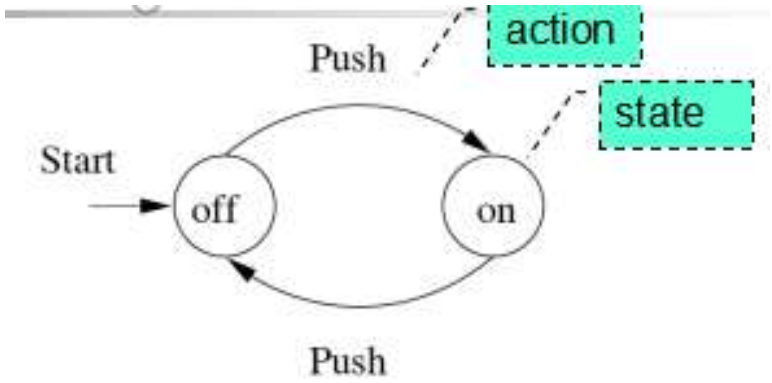
  L={aa,aab,aba,baab,abba….}

Finite Automata

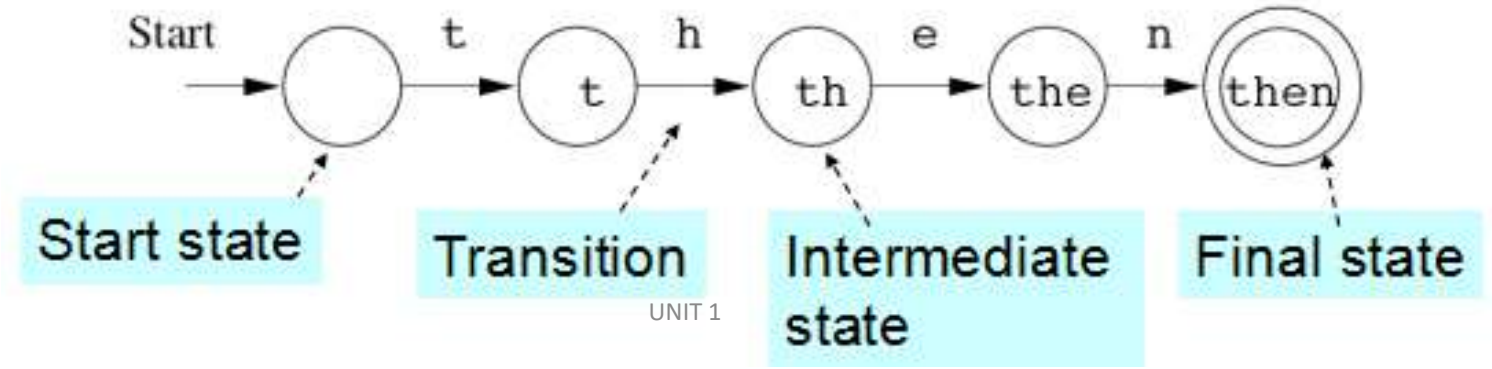
- Finite Automata is a mathematical model of system with certain input and it is processed through various intermediate states and finally gives certain output.
- This automaton consists of states and transitions. The State is represented by circles, and the Transitions are represented by arrows.
- An automaton with a finite number of states is called a Finite Automaton (FA) or Finite State Machine (FSM).

Finite Automata : Examples

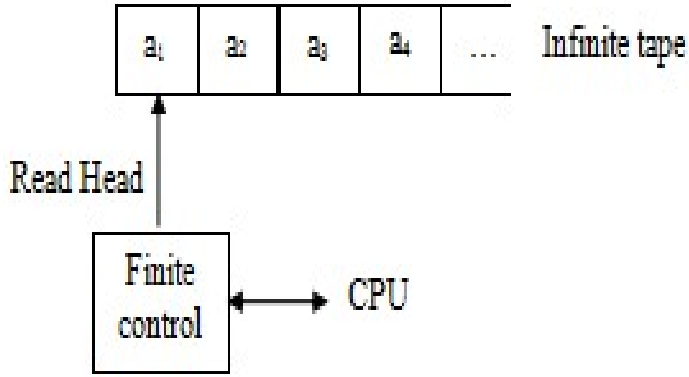
- On/Off switch



- Modeling recognition of the word "*then*"



Model of Finite Automata



- Input Tape

  It is divided into number of cells.

  Each cell holds one symbol.

- Read Head

  Reads one cell at a time and moves ahead.

- Finite Control

  Acts like a CPU.

  Depending on current state and input symbol read from tape, it changes state.

Formal definition of a Finite Automata

- An Finite Automaton can be represented by a 5-tuple

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

  where

  - Q is a finite set of states.
  - $\sum$ is a finite set of symbols, called the alphabet of the automaton.
  - δ is the transition function.
  - q_0 is the initial state from where any input is processed ($q_0 \in Q$).
  - F is a set of final state/states of Q ($F \subseteq Q$).

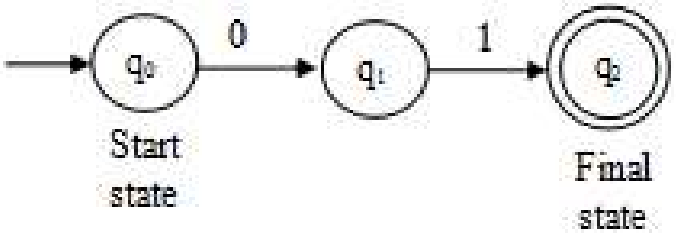
Representation of Finite Automata

1.Transition Diagram

2.Transition Table

Transition Diagram

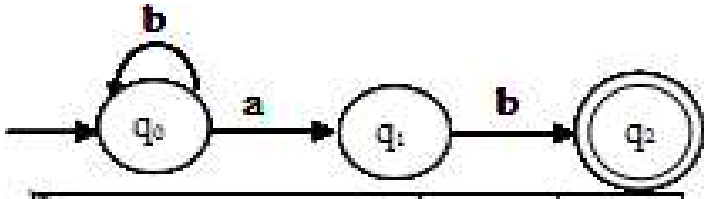
- It can be directed graph with vertices of a graph corresponding to states and edges indicates transition from one state to another
  - Start state indicated by circle.
  - State transition indicated by arrow.
  - Final state indicated by double circle.



Representation of Finite Automata

Transition Table

- It is a tabular listing of the transition function which by implication tells us, set of states and input alphabet.



State ↓ / Input →	a	b
→ q_0	q_1	q_0
q_1	ϕ	q_2
*q_2	ϕ	ϕ

$\delta(q_0, a) = q_1$
$\delta(q_0, b) = q_0$
$\delta(q_1, b) = q_2$

Applications of Finite Automata

- Software for designing and checking the behavior of digital circuits

- Lexical analyzer of a typical compiler
- Software for scanning large bodies of text (e.g., web pages) for pattern finding
- Software for verifying systems of all types that have a finite number of states (e.g., stock market transaction, communication/network protocol)

Types of Finite Automata

1. **Deterministic Finite Automata (DFA)**
2. **Non Deterministic Finite Automata (NFA)**
   - **NFA with epsilon**
   - **NFA without epsilon**

Deterministic Finite Automata (DFA)

- The term deterministic refers to the fact that **on each input there is one and only one state to which the automaton can transition from its current state.**
- Deterministic refers to the uniqueness of the computation.
- In DFA, there is only one path for specific input from the current state to the next state.
- DFA does not accept the null move, i.e., the DFA cannot change state without any input character.
- It is used in Lexical Analysis in Compiler.

Example

- In the following diagram, we can see that
- From state **q0 for input a**, there is only one path which is going to q1.
- Similarly, from **q0, there is only one path for input b** going to q2.

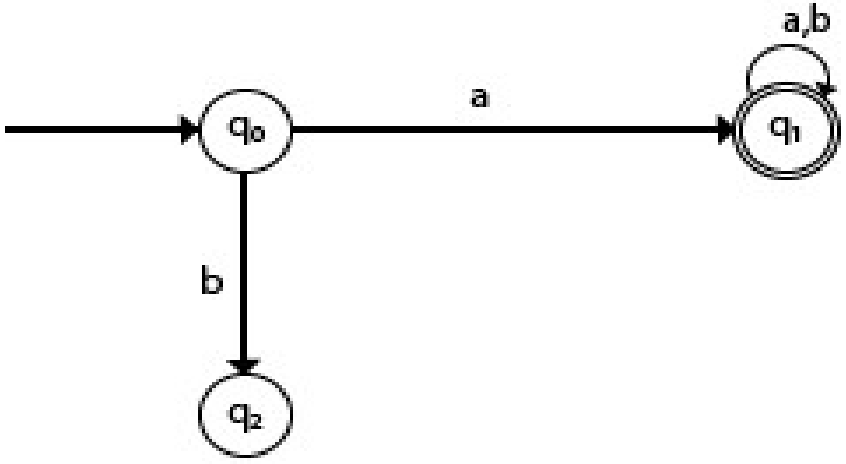


Fig:- DFA

Definition of DFA

A Deterministic Finite Automata (DFA) consists of:

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

Q is a finite set of states.

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

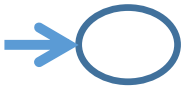
δ is the transition function where $\delta: Q \times \sum \rightarrow Q$

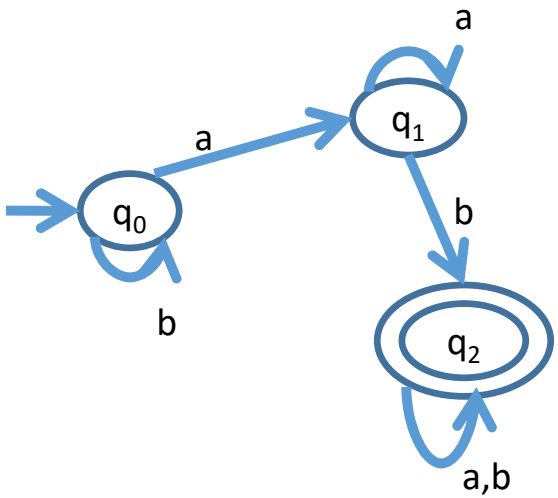
$\mathbf{q_0}$ is the initial state ($q_0 \in Q$).

F is a set of final state/states of Q ($F \subseteq Q$).

Representation DFA

Graphical Representation: State Transition Diagram

- ✓States : 
- ✓Transition: 
- ✓Initial State: 
- ✓Final State: 
- ✓Input Symbol : a, b, 0,1....



Representation DFA

- **Tuple Representation : M=(Q, ∑, δ, q_0, F)**

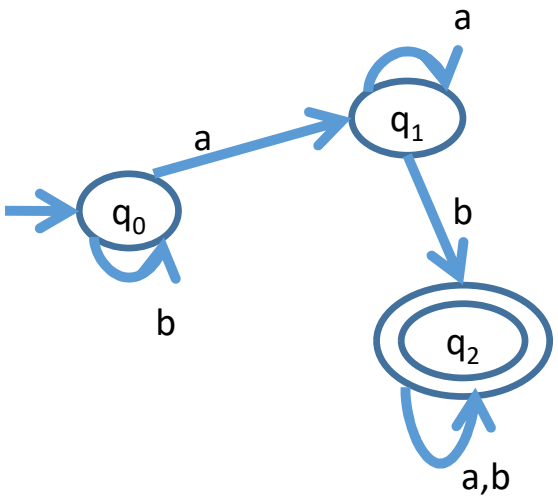
Q is a finite set of states.

$\{q_0, q_1, q_2\}$

∑ is a finite set of symbols called the alphabet.

{a,b}

δ is the transition function where δ: Q × ∑ → Q



$\delta(q_0,a)=q_1$ $\delta(q_0,b)=q_0$ $\delta(q_1,a)=q_1$

$\delta(q_1,b)=q_2$ $\delta(q_2,a)=q_2$ $\delta(q_2,b)=q_2$

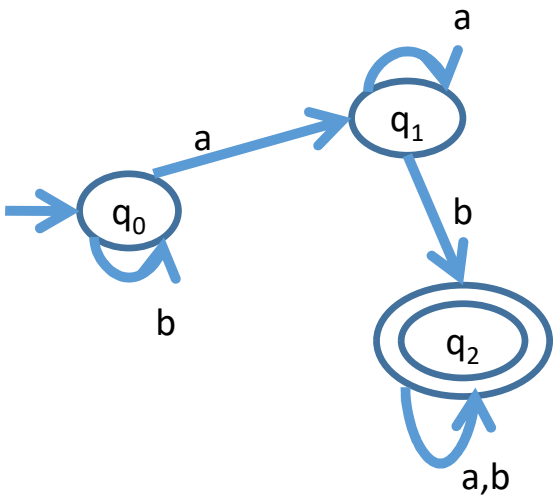
Representation DFA

q_0 is the initial state ($q_0 \in Q$).

$q_0=q_0$

F is a set of final state/states of Q ($F \subseteq Q$).

$F=\{q_2\}$



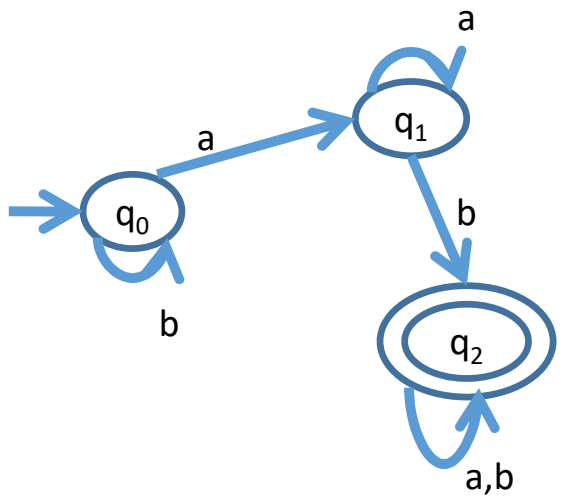
$M=(Q, \sum, \delta, q_0, F)$

$M=(\{q_0,q_1,q_2\}, \{a , b\}, \{\delta(q_0,a)=q_1, \delta(q_0,b)=q_0, \delta(q_1,a)=q_1, \delta(q_1,b)=q_2, \delta(q_2,a)=q_2, \delta(q_2,b)=q_2\}, q_0, \{q_2\})$

Representation DFA

Transition Table

	0	1
→q_0	**q_1**	**q_0**
q_1	q_1	q_2
*q_2	q_2	q_2

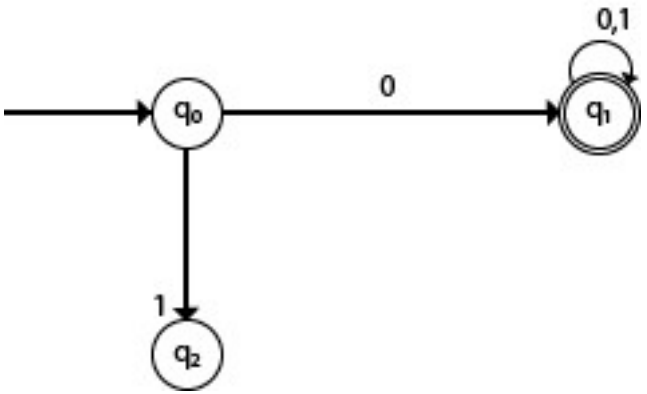


$$M=(\{q_0,q_1,q_2\}, \{a , b\} , \delta, q_0 , \{q_2\})$$

Example Problem 1

- **DFA with ∑ = {0, 1} accepts all strings starting with 0.**

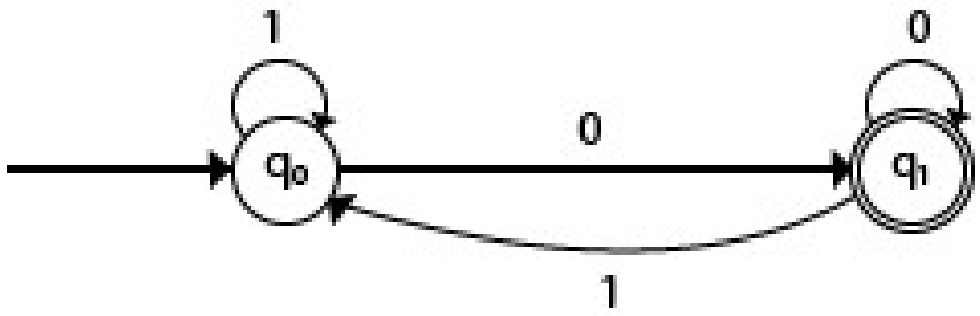
- L={0,01,00,000,011,.....}



Example Problem 2

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

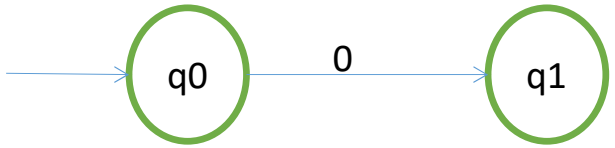
- L={0,00,10,100,10010,.....}



Construction of DFA

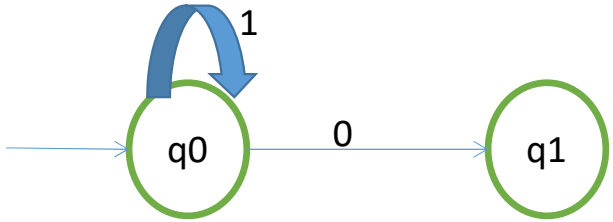
L={0,00,10,100,10010,.....}

step 1



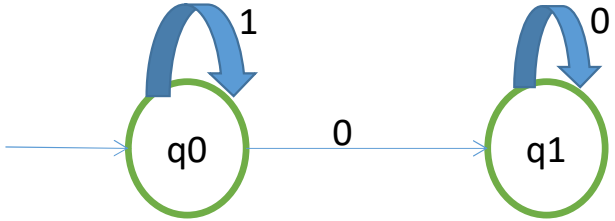
0

step 2



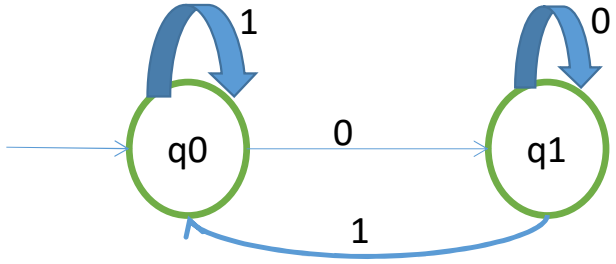
For q0 11110

step 3



For q1 0000

step 4



For q1 010

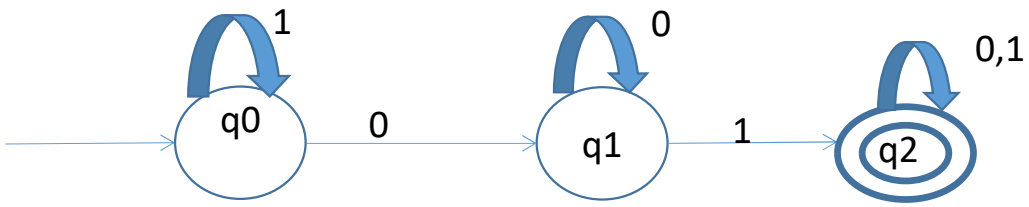
Example Problem 3

- Design a DFA for the language accepting all strings with a substring 01 over input alphabets ∑ = {0, 1}

soln:

L={ 01,001,101,010,011,.....}

Trasition Diagram



Trasition Table

State/Input	0	1
→q_0	q_1	q_0
q_1	q_1	q_2
*q_2	q_2	q_2

- **Extended Transition Function**

$$\hat{\delta}(q, w) = \delta\left(\hat{\delta}(q, x), a\right)$$

Take a String 101 from given language

δ(q0, 101) = δ(δ'(q0,1),01)

=δ(q0,01)

δ(q0,01) =δ(δ'(q0,0),1)

=δ(q1,1)

= q2

string 101 is accepted

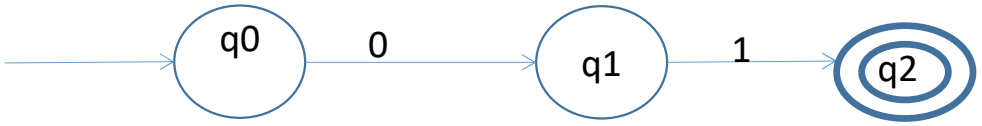
where

M=($\{q_0,q_1,q_2\}$, $\{0 , 1\}$, **δ**, q_0 , $\{q_2\}$)

Construction of DFA

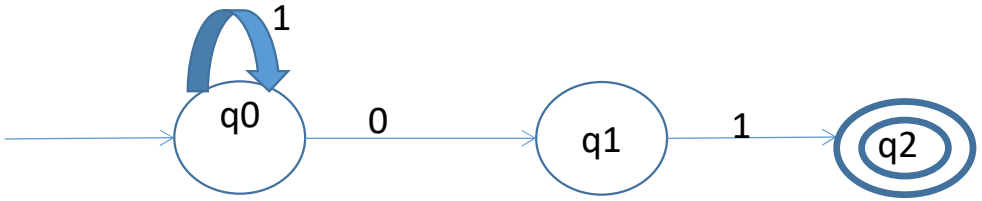
L={ 01,101,001,0101.....}

step 1



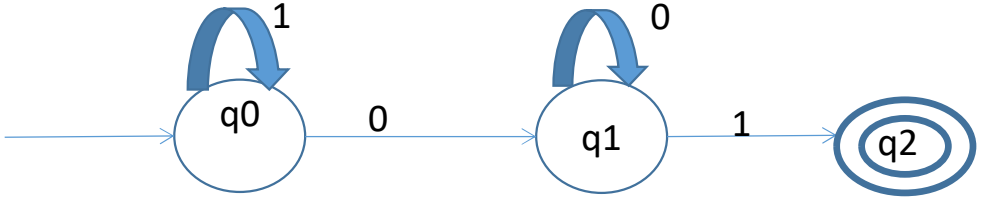
01

step 2



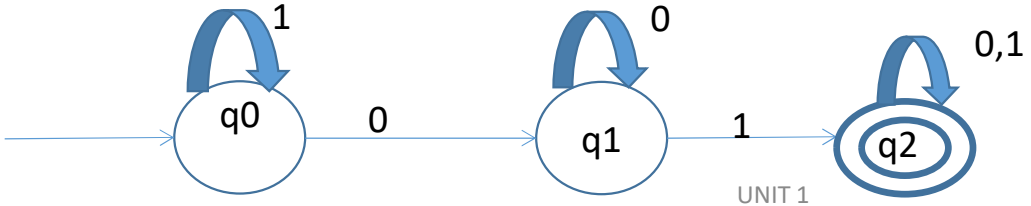
For q0 11101

step 3



for q1 00001

step 4

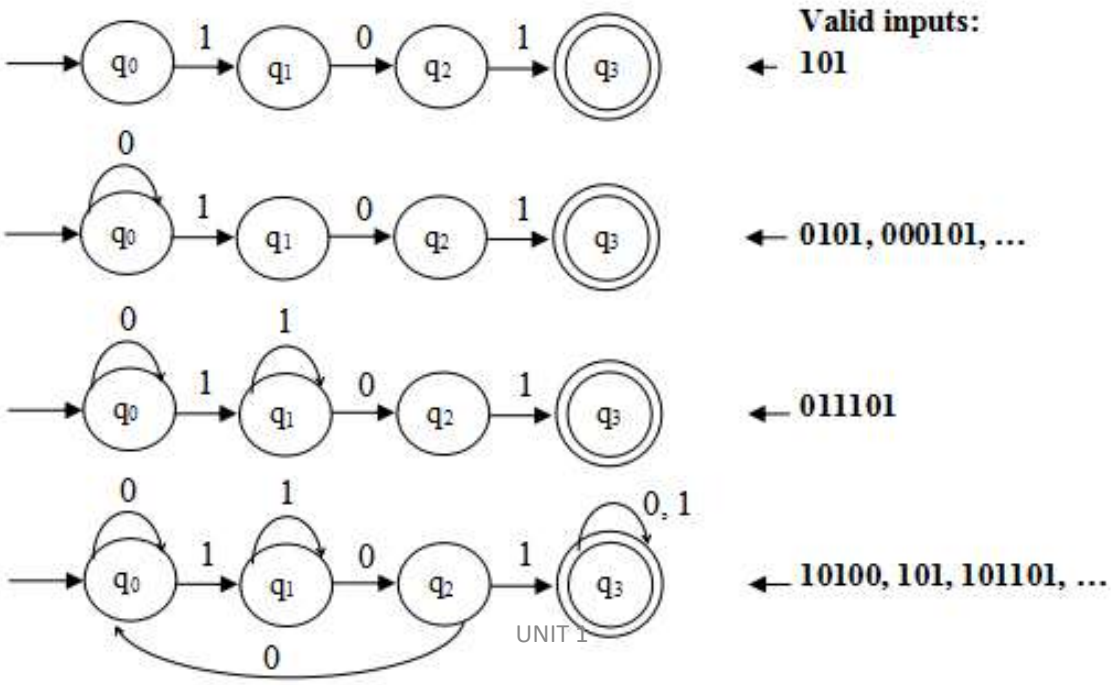


for q2 010011

Example Problem 4

- **Design a DFA that accepts all strings containing substring '101'.**

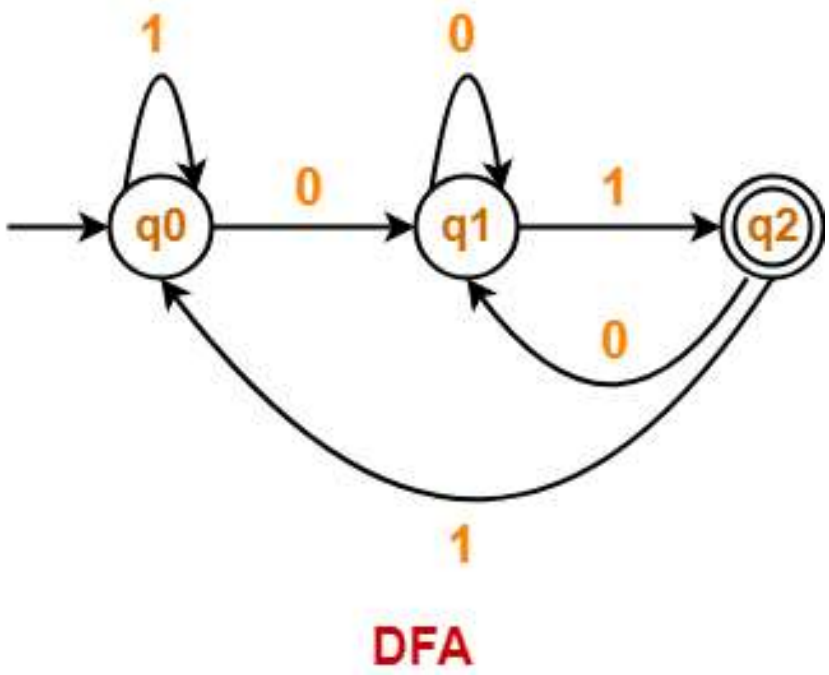
L={101, 0101,0001101,10100,10111.....}



Example Problem 5

- Draw a DFA for the language accepting strings ending with ’01’ over input alphabets ∑ = {0, 1}

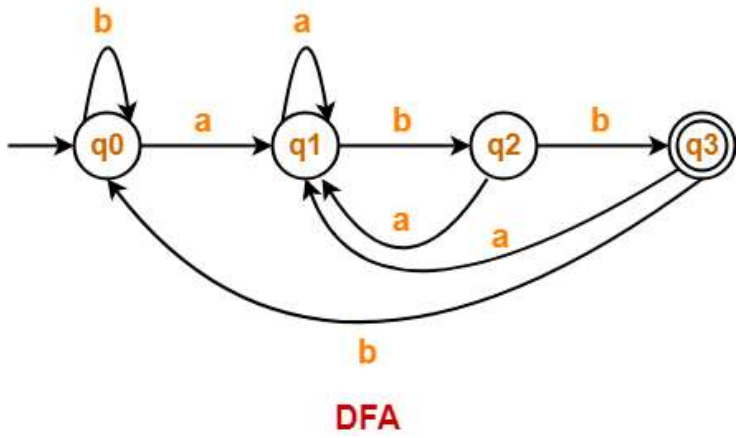
- L={01,101,001,0101.....}



DFA

Example Problem 6

- Draw a DFA for the language accepting strings ending with ‘abb’ over input alphabets ∑ = {a, b}



DFA

- L={abb,aabb,babb,ababb.....}

Home Work

- **Problem-7:**

- Draw a DFA for the language accepting strings ending with ‘abba’ over input alphabets ∑ = {a, b}

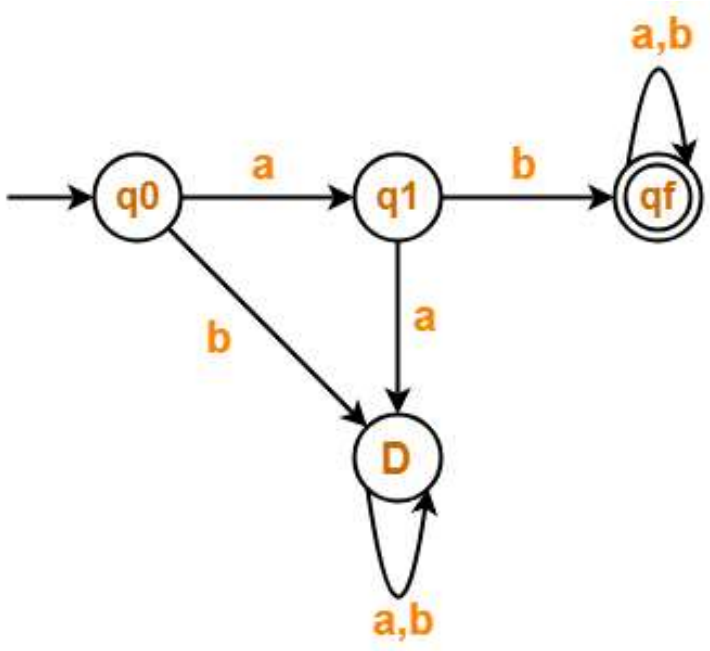
- **Problem-8:**

- Draw a DFA for the language accepting strings ending with ‘0011’ over input alphabets ∑ = {0, 1}

Example Problem 9

- Draw a DFA for the language accepting strings starting with ‘ab’ over input alphabets ∑ = {a, b}

- L={ab,aba,abab.....}



DFA

Home Work

- **Problem-10:**
- Draw a DFA for the language accepting strings starting with ‘a’ over input alphabets ∑ = {a, b}
- **Problem-11:**
- Draw a DFA for the language accepting strings starting with ‘101’ over input alphabets ∑ = {0, 1}
- **Problem-12:**
- Draw a DFA that accepts a language L over input alphabets ∑ = {0, 1} such that L is the set of all strings starting with ’00’.

Non-Deterministic finite automata (NFA)

- NFA stands for **non-deterministic finite automata**. It is easy to construct an NFA than DFA for a given regular language.
- The finite automata are called NFA when there exist many paths for specific input from the current state to the next state.
- Every NFA is not DFA, but each NFA can be translated into DFA.
- NFA is defined in the same way as DFA but with the following two exceptions, it contains multiple next states, and it contains ε transition.

Example

- In the following digaram, we can see that
- From state **q0 for input a**, there are two next states q1 and q2,
- Similarly, from **q0 for input b**, the next states are q0 and q1.
- Thus it is **not fixed or determined** that **with a particular input where to go next.**
- Hence this FA is called **non-deterministic finite automata**.

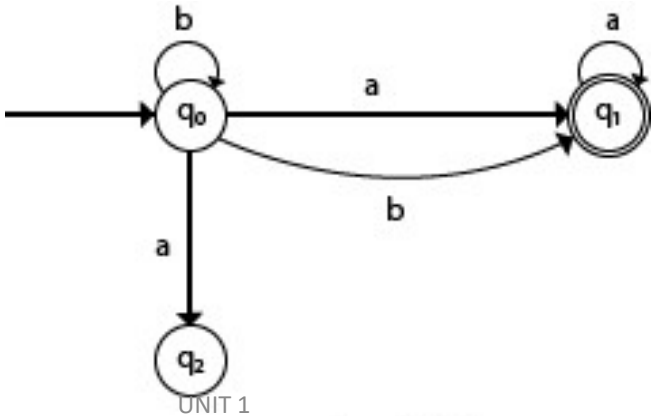


Fig:- NDFA

Definition of NFA

A Non Deterministic Finite Automata (NFA) consists of:

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

Q is a finite set of states.

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

δ is the transition function where $\delta: Q \times \sum \rightarrow 2^Q$

$\mathbf{q_0}$ is the initial state ($q_0 \in Q$).

F is a set of final state/states of Q ($F \subseteq Q$).

Example Problem 1

- NFA with ∑ = {0, 1} accepts all strings starting with 01.

- L={01,0100,0111,010011,.....}

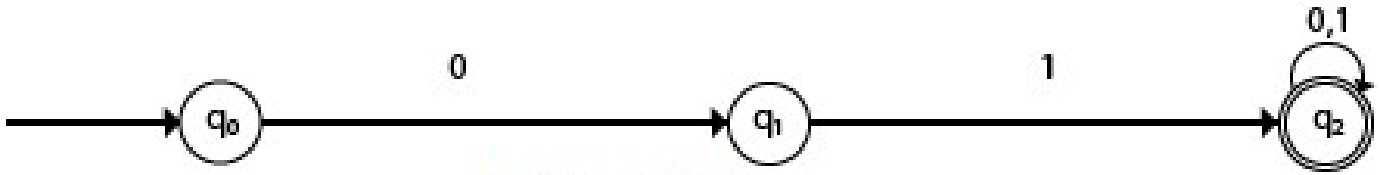
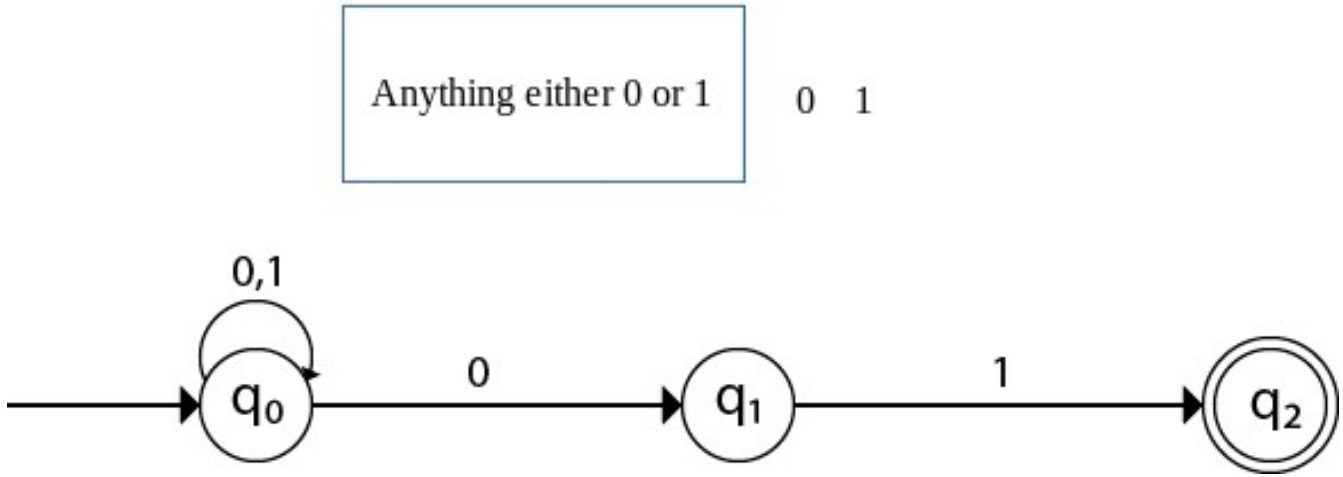


Fig: NFA

Example Problem 2

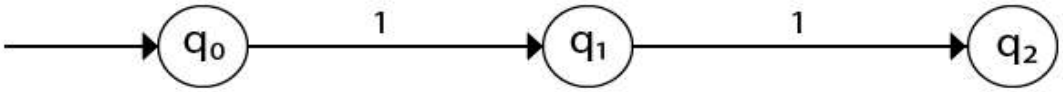
- Design an NFA with ∑ = {0, 1} accepts all string ending with 01.



Example Problem 3

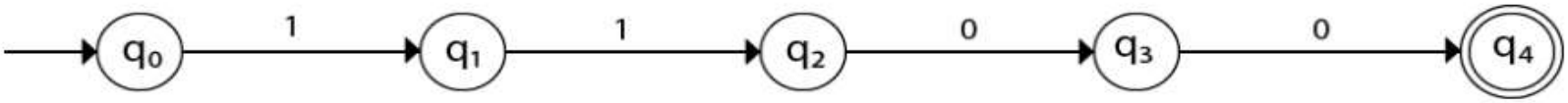
- Design an NFA with ∑ = {0, 1} in which double '1' is followed by double '0'. L={1100, 01100,11100,11000,11001,....}

The FA with double 1 is as follows:



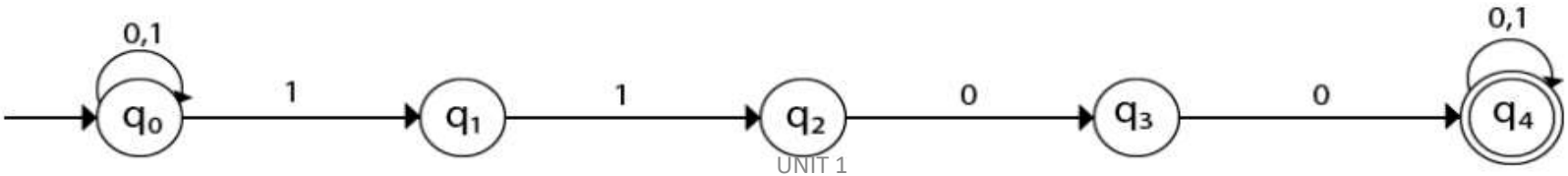
It should be immediately followed by double 0.

Then,



Now before double 1, there can be any string of 0 and 1. Similarly, after double 0, there can be any string of 0 and 1.

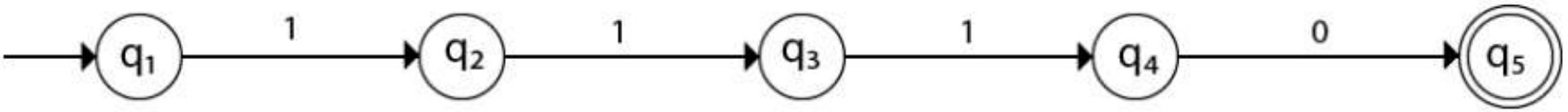
Hence the NFA becomes:



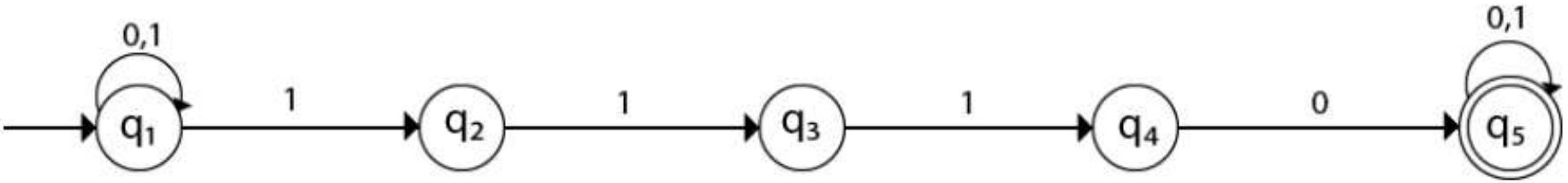
Example Problem 4

- Design an NFA in which all the string contain a substring 1110.
- L={1110,01110,11110,11101,11101,.....}

The language consists of all the string containing substring 1010. The partial transition diagram can be:



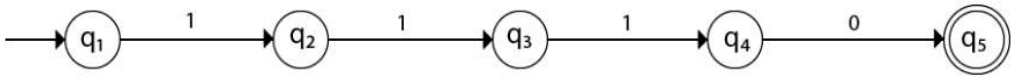
Now as 1010 could be the substring. Hence we will add the inputs 0's and 1's so that the substring 1010 of the language can be maintained. Hence the NFA becomes:



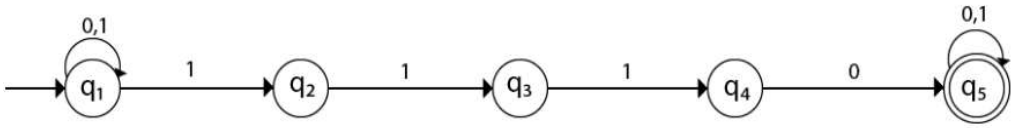
Trasition Table

State/Input	0	1
→q_1	q_1	q_1, q_2
q_2	Φ	q_3
q_3	Φ	q_4
q_4	q_5	Φ
*q_5	q_5	q_5

The language consists of all the string containing substring 1010. The partial transition diagram can be:

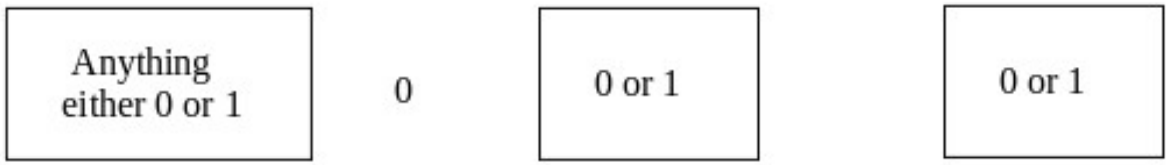


Now as 1010 could be the substring. Hence we will add the inputs 0's and 1's so that the substring 1010 of the language can be maintained. Hence the NFA becomes:

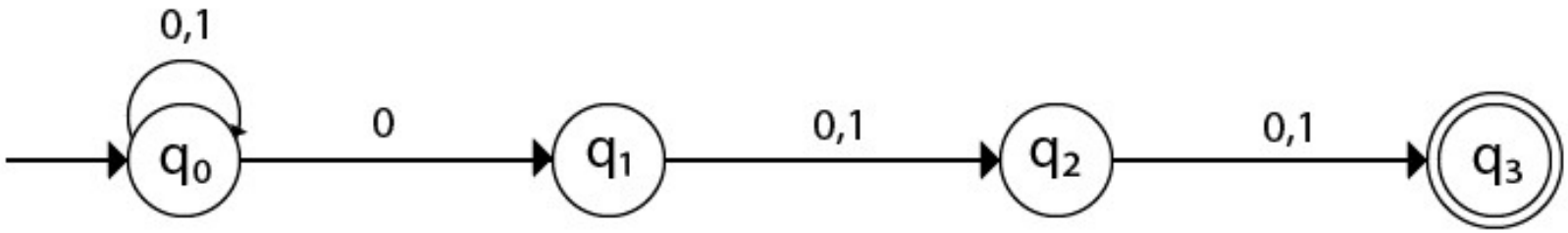


Example Problem 5

- **Design an NFA with ∑ = {0, 1} accepts all string in which the third symbol from the right end is always 0.**

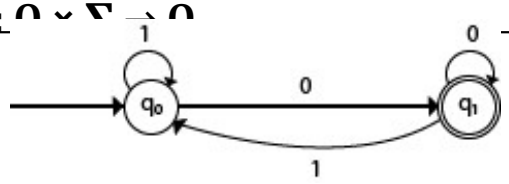
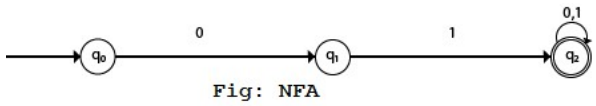


Thus we get the third symbol from the right end as '0' always. The NFA can be:

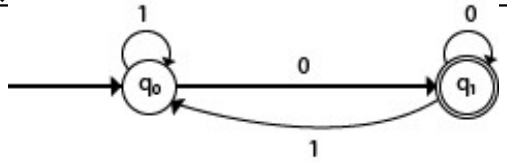
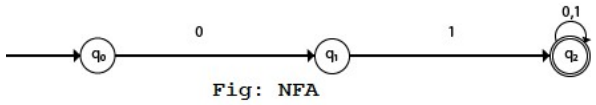


The above image is an NFA because in state q0 with input 0, we can either go to state q0 or q1.

Difference Between DFA and NFA

DFA	NFA
It stands for deterministic finite automata.	It stands for non-deterministic finite automata.
The transition takes place from a state to a single particular state for each input symbol.	For each input symbol, the transition can be to multiple next states.
There are no empty string transitions in DFA.	Empty string transitions are also permitted.
Backtracking is allowed in DFA.	In NFA, it is not always possible to backtrack.
DFA requires more space.	NFA requires less space.
There no such randomness as the transition can be determined in DFA.	There is much more randomness as the transition from one state to other cannot be determined in this.
Ex: δ: Q × Σ → Q 	**Ex: δ: Q × ∑ → 2Q**  Fig: NFA

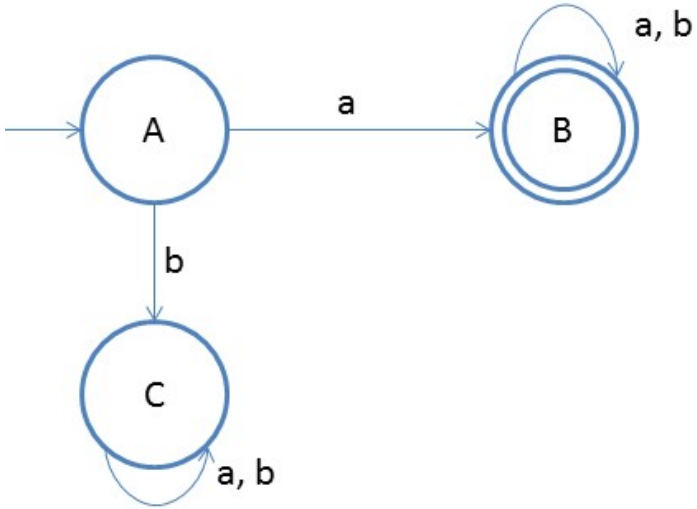
Difference Between DFA and NFA

DFA	NFA
It stands for deterministic finite automata.	It stands for non-deterministic finite automata.
The transition takes place from a state to a single particular state for each input symbol.	For each input symbol, the transition can be to multiple next states.
There are no empty string transitions in DFA.	Empty string transitions are also permitted.
Backtracking is allowed in DFA.	In NFA, it is not always possible to backtrack.
DFA requires more space.	NFA requires less space.
There no such randomness as the transition can be determined in DFA.	There is much more randomness as the transition from one state to other cannot be determined in this.
Ex: δ: Q × Σ → Q 	**Ex: δ: Q × ∑ → 2Q**  Fig: NFA

DFA Problem

Design a DFA in which set of all strings can be accepted which start with a.

- Given: Input alphabet, Σ={a, b}
- Language L = {a, ab, aba, aabb, aaabbba, …..}

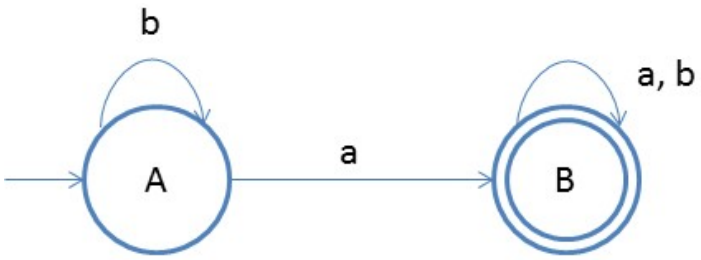


1.We know that first input must be a and from start state on a we should go to final state, so two states are confirmed: start state and final state.
2.And if the first input is something else than 'a' then string should not be accepted. And we don't care whatever comes on state C as state C now will work as dead state.
3.Once we reach final state we accept everything that is why we have a loop

DFA Problem

Design a DFA in which set of all strings can be accepted which contains ‘a’.

- Given: Input alphabet, Σ={a, b}
- Language L = {a, ab, **a**aabbb, abba,b**a**ba ….}

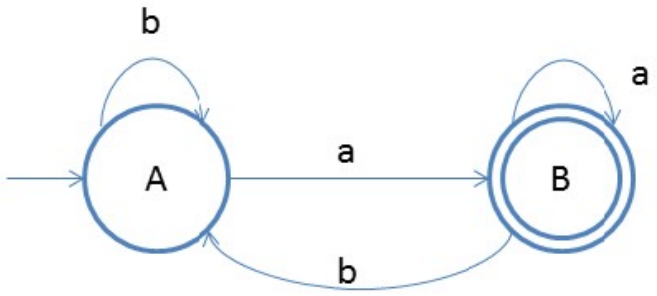


1.We know that first input must be a and from start state on a we should go to final state, so two states are confirmed: start state and final state.
2And if the first input is b then we will loop it to the start state itself.
3Once we reach final state we accept everything that is why we have a loop

DFA Problem

Design a DFA in which set of all strings can be accepted which end with ‘a’.

- Given: Input alphabet, Σ={a, b}
- Language L ={a,aa, aba, aaabbba, abba,baba ….}

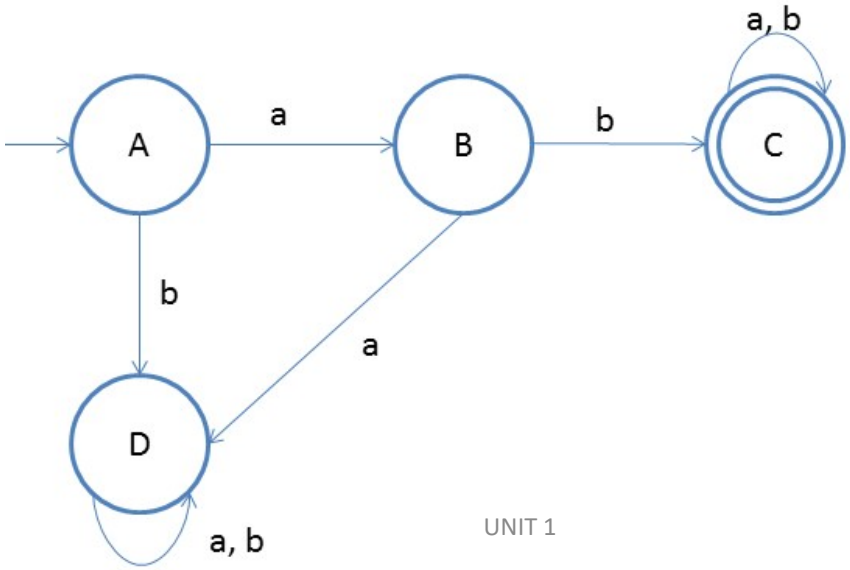


1. We will first start accepting ‘b’ on start state itself and then on a we will go to final state.
2.And on final state we do not care for a’s so we have to put a self-loop.
3. On final state if ‘b’ comes then we will redirect it start state so that any string which does not end with ‘a’ should not get accepted.

DFA Problem

Design a DFA in which set of all strings can be accepted which start with ab.

- Given: Input alphabet, Σ={a, b}
- Language L ={ab, aba, abaabbb, abba, ….}



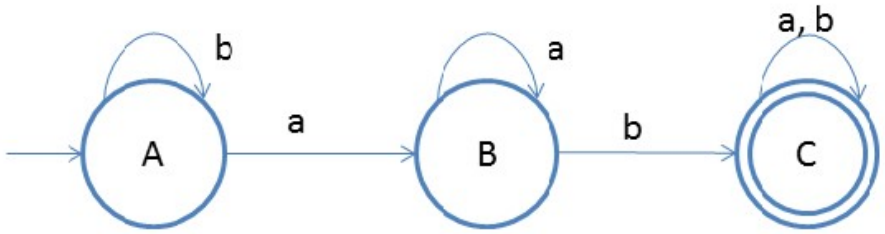
Follow the steps below and you will understand

- First we will make DFA for accepting the smallest string that is ‘ab’ and after that whatever comes we don’t care as the condition is satisfied.
- In DFA we have to take care of all the input alphabets at every state.
- so we have to take care of input symbol ‘b’ on state A, that is we have to reject it cause first ‘a’ should come then ‘b’.
- So we will make one more state D to take care of ‘b’ from state A
- On state B if a comes then we have to reject it, so we will direct this input to state D
- On state D for input a and b we do not care so we will make one self-loop

DFA Problem

Design a DFA in which set of all strings can be accepted which contain ab.

- Given: Input alphabet, Σ={a, b}
- Language L ={ab, aba, abaabbb, abba, bbabaabb ….}



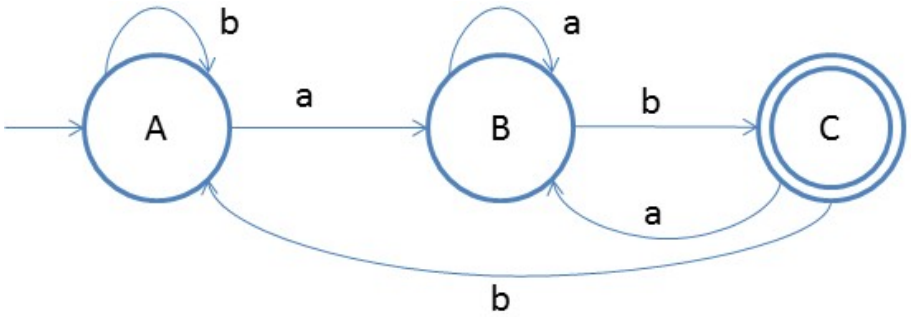
Follow the steps below and you will understand

- First we will make DFA for accepting the smallest string that is ‘ab’ and after that whatever comes at final state we don’t care as the condition is satisfied.
- In DFA we have to take care of all the input alphabets at every state.
- so we have to take care of input symbol ‘b’ on state A, that is we made self-loop on start state.
- On state B if a comes then we will accept it as repetition of ‘a’ after a will not ruin anything . Because we want ab in the entire string anywhere.
- On state C for input a and b we do not care so we will make one self-loop for ‘a’, ‘b’

DFA Problem

Design a DFA in which set of all strings can be accepted which ends with ab.

- Given: Input alphabet, Σ={a, b}
- Language L ={ab, abab, abaabbab, abbab, bbabaabab ….}

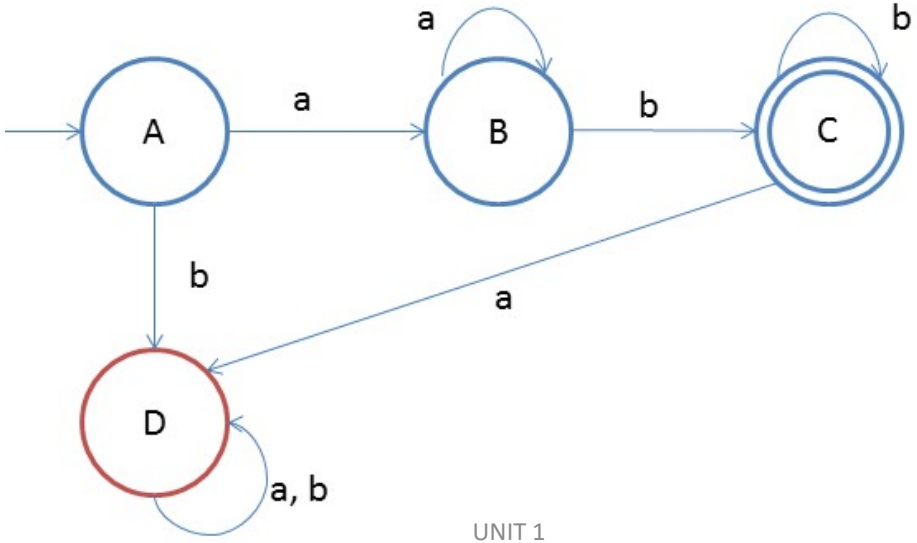


Follow the steps below and you will understand

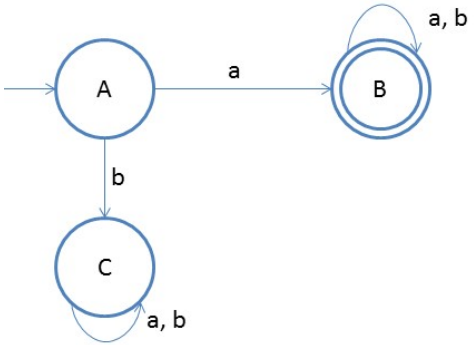
- First we will make DFA for accepting the smallest string that is ‘ab’
- In DFA we have to take care of all the input alphabets at every state.
- so we have to take care of input symbol ‘b’ on state A, that is we made self-loop on start state.
- On state B if a comes then we will accept it as repetition of ‘a’ after a will not ruin anything . Because we want ab in the end.
- on State C if b comes then that will be a problem as we only want ab in the end not bb, so we will direct b to state A.
- If ‘a’ comes on state C then we will direct it to state B and if one ‘b’ comes then we will be good by getting ‘ab’ in the end

DFA Problem

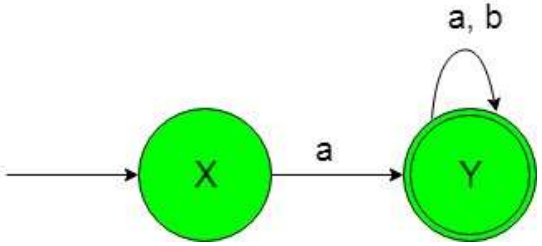
- Design a DFA such that:
- $L = \{a^n b^m \mid n,m \geq 1\}$ Given: Input alphabet, Σ={a, b}
- Language L = {ab, aab, aaab, abbb, aabb, aaaabbbb, ...}

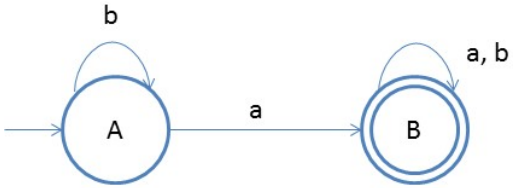


DFA

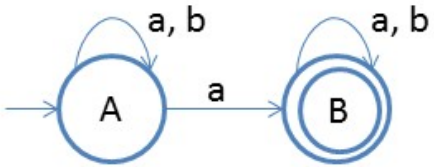


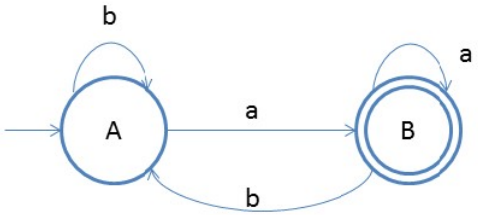
start with a.



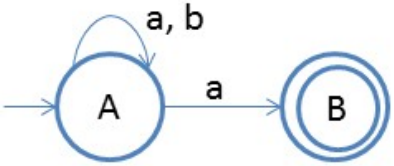


contains a.

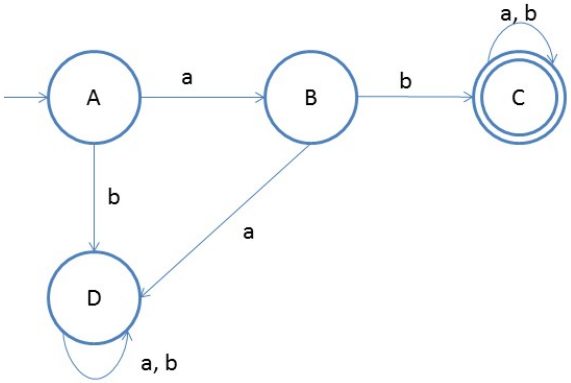


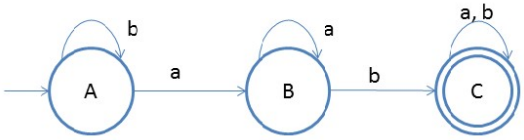


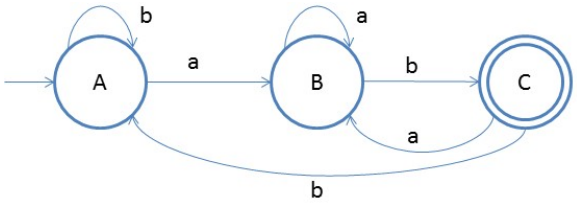
end with a.



DFA

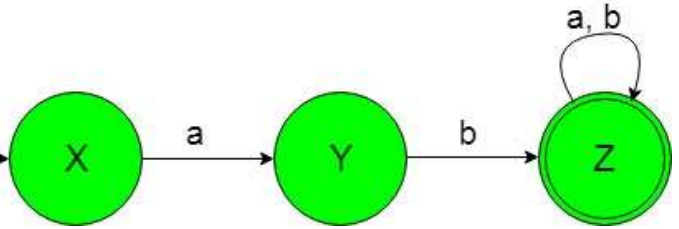




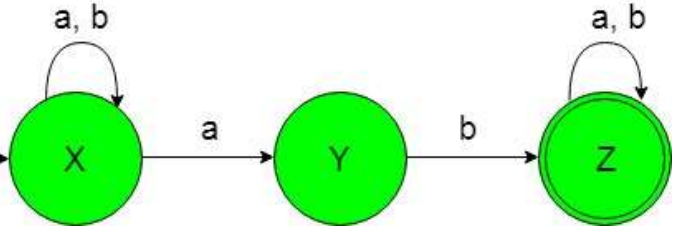


NFA

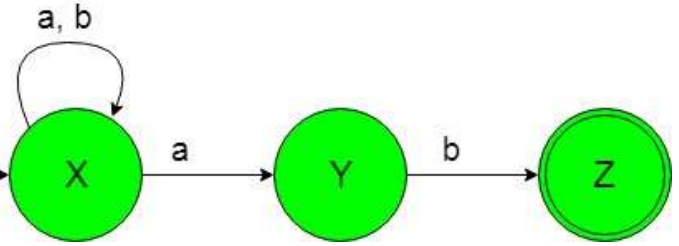
start with ab



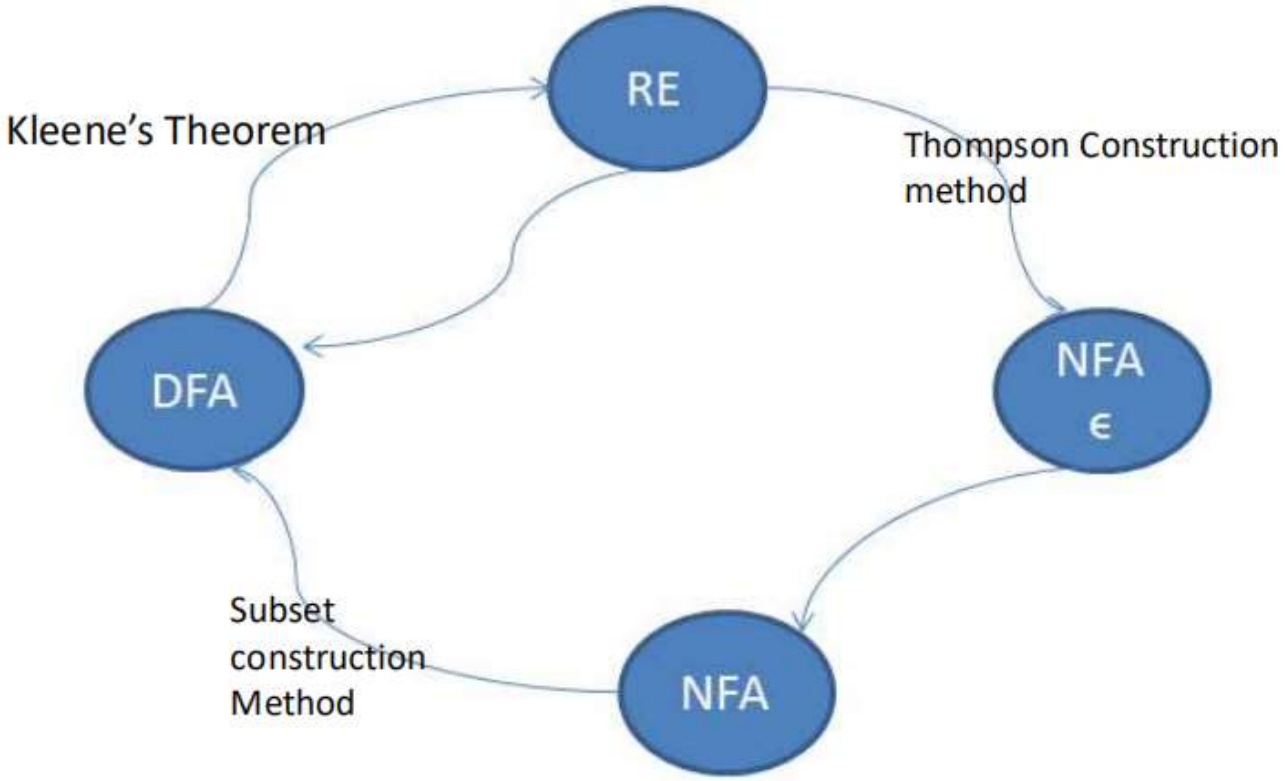
contains ab



end with ab



Conversions



Equivalence of NFA and DFA

- Every DFA is NFA but not vice versa.
- Both NFA and DFA have same power and each NFA can be translated into a DFA.
- There can be multiple final states in both DFA and NFA.
- NFA is more of a theoretical concept.
- DFA is used in Lexical Analysis in Compiler.

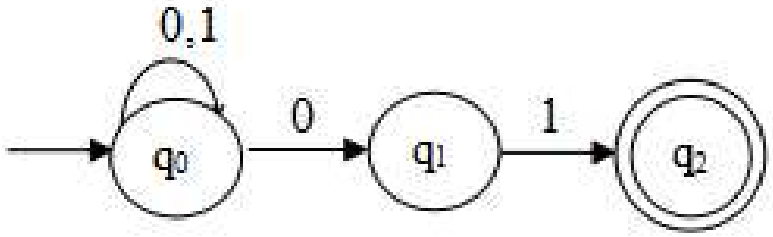
NFA to DFA (Subset Construction Method)

Procedure

1. If $Q=\{q_0, q_1, q_2\}$, then create subsets, $\{q_0\}, \{q_1\}, \{q_2\}, \{q_0, q_1\}, \{q_0, q_2\}, \{q_1, q_2\}, \{q_0, q_1, q_2\}$ and ϕ. Thus if no of states = n; the no of subsets $=2^n$.
2. Construct the transition table for the subsets over the given language, $\sum$.
3. Minimise the table by considering only the reachable states from the start state.
4. Draw the DFA from the minimised transition table.

NFA to DFA (Subset Construction Method)

- Construct a DFA from the following NFA that accepts all strings of 0's and 1's that end with '01'.



- Soln:

Step 1

Here ,

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

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

$q_0 = \{q_0\}$

$F = \{q_2\}$

Q \ Σ	0	1
→ q_0	$\{q_0, q_1\}$	$\{q_0\}$
q_1	ϕ	$\{q_2\}$
* q_2	ϕ	ϕ

Step 2

Subsets $Q_D = \{\phi, \{q_0\}, \{q_1\}, \{q_2\}, \{q_0, q_1\}, \{q_0, q_2\}, \{q_1, q_2\}, \{q_0, q_1, q_2\}\}$

Step 3

Transition table: δ_D

Q \ Σ	0	1
ϕ	ϕ	ϕ
→ $\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_1\}$	ϕ	$\{q_2\}$
★ $\{q_2\}$	ϕ	ϕ
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
★ $\{q_0, q_2\}$	$\{q_0, q_1\}$	$\{q_0\}$
★ $\{q_1, q_2\}$	ϕ	$\{q_2\}$
★ $\{q_0, q_1, q_2\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$

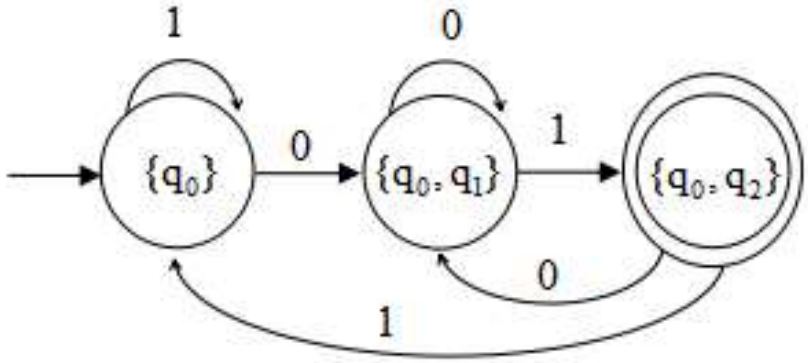
Step 4

Minimised transition table

Q \ Σ	0	1	
→ $\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$	← $\{q_0\}$ Processed
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$	← $\{q_0, q_1\}$ Processed
* $\{q_0, q_2\}$	$\{q_0, q_1\}$	$\{q_0\}$	← $\{q_0\}$ $\{q_0, q_1\}$ both are processed

Step 5

Minimised DFA



Convert the given NFA to DFA.

Q \ Σ	0	1
→p	{p, q}	{p}
q	{r}	{r}
r	{s}	ϕ
*s	{s}	{s}

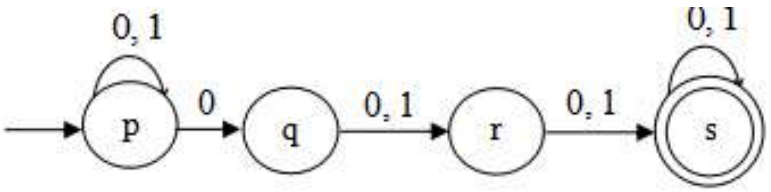
Step 1

$$Q_N = \{p, q, r, s\}$$

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

$$q_0 = \{p\}$$

$$F_N = \{s\}$$



Step 2

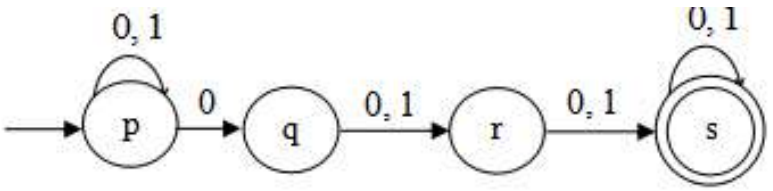
Subsets

$$Q_D = \{\phi, \{p\}, \{q\}, \{r\}, \{s\}, \{p, q\}, \{p, r\}, \{p, s\}\{q, r\}, \{q, s\}, \{r, s\}, \{p, q, r\}, \{p, q, s\}, \{q, r, s\}, \{p, r, s\}, \{p, q, r, s\}\}$$

$$|Q_D| = 2^4 = 16$$

Step 3

Transition table



Q \ Σ	0	1
ϕ	ϕ	ϕ
→{p}	{p, q}	{p}
{q}	{r}	{r}
{r}	{s}	ϕ
*{s}	{s}	{s}
{p, q}	{p, q, r}	{p, r}
{p, r}	{p, q, s}	{p}
*{p, s}	{p, q, s}	{p, s}
{q, r}	{r, s}	{r}
*{q, s}	{r, s}	{r, s}
*{r, s}	{s}	{s}
{p, q, r}	{p, q, r, s}	{p, r}
*{p, q, s}	{p, q, r, s}	{p, r, s}
*{p, r, s}	{p, q, s}	{p, s}
*{q, r, s}	{r, s}	{r, s}
*{p, q, r, s}	{p, q, r, s}	{p, r, s}

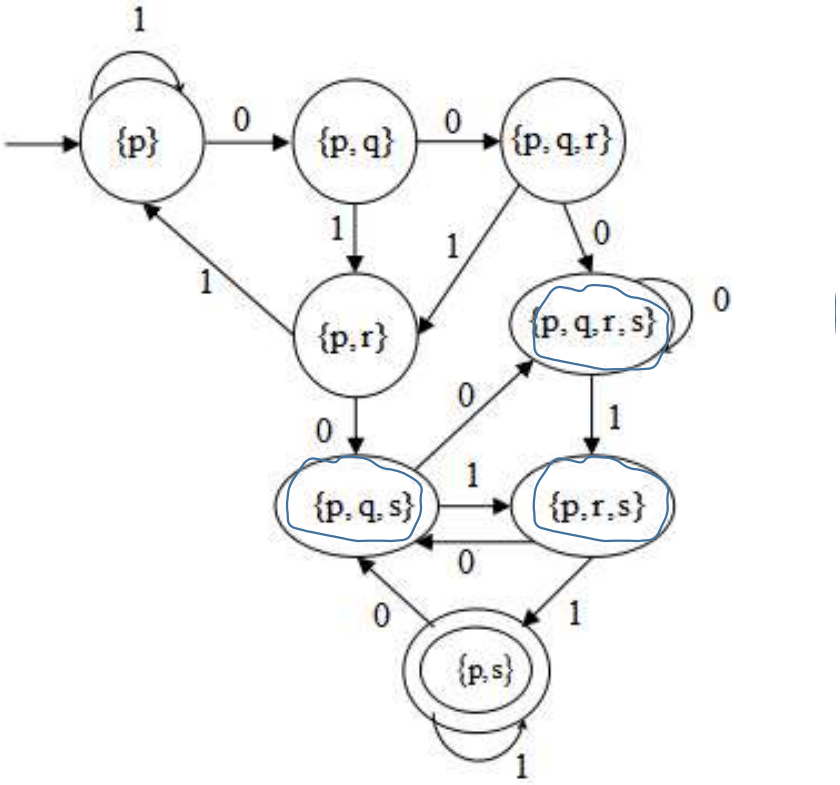
Step 4

Minimised transition table

Q \ Σ	0	1
→{p}	{p, q}	{p}
{p, q}	{p, q, r}	{p, r}
{p, q, r}	{p, q, r, s}	{p, r}
{p, r}	{p, q, s}	{p}
*{p, q, r, s}	{p, q, r, s}	{p, r, s}
*{p, q, s}	{p, q, r, s}	{p, r, s}
*{p, r, s}	{p, q, s}	{p, s}
* {p, s}	{p, q, s}	{p, s}

Step 5

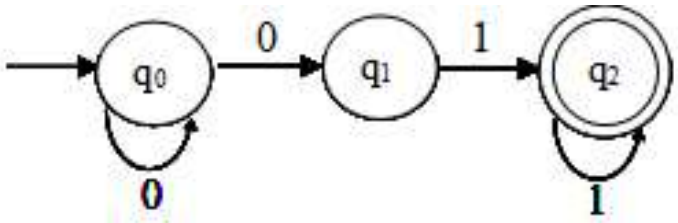
DFA



Note:

- After conversion, the number of states in the resulting DFA may or may not be same as NFA.
- The maximum number of states that may be present in the DFA are $2^{\text{Number of states in the NFA.}}$
- In general, the following relationship exists between the number of states in the NFA and DFA-
- $1 <= n <= 2^m$
- Here,
- n = Number of states in the DFA
- m = Number of states in the NFA
- In the resulting DFA, all those states that contain the final state(s) of NFA are treated as final states.

Construct a DFA for the following NFA.



Solution

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

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

$q_0 = \{q_0\}$

$F_N = \{q_2\}$

Q \ Σ	0	1
→ {q0}	{q0, q1}	ϕ
{q0}	ϕ	{q2}
*{q2}	ϕ	{q2}

Construction of DFA

$Q_D = \{\phi, \{q_0\}, \{q_1\}, \{q_2\}, \{q_0, q_1\}, \{q_0, q_2\}, \{q_1, q_2\}, \{q_0, q_1, q_2\}\}$

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

$q_0 = \{q_0\}$

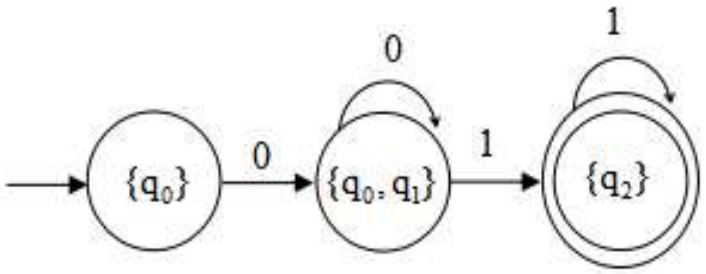
Transition table

Q \ Σ	0	1
ϕ	ϕ	ϕ
$\rightarrow\{q_0\}$	$\{q_0, q_1\}$	ϕ
$\{q_1\}$	ϕ	$\{q_2\}$
$*\{q_2\}$	ϕ	$\{q_2\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_2\}$
$*\{q_0, q_2\}$	$\{q_0, q_1\}$	$\{q_2\}$
$*\{q_1, q_2\}$	ϕ	$\{q_2\}$
$\{q_0, q_1, q_2\}$	$\{q_0, q_1\}$	$\{q_2\}$

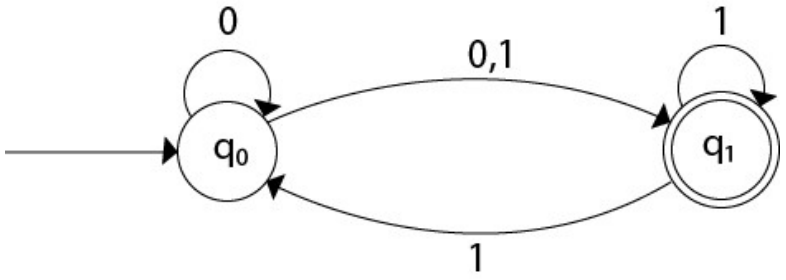
Minimized transition table

	Q \ Σ	0	1
	$\{q_0\}$	$\{q_0, q_1\}$	ϕ
$\rightarrow$	$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_2\}$
*	$\{q_2\}$	ϕ	$\{q_2\}$

Minimized DFA

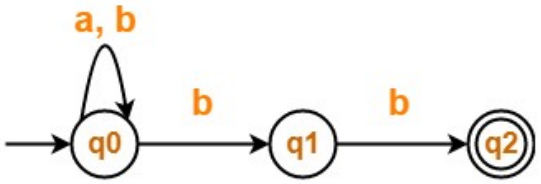


Home Work



- Convert the following NFA to

Q \ Σ	0	1
→ {p}	{q, s}	{q}
q *	{r}	{q, r}
r	{s}	{p}
* s	ϕ	{p}



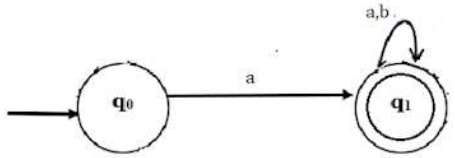
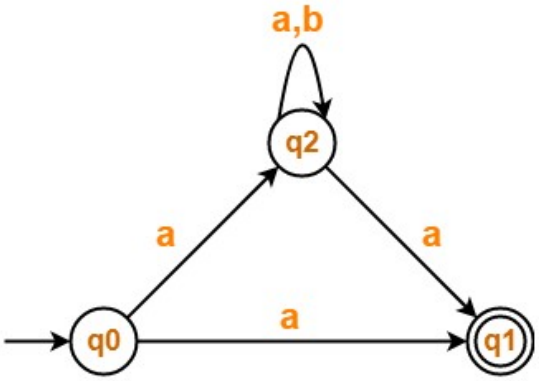


Fig: Transition diagram



NFA with Epsilon(ε) Transition (ε-NFA):

- The NFA with epsilon-transition is a finite state machine in which the transition from one state to another state is allowed without any input symbol i.e. empty string ε.
- Adding the transition for the empty string doesn't increase the computing power of the finite automata but adds some flexibility to construct then DFA and NFA.
- NFA with ∈ move:

If any FA contains ε transaction or move, the finite automata is called NFA with ∈ move.

Definition of ε-NFA

Formally, ε-NFA is defined by five tuples as:

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

Q is a finite set of states.

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

δ is the transition function where $\delta: Q \times \sum \cup \{\varepsilon\} \rightarrow 2^Q$

$\mathbf{q_0}$ is the initial state ($q_0 \in Q$).

F is a set of final state/states of Q ($F \subseteq Q$).

Example: ε-NFA

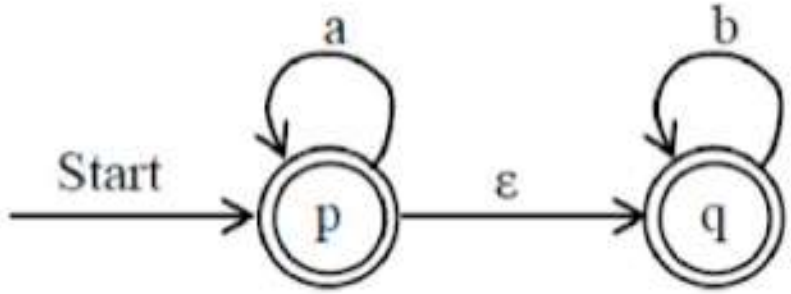


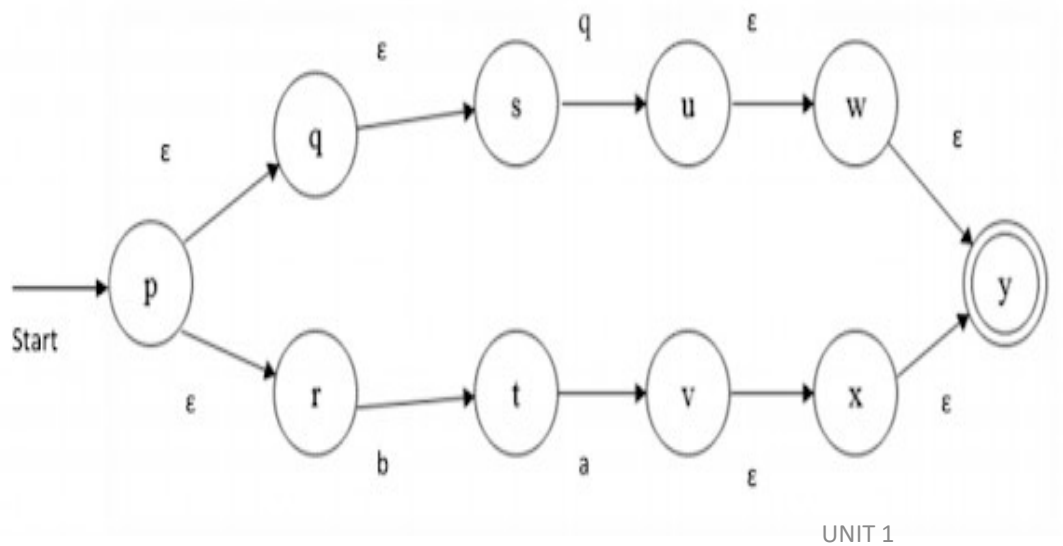
Fig: ε-NFA accepting the language {a, aa, ab, abb, b, bbb, ...

Transition Table:-

δ	ε	a	b
→*p	{q}	{p}	{φ}
*q	{φ}	{φ}	{q}

ε-closure

ε-closure for a given state A means a set of states which can be reached from the state A with only ε(null) move including the state A itself.



ε-closure (P) = {p,q,r,s}
ε-closure (q) = {q, s}
ε-closure (r) = {r}
ε-closure (s) = {s}
ε-closure (t) = {t}
ε-closure (u) = {u,w,y}
ε-closure (v) = {v, x, y}
ε-closure (w) = {w, y}
ε-closure (x) = {x, y}
ε-closure (y) = {y}

Steps for converting ε-NFA to DFA:

Step 1: We will take the ε-closure for the starting state of NFA as a starting state of DFA.

Step 2: Find the states for each input symbol that can be traversed from the present. That means the union of transition value and their closures for each state of NFA present in the current state of DFA.

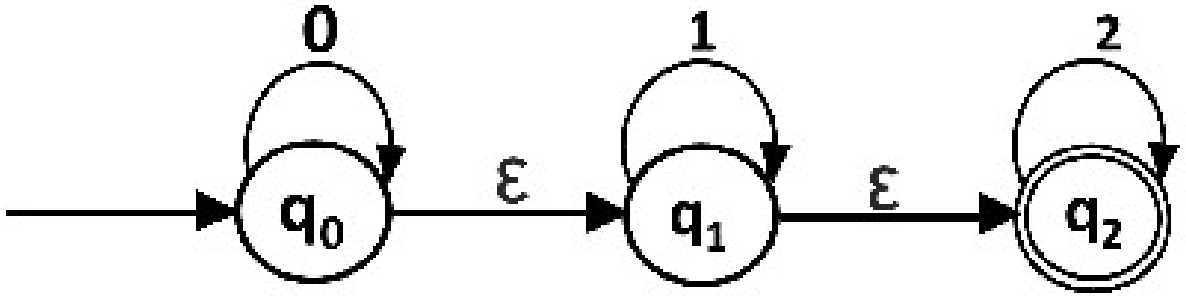
Step 3: If we found a new state, take it as current state and repeat step 2.

Step 4: Repeat Step 2 and Step 3 until there is no new state present in the transition table of DFA.

Step 5: Mark the states of DFA as a final state which contains the final state of NFA.

ε-NFA to DFA

- Convert the NFA with ε into its equivalent DFA.



Soln

Step 1

Let us obtain the ε-closure of each state.

ε-closure(q0) = {q0, q1, q2}

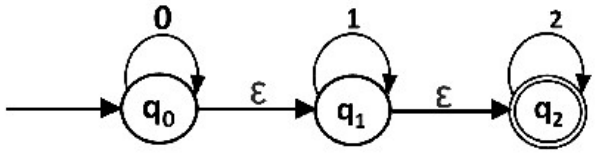
ε-closure(q1) = {q1, q2}

ε-closure(q2) = {q2}

Step 2

Now we will obtain δ' transition.

Let ε-closure(q0) = {q0, q1, q2} call it as state A.



δ'(A, 0) = ε-closure{δ((q0, q1, q2), 0)}

= ε-closure{δ(q0, 0) ∪ δ(q1, 0) ∪ δ(q2, 0)}

= ε-closure{q0}

= {q0, q1, q2}

δ'(A, 1) = ε-closure{δ((q0, q1, q2), 1)}

= ε-closure{δ(q0, 1) ∪ δ(q1, 1) ∪ δ(q2, 1)}

= ε-closure{q1}Thus we have obtained

= {q1, q2} call it as state B

δ'(A, 2) = ε-closure{δ((q0, q1, q2), 2)}

= ε-closure{δ(q0, 2) ∪ δ(q1, 2) ∪ δ(q2, 2)}

= ε-closure{q2}

= {q2} call it state C

Thus we have obtained

δ'(A, 0) = A

δ'(A, 1) = B

δ'(A, 2) = C

Step 3

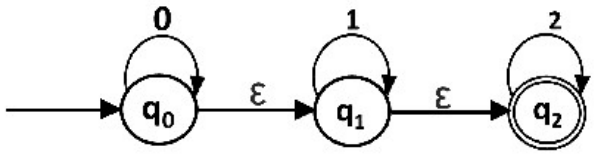
Now we will find the transitions on states B and C for each input.

$\delta'(B, 0) = \varepsilon\text{-closure}\{\delta((q1, q2), 0)\}$

$= \varepsilon\text{-closure}\{\delta(q1, 0) \cup \delta(q2, 0)\}$

$= \varepsilon\text{-closure}\{\phi\}$

$= \phi$



$\delta'(B, 1) = \varepsilon\text{-closure}\{\delta((q1, q2), 1)\}$

$= \varepsilon\text{-closure}\{\delta(q1, 1) \cup \delta(q2, 1)\}$

$= \varepsilon\text{-closure}\{q1\}$

$= \{q1, q2\}$ i.e. state B itself

$\delta'(B, 2) = \varepsilon\text{-closure}\{\delta((q1, q2), 2)\}$

$= \varepsilon\text{-closure}\{\delta(q1, 2) \cup \delta(q2, 2)\}$

$= \varepsilon\text{-closure}\{q2\}$

$= \{q2\}$ i.e. state C itself

Thus we have obtained

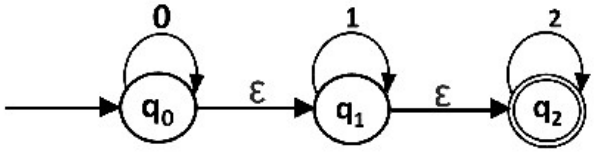
$\delta'(B, 0) = \phi$

$\delta'(B, 1) = B$

$\delta'(B, 2) = C$

Step 4

Now we will obtain transitions for C:



$$\delta'(C, 0) = \varepsilon\text{-closure}\{\delta(q2, 0)\}$$
$$= \varepsilon\text{-closure}\{\phi\}$$
$$= \phi$$

$$\delta'(C, 1) = \varepsilon\text{-closure}\{\delta(q2, 1)\}$$
$$= \varepsilon\text{-closure}\{\phi\}$$
$$= \phi$$

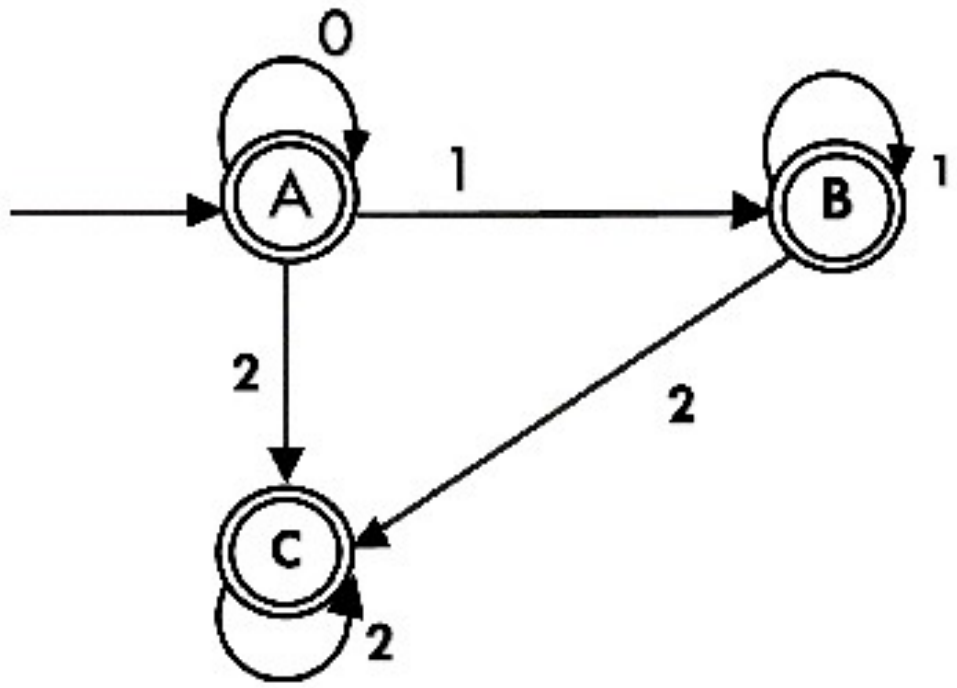
$$\delta'(C, 2) = \varepsilon\text{-closure}\{\delta(q2, 2)\}$$
$$= \{q2\}$$

Step 5

As A = {q0, q1, q2} in which final state q2 lies hence A is final state.

B = {q1, q2} in which the state q2 lies hence B is also final state.

C = {q2}, the state q2 lies hence C is also a final state.



Steps for converting ε-NFA to NFA

Step 1: Find out all the ε transitions from each state from Q. That will be called as ε-closure{q1} where qi ∈ Q.

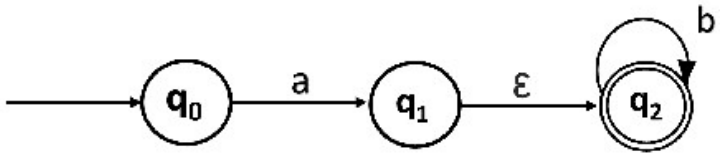
Step 2: Then δ' transitions can be obtained. The δ' transitions mean a ε-closure on δ moves.

Step 3: Repeat Step-2 for each input symbol and each state of given NFA.

Step 4: Using the resultant states, the transition table for equivalent NFA without ε can be built.

ε-NFA to NFA

- Convert the following NFA with ε to NFA without ε.



Step 1

We will first obtain ε-closures of q0, q1 and q2 as follows:

ε-closure(q0) = {q0}

ε-closure(q1) = {q1, q2}

ε-closure(q2) = {q2}

Step 2

Now the δ' transition on q0 for each input symbol is obtained as:

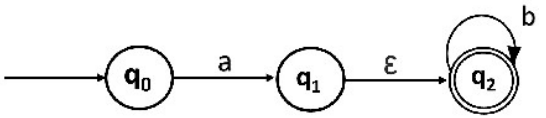
δ'(q0, a) = ε-closure(δ(δ^(q0, ε),a))

= ε-closure(δ(ε-closure(q0),a))

= ε-closure(δ(q0, a))

= ε-closure(q1)

= {q1, q2}



ε-closure(q0) = {q0}
ε-closure(q1) = {q1, q2}
ε-closure(q2) = {q2}

δ'(q0, b) = ε-closure(δ(δ^(q0, ε),b))

= ε-closure(δ(ε-closure(q0),b))

= ε-closure(δ(q0, b))

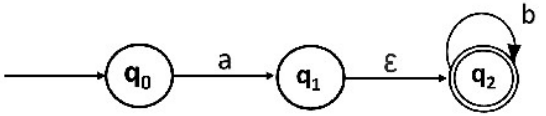
= Φ

Step 3

Now the δ' transition on q1for each input symbol is obtained as:

δ'(q1, a) = ε-closure(δ(δ^(q1, ε),a))
= ε-closure(δ(ε-closure(q1),a))
= ε-closure(δ(q1, q2), a)
= ε-closure(δ(q1, a) ∪ δ(q2, a))
= ε-closure(Φ ∪ Φ)
= Φ

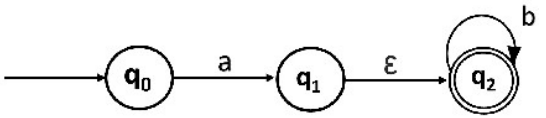
δ'(q1, b) = ε-closure(δ(δ^(q1, ε),b))
= ε-closure(δ(ε-closure(q1),b))
= ε-closure(δ(q1, q2), b)
= ε-closure(δ(q1, b) ∪ δ(q2, b))
= ε-closure(Φ ∪ q2)
= {q2}



ε-closure(q0) = {q0}
ε-closure(q1) = {q1, q2}
ε-closure(q2) = {q2}

Step 4

The δ' transition on q2 is obtained as:



ε-closure(q0) = {q0}
ε-closure(q1) = {q1, q2}
ε-closure(q2) = {q2}

δ'(q2, a) = ε-closure(δ(δ^(q2, ε),a))
= ε-closure(δ(ε-closure(q2),a))
= ε-closure(δ(q2, a))
= ε-closure(Φ)
= Φ

δ'(q2, b) = ε-closure(δ(δ^(q2, ε),b))
= ε-closure(δ(ε-closure(q2),b))
= ε-closure(δ(q2, b))
= ε-closure(q2)
= {q2}

Now we will summarize all the computed δ' transitions:

δ'(q0, a) = {q0, q1}

δ'(q0, b) = Φ

δ'(q1, a) = Φ

δ'(q1, b) = {q2}

δ'(q2, a) = Φ

δ'(q2, b) = {q2}

The transition table can be:

States	a	b
→q0	{q1, q2}	Φ
*q1	Φ	{q2}
*q2	Φ	{q2}

Step 4

- State q1 and q2 become the final state as **ε-closure of q1 and q2 contain the final state q2**.
- The NFA can be shown by the following transition diagram:

