SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE
www.sathyabama.ac.in

SCHOOL OF COMPUTING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

UNIT – I – THEORY OF COMPUTATION – SCSA1302

SYLLABUS

Introduction - Regular Languages and Regular Expressions - Deterministic Finite Automata - Non-Deterministic Finite Automata - NFA to DFA Conversion – NFA NULL Construction - NFA NULL to NFA Conversion - Kleene's Theorem Part I & II.

1. Introduction

1.1. What is Theory of Computation or Automata Theory?

- Theory of Computation is how efficiently problems can be solved on a model of computation, using an algorithm.
- It is mainly about what kind of things can you really compute mechanically, how fast and how much space does it take to complete the task.
- Ex1: To design a machine that accepts all binary strings ends in 0 and reject all other that does not ends in 0.

  11011010 – Accept
- Ex2: To design a machine to accepts all valid 'C' codes

  Machine will check the binary equivalent of this code and from this binary equivalent it tells weather it is valid piece of C code or invalid.

  Question : Is it possible to design a machine?

  Yes – The best example is Compiler.
- Ex3: To design a machine that accepts all valid 'C' codes and never goes into infinite.

  Question : Is it possible to design a machine?

  No

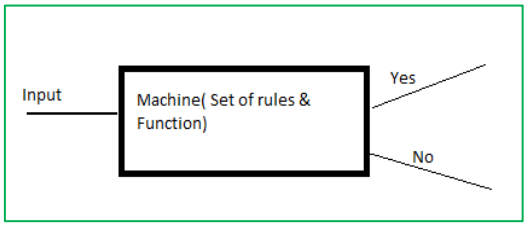


Figure 1.1. Model of a TOC

Table 1.1 Modelling Languages and Language Acceptor

Language Generator	Language Acceptor
Regular Language	Finite Automata DFA NFA
Context Free Language	PDA DPDA NDPDA
Recursive Language	Turing Machine

LAYERS AND LEVELS IN THEORY OF COMPUTATION:

- FSM – Finite State Machine – Simplest model of Computation and it has very limited memory.

Perform low level computation and calculations

- CFL – Context Free Language

Performs some higher level of computation.

- Turing Machine – Much powerful model perform very high level computation designed by Alan Turing in 1940.
- Undecidable – Problem cannot be solved mechanically is falls under undecidable layer.

1.2 Basic Units of Regular Language:

Alphabets (∑) : { a, b} or {0,1}

String(w) : Collection of input alphabets

Language (L) : Collection of Strings

Empty Set : Ø

NULL String : ε or λ

1.3. Finding Language using Conditions:

1. Find L with 0'S and 1's with odd no. of 1's

∑ ={ 0,1}

w ={ 1,01,10,100,010, 111,1011……..}

w ={ 1,01,10,100,010, 110 ,111,1011……..} --invalid

L = {w/w consists of odd no. of 1's}

2. Find L with 0's and 1's with even no. of 1's

∑ ={ 0,1}

w ={Λ ,11,011,101,110,0110, 1010, ……..}

w ={Λ ,11,011,101,100, 110,0110, 1010, …..} --invalid

L = {w/w consists of even no. of 1's}

3. Find L with a's and b's with even no. of a's

∑ ={ a,b}

w ={Λ ,aa,baa,aba,aab,baab, abba,abab ……..}

w ={Λ ,aa,baa,aaa,aab,baab, abba,abab ……..} --invalid

L = {w/w consists of even no. of a's}

4. Find L with a's and b's with even no of a's and b's.

∑ ={ a,b}

w ={Λ ,aa,bb,aabb,abab,baba,bbaa ……..}

L = {w/w consists of even no. of a's and b's}

5. Find L with a's and b's having a substring aa

∑ ={ a,b}

w ={aa,baa,aab,baab, abaa,aabaa ……..}

L = {w/w consists of a substring aa}

1.4. Regular Language:

- A language is regular if there exits a DFA for that language.
- The language accepted by DFA is RL.

1.5. Regular Expression:

- A Mathematical notation used to describe the regular language.
- This is formed by using 3 Symbols:
  (i). [dot operator] – for concatenation
  (ii) + [Union operator] –at least 1 occurrence

  eg) 1^+ = {1,11,111,-------}

  (iii) {*} [Closure Operator] – Zero or more occurrences

  eg) 1* = {Λ ,1,11,111,…..}

1.6 Basic Regular Expressions:

- Ø is a RE and denotes the empty set.
- ε is a RE and denotes the set { ε }
- For each a in ∑, a is a RE and denotes the set {a}
- If r and s are RE that denoting the languages R and S respectively, then,
  - (r+s) is a RE that denotes the set (RUS)
  - (r.s) is a RE that denotes the set R.S
  - (r)* is a RE that denotes the set R*

1.7 Problems on RE:

1. **Write the RE for the language of even no. of 1's.**

   ∑={0,1}

   W={∧,11,011,101,110,1111,1100,....}

   RE=(11)*

2. **Write the RE for the language of odd no. of 1's.**

   ∑={0,1}

   W={1,10,01,100,111,1110,.....}

   RE=(11)*.1

3. **Write the RE for the language of any length including Λ.(a,b)**

   ∑={a,b}

   W={∧,a,b,aa,ab,aab,baa,aaaa,.....}

   RE=(a+b)*

4. **Write the RE with a string starting with a.**

   ∑={a,b}

   W={a,aa,ab,aaa,abb,.....}

   RE=(a).(a+b)*

5. **Write the RE for the language having a substring aa.**

   ∑={a,b}

   W={aa,baa,aab,baaa,aaaa....}

   RE=(a+b)*.aa.(a+b)*

6. **Write the RE with a string starting with either a or ab.**

   ∑={a,b}

W={a,ab,aa,aba,abb,abbb,....}

RE=(a+ab).(a+b)*

7. **Write RE with a string consists of a's and b's ending with abb.**

∑={a,b}

W={abb,aabb,babb,aaabb,.....}

RE=(a+b)*.abb

8. **Write RE with a string consists of a's and b's starting with abb.**

∑={a,b}

W={abb,abbb,abba,abbaa,.....}

RE=abb.(a+b)*

9. **Write the RE for identifiers in 'C' Programming.**

Letter=(a-z)

Digit=(0-9)

RE=(letter+_).(letter+_+digit)*

10. **Write the RE with a string that should not start with two 0's.**

∑={0,1}

W={0,1,01,011,110,....}

RE=(01+1).(1+0)*

11. **Write RE with a string that begins and ends with double consecutive letters.**

∑={a,b}

W={aa,bb,aabb,bbaa,aabaa,bbbaa,.....}

RE=(aa+bb).(a+b)*.(aa+bb)

12. **Strings of a's and b's in which 3rd symbol from right end is 'a'.**

∑={a,b}

W={aaa,abb,aabb,bbabb,…}

RE=(a+b)*a.(a+b)(a+b)

13. **Write RE for the strings consisting of atleast 1 'a' followed by strings consisting of atleast 1 'b' followed by strings consisting of atleast 1 'c'.**

∑={a,b,c}

W={abc,aabc,abbc,aabbcc,aaabbc,…}

RE=$a^+.b^+.c^+$

14. Write the RE for the strings over {0,1} of length 6 or less.

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

Length 6-{001100,110011,010101,000000,..}

(0+1).(0+1).(0+1).(0+1).(0+1).(0+1)

$(0+1+\Lambda)^6$

(0+1)6

(0+1+Λ)*

15. Write RE for the string(a,b) whose length divisible by 3.

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

W={Λ,aaa,aba,abb,aab,aabbaa,….}

RE=(aaa+aab+aba+baa+bbb+baa+abb+bab)*

1.8 What is Finite Automata?

- Simplest model of a computing device.
- Finite automata are used to recognize patterns.
- A machine that accepts Regular Language.
- It takes the string of symbol as input and changes its state accordingly. When the desired symbol is found, then the transition occurs.
- At the time of transition, the automata can either move to the next state or stay in the same state.
- Finite automata have two states, Accept state or Reject state. When the input string is processed successfully, and the automata reached its final state, then it will accept.

Applications:

- Compilers
- Text processing
- Hardware design.

Types of Automata:

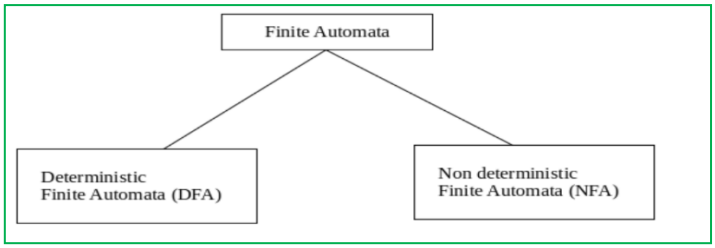


Figure 1.2 Types of Automata

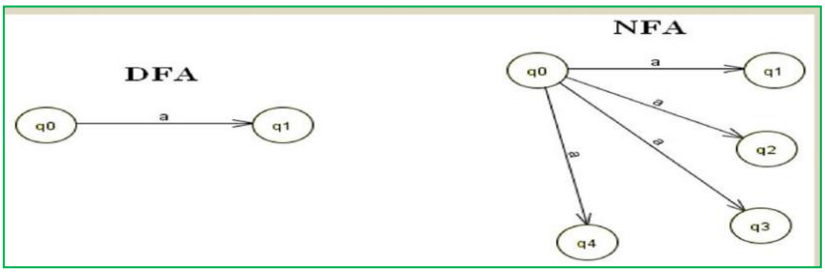


Figure 1.3 Difference between DFA and NFA

2. DFA (Deterministic Finite Automata):

- Only one path for specific input from the current state to the next state.
- DFA does not accept the null move.
- It is used in Lexical Analysis phase in Compilers.

  Example: RE=$(a+b)^+$

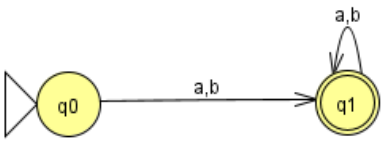


Figure 1.4 Sample DFA

Definition of DFA:

A finite automaton is a collection of 5-tuple (Q, ∑, δ, q0, F), where:

Q: finite set of states

∑: finite set of the input symbol

q0: initial state

F: final state

δ: Transition function

2.2 Construction of DFA:

1. **Construct DFA to accept strings of a's and b's having a substring aa.**

   W={aa,aaa,baa,aab,aabb,abaa…..}

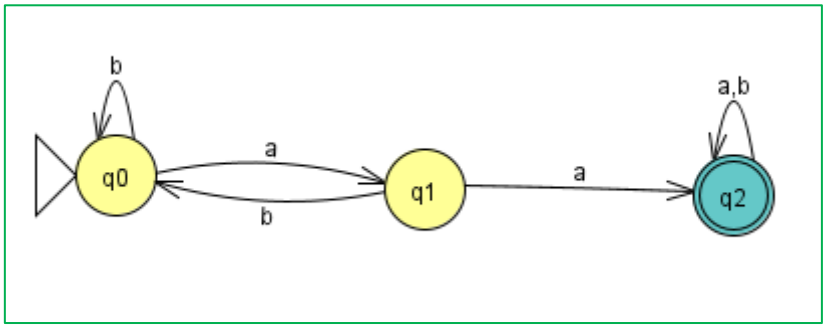


Figure 1.5 State Diagram

DFA Definition

M=(Q, ∑,q_0, δ,A)

Q={q0,q1,q2}

∑={a,b}

q_0 =q0

A = q2

δ –Transition Function

δ(q0,a)=q1

δ(q0,b)=q0

δ(q1,a)=q2

δ(q1,b)=q0

δ(q2,a)=q2

δ(q2,b)=q2

2. **Construct DFA to accept string of a's and b's having exactly 1 'a'.**

   W={a,ab,ba,bba,abb,bbba,….}

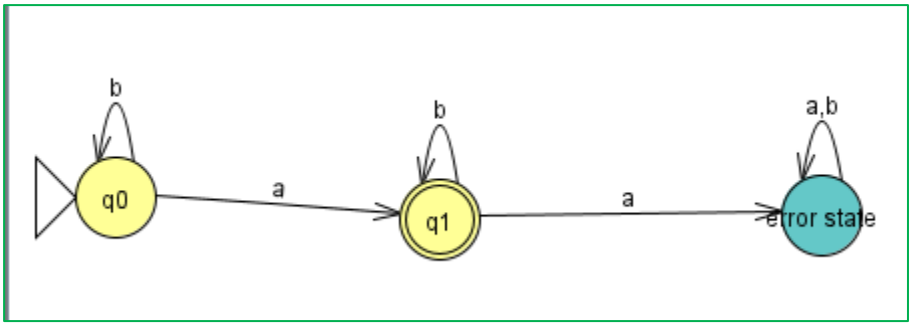


Figure 1.6 State Diagram

DFA Definition

M=(Q, ∑,q0, δ,A)

Q={q0,q1}

∑={a,b}

q0 =q0

A = q1

δ –Transition Function

δ(q0,a)=q1

δ(q0,b)=q0

δ(q1,a)=null

δ(q1,b)=q1

3. Construct DFA to accept strings of a's and b's with atleast 1 'a'.

W={a,aa,ab,ba,bba,baa,….}

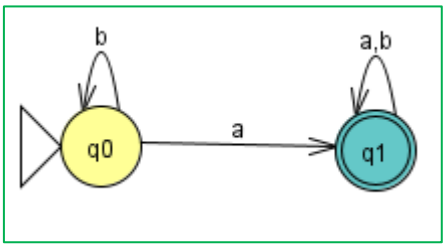


Figure 1.7 State Diagram

DFA Definition

M=(Q, ∑,q0, δ,A)

Q={q0,q1}

∑={a,b}

q0 =q0

A = q1

δ –Transition Function

δ(q0,a)=q1

δ(q0,b)=q0

δ(q1,a)=q1

δ(q1,b)=q1

4. **Construct DFA to accept strings of 0's and 1's with substring 01.**

W={01,001,010,011,1001,…..}

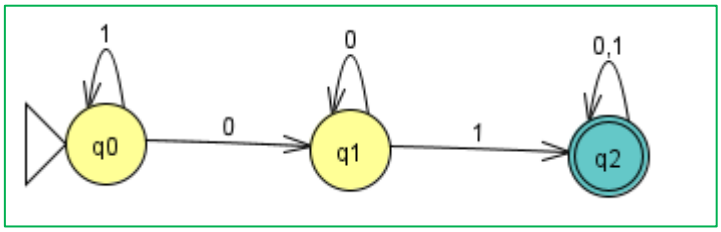


Figure 1.8 State Diagram

DFA Definition:

M=(Q, ∑,q0, δ,A)

Q={q0,q1,q2}

∑={0,1}

q0 =q0

A = q2

δ –Transition Function

δ(q0,0)=q1

δ(q0,1)=q0

δ(q1,0)=q1

δ(q1,1)=q2

δ(q2,0)=q2

δ(q2,1)=q2

5. **Construct DFA to accept strings of a's and b' not more than 3 'a'.**

W={Λ,a,aa,aaa,aba,abb,bbb,aabb,…..}

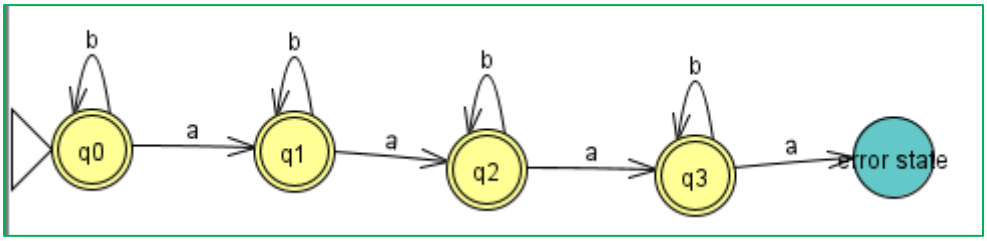


Figure 1.9 State Diagram

DFA Definition

M=(Q, ∑,q0, δ, A)

Q={q0,q1,q2,q3}

∑={a,b}

q0 =q0

A = {q0,q1,q2,q3}

δ –Transition Function

δ(q0,a)=q1

δ(q0,b)=q0

δ(q1,a)=q2

δ(q1,b)=q1

δ(q2,a)=q3

δ(q2,b)=q2

δ(q3,a)=null

δ(q3,b)=q3

6. Construct DFA to accept strings that end with 011.

W={011,1011,0011,11011,00011,100011…..}001000011,01100011,

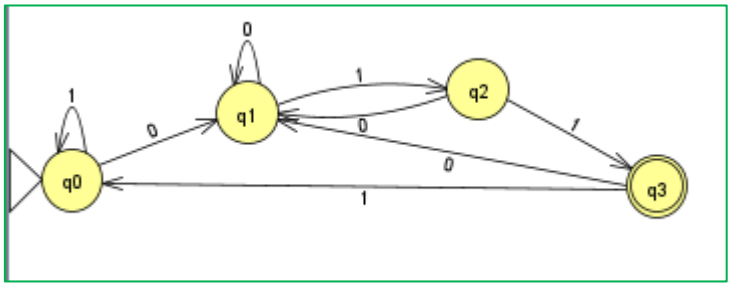


Figure 1.10 State Diagram

DFA Definition

M=(Q, ∑,q0, δ, A)

Q={q0,q1,q2,q3}

∑={0,1}

q0 =q0

A = q3

δ –Transition Function

δ(q0,0)=q1

δ(q0,1)=q0

δ(q1,0)=q1

δ(q1,1)=q2

δ(q2,0)=q1

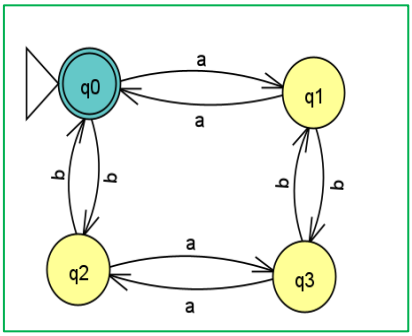
δ(q2,1)=q3

δ(q3,0)=q1

δ(q3,1)=q0

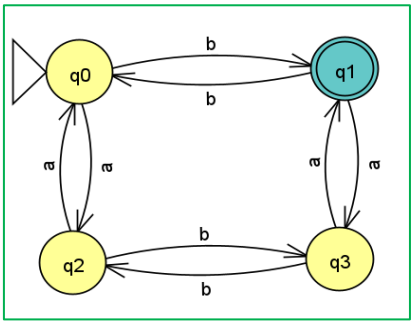
7. **Construct DFA to accept strings of a's and b's with even number of a's and b's.**

W={Λ,aa,bb,aabb,abab,baba,aaaabb…}



8. **Construct DFA with even number of a's and odd no. of b's.**

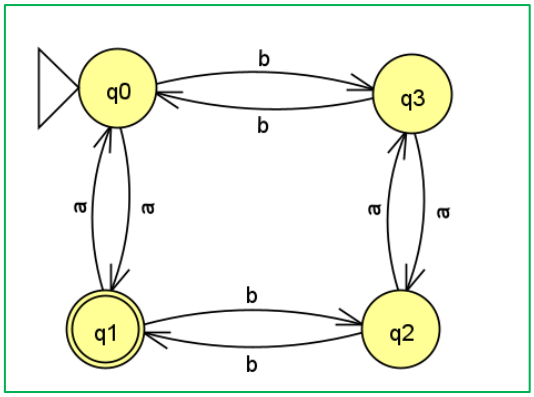
W={b,aab,aba,baa,ababb,aaaab,aababab,….}



M=(Q, ∑,q0, δ, A)

9. **Construct DFA with odd number of a's and even no. of b's.**

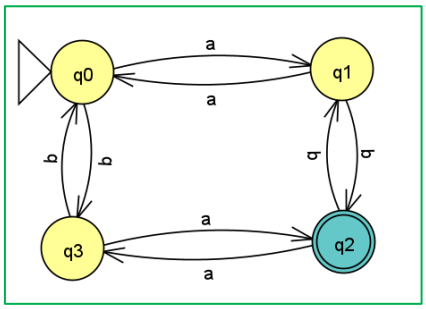
W={a,abb,bba,bab,abbaa,ababa,…..}



M=(Q, ∑,q0, δ, A)

10. **Construct DFA with odd number of a's and b's.**

W={ab,ba,babb,ababab,…….}



M=(Q, ∑,q0, δ, A)

11. Construct DFA to accept odd and even no's represented using binary notation.

0->0

1->1

2->10

3->11

4->100

5->101

6->110

7->111

8->1000

010

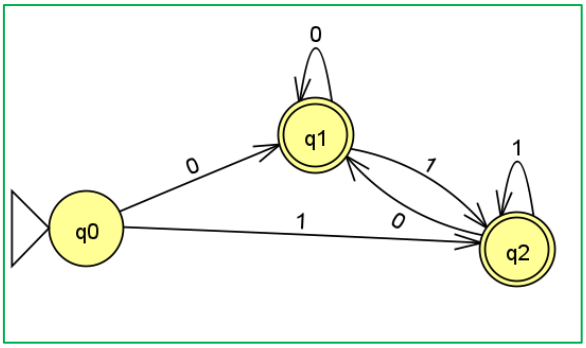


Figure 1.11.State Diagram

M=(Q, ∑,q0, δ, A)

2.2 Extension of δ: δ*

δ-> for one input symbol

δ=Q x ∑ -> Q

δ*->the state in which FA ends up if it begins in state q and receives a string x of input symbols

δ*(q,x) i.e. δ*: Q x ∑*->Q

Recursive Definition of δ*:

Let M=(Q, ∑, q0, A , δ) be finite automata, we define the function δ*: Q x ∑*->Q as follows:

(i) for any qϵQ, δ*(q,Λ)=q

(ii) for any y ϵ∑*, a ϵ∑ and q ϵQ

δ*(q,ya) = δ(δ*(q,y), a)

For strings of of length 1, δ and δ* can be used interchangeably.

PROBLEMS:

Find δ*(q0,abc)

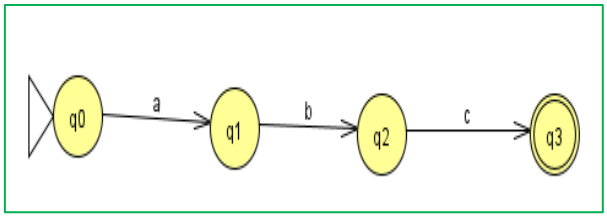


Figure 1.12 State Diagram

Solution :

δ*(q,Λ)=q

δ*(q,ya) = δ(δ*(q,y), a)

Λa=aΛ=a

$\delta^*(q0,abc) = \delta(\delta^*(q0,ab),c)$

$= \delta(\delta(\delta^*(q0,a),b),c)$

$= \delta(\delta(\delta(\delta^*(q0,\Lambda),a),b),c)$

$= \delta(\delta\ \delta(q0,a),b),c)$

$= \delta(\delta(q1,b),c)$

$= \delta(q2,c)$

$=q3$

2.4 STRING ACCEPTANCE BY FINITE AUTOMATA:

Let $M=(Q,\sum,q0,A,\delta)$ be an FA. A string $x\varepsilon\sum^*$ is accepted by M, if

$\delta^*(q0,x)\epsilon A$

2.5 LANGUAGE ACCEPTANCE BY FINITE AUTOMATA BY FINITE AUTOMATA:

The language is accepted by M or the language recognized by M is the set,

$L(M)= \{x \mid x \epsilon \sum^* \text{ and } \delta^*(q0,x)\epsilon A\}$

3.NON-DETERMINISTIC FINITE AUTOMATA:

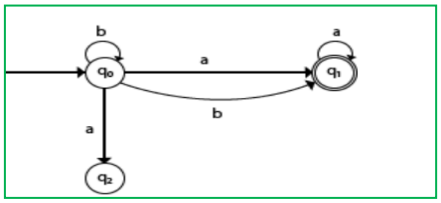


Figure 1.13 NFA

- 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.
- Types
  - (i) NFA without Λ
  - (ii) NFA with Λ

Advantages of NFA over DFA:

- DFAs are faster but more complex.
- Build a FA representing the language that is a union, intersection, concatenation etc. of two (or more) languages easily by using NFA's.

Definition:

NFA has 5 tuples $M=(Q,\sum,q0,A,\delta)$, where

Q: finite set of states

∑: finite set of the input symbol

q0: initial state

A: final state

δ: Transition function

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

3.1 Example:

Obtain an NFA to accept the language L={ w| w ε $abab^n$ or aba^n}

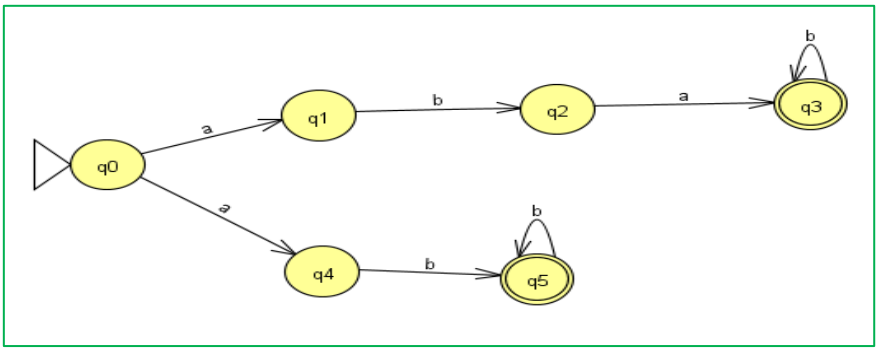


Figure 1.4 NFA State Diagram

Q={q0,q1,q2,q3,q4,q5}

∑={a,b}

q0={q0}

A={q3,q3}

δ:

δ(q0,a)={q1,q4}

δ(q0,b)= ∅

δ(q1,a)= ∅

δ(q1,b)={q2}

δ(q2,a)={q3}

δ(q2,b)= ∅

δ(q3,a)= ∅

δ(q3,b)={q3}

δ(q4,a)= ∅

δ(q4,b)={q5}

δ(q5,a)={q5}

δ(q5,b)= ∅

3.2 Problem: convert NFA to DFA [Subset Construction Method]

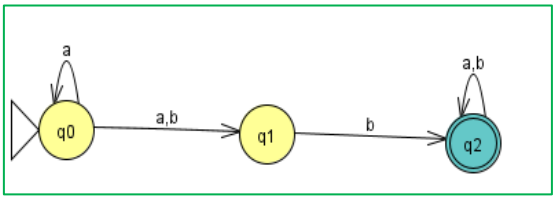


Figure 1.15 NFA State Diagram

Solution:

$M_N=(Q_N,\sum_N,q0,A_N,\delta_N)$ be an NFA.

$\delta_N(q0,a)=\{q0,q1)$

$\delta_N(q0,b)=\{q1)$

$\delta_N(q1,a)= \emptyset$

$\delta_N(q1,b)=\{q2)$

$\delta_N(q2,a)=\{q2)$

$\delta_N(q2,b)=\{q2)$

$M_D=(Q_D,\sum_D,q0,A_D,\delta_D)$ be an DFA.

Step 1:

Start state of NFA is the start state of DFA.

Obtain the transitions from this state.

q0 is the start state.

$\delta_D (q0,a)=\delta_N(q0,a)$

$=[q0,q1]$----→A

$\delta_D (q0,b)=\delta_N(q0,b)$

$=[q1]$----→B

State	a	b
q0	A	B
A		
B		

Step: 2 Transition from A:

$\delta_D (A,a)= \delta_N(A,a)$

$= \delta_N([q0,q1],a)$

$= \delta_N(q0,a) U \delta_N(q1,a)$

$=\{q0,q1\} U \emptyset$

$=[q0,q1]$------→A

$\delta_D (A,b)= \delta_N(A,b)$

$= \delta_N([q0,q1],b)$

$= \delta_N(q0,b) \cup \delta_N(q1,b)$

$=\{q1\} \cup \{q2\}$

$=[q1,q2] \rightarrow C$

State	a	b
q0	A	B
A	A	C
B		
C		

Transition from B:

δ_D (B,a)= δ_N(B,a)

$= \delta_N([q1],a)$

$= \delta_N(q1,a)$

$=\emptyset$

δ_D (B,b)= δ_N(B,b)

$= \delta_N([q1],b)$

$= \delta_N(q1,b)$

$=\{q2\} \rightarrow D$

State	a	b
q0	A	B
A	A	C
B	∅	D
C		
D		

Transition from C:

δ_D (C,a)= δ_N(C,a)

$= \delta_N([q1,q2],a)$

$= \delta_N(q1,a) \cup \delta_N(q2,a)$

$=\emptyset \cup q2$

$=[q2] \rightarrow D$

δ_D (C,b)= δ_N(C,b)

$= \delta_N([q1,q2],b)$

$= \delta_N(q1,b) \cup \delta_N(q2,b)$

$=\{q2\} \cup q2$

=[q2]------→D

State	a	b
q0	A	B
A	A	C
B	∅	D
C	D	D
D		

Transition from D:

δ_D (D,a)= δ_N(D,a)

= δ_N([q2],a)

= δ_N(q2,a)

=[q2]------→D

δ_D (D,b)= δ_N(D,b)

= δ_N([q2],b)

= δ_N(q2,b)

=q2---→D

State	a	b
q0	A	B
A	A	C
B	∅	D
C	D	D
D	D	D

Step: 3 construct the DFA

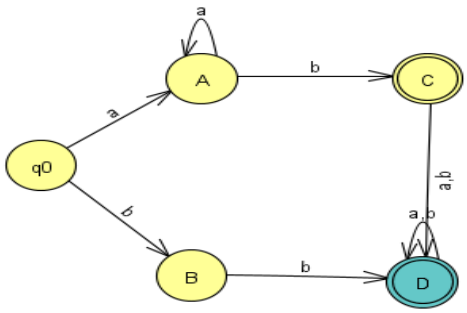


Figure 1.16 State Diagram

The final state q2 of NFA is there in C and D states in DFA. So make C and D as final state.

3.3 STRING ACCEPTANCE BY AN NFA:

Let M=(Q,∑,q0,A,δ) be an NFA, the string xε∑* is accepted by M, if

$$\delta^*(q0,x) \cap A \neq \emptyset$$

3.4 LANGUAGE ACCEPTANCE BY A NFA:

The language recognized or accepted by M is the set L(M) of all strings accepted by M.

$$L(M)=\{x|\ \delta^*(q0,x) \cap A \neq \emptyset\}$$

3.5 Recursive definition of δ* for an NFA:

Let M=(Q,∑,q0,A,δ) be an NFA, the function $\delta^*:Q \times \sum^* \rightarrow 2^Q$ is defined as follows.

i) For any qεQ, δ*(q,Λ)={q}

ii) For any qεQ, yε∑* and aε∑,

$$\delta^*(q,ya)=(\bigcup_{r\varepsilon\delta^*(q,y)}^{\delta(r,a)})$$

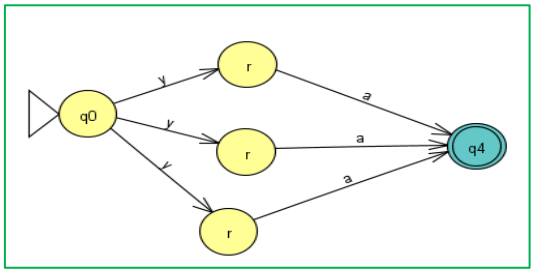


Figure 1.17 State Diagram

Problems:

M=(Q,∑,q0,A,δ)

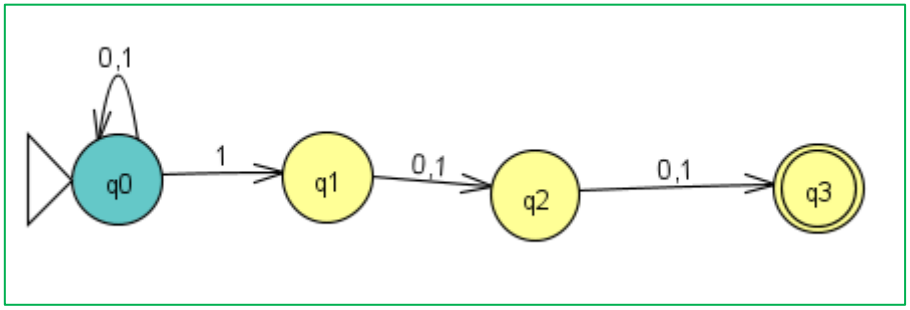


Figure 1.18 State Diagram

Determine:

i) **δ*(q0,11)=** $(\bigcup_{r\varepsilon\delta*(q,1)}^{\delta(r,1)})$

$= \bigcup_{r\varepsilon\{q0,q1\}}^{\delta(r,1)})$

= δ(q0,1) U δ(q1,1)

={q0,q1} U {q2}

={q0,q1,q2}

∴11 is not accepted.

ii) **δ*(q0,01)=** $(\bigcup_{r\varepsilon\delta*(q,0)}^{\delta(r,1)})$

$= \bigcup_{r\varepsilon\{q0\}}^{\delta(r,1)})$

= δ(q0,1)

={q0,q1}

∴01 is not accepted.

iii) **δ*(q0,111)=** $(\bigcup_{r\varepsilon\delta*(q,11)}^{\delta(r,1)})$

$= \bigcup_{r\varepsilon\{q0,q1,q2\}}^{\delta(r,1)})$

= δ(q0,1) U δ(q1,1) U δ(q2,1)

={q0,q1} U{q2}U{q3}

={q0,q1,q2,q3}

∴111 is accepted.

iv) **δ*(q0,011)=** $(\bigcup_{r\varepsilon\delta*(q,01)}^{\delta(r,1)})$

$= \bigcup_{r\varepsilon\{q0,q1\}}^{\delta(r,1)})$

= δ(q0,1) U δ(q1,1)

={q0,q1} U{q2}

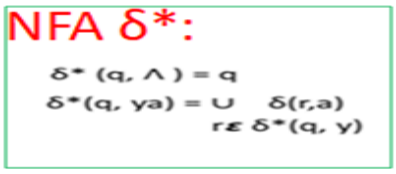
={q0,q1,q2}

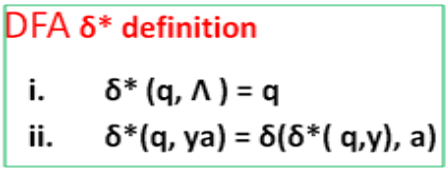
∴11 is not accepted.

3.6 THEOREM 1:

Statement:

For any NFA machine M=(Q, ∑, q0, δ , A) accepting a language L ⊆ ∑*, there is a DFA machine M1= (Q, ∑, q0, δ , A) that accepts L.





Proof by Structural Induction:

Proof:

DFA- $M_1$is defined as follows:

- $Q_1=2^Q$
- $q_1=q_0$
- $A_1=\{q\epsilon Q_1 \mid A\subseteq Q\}$
- For,$q\epsilon Q_1$ and $a\epsilon\sum$,

$$\delta_1(q,a)=\bigcup_{r\epsilon q}\delta(r,a) \quad \text{-------1}$$

To prove, $\delta_1^*(q_1,x)= \delta^*(q_0,x)$

M_1accepts the same language as M.

Proof by Structural Induction:

1.Basis Step

Strings of length 0

If x=Λ,

LHS = $\delta_1^*(q_1,x) = \delta_1^*(q_1,\Lambda)$

= q_1------- (By definition of δ_1^*)

= $\{q_0\}$ ------- (By definition of q_1)

= $\delta^*(q_0,\Lambda)$------- (By definition of δ^*)

= $\delta^*(q_0,x)$ = RHS

2.Induction Hypothesis:

Assume the statement to be proved is true.

x is a string satisfying,

$$\delta_1^*(q_1,x)= \delta^*(q_0,x)$$

3.Induction Step:

To prove, for any $a\epsilon\sum$, $x\epsilon\sum^*$

$\delta_1^*(q_1,xa)= \delta^*(q_0,xa)$ to be proved

LHS=$\delta_1^*(q_1,xa)=\delta_1(\delta_1^*(q_1,x),a)$ ------- (By definition of δ_1^*)

= $\delta_1(\delta^*(q_0,x),a)$ -------(By Induction Hypothesis)

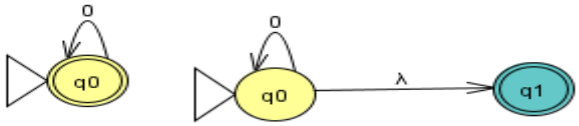
= $\bigcup_{r\varepsilon\delta*\{q0,x\}}^{\delta(r,a)}$) ------(By 1)

=$\delta^*(q_0,xa)$ =RHS------- (By definition of δ^*)

Hence Proved.

4.NFA with Λ transition:

- This allows transitions not only input symbols but also on null inputs.
- Λ or ε
- aΛ=a | oΛ=0 |1Λ = 1



o*

4.1 Definition:NFA-Λ

NFA-Λ is a 5 tuple machine M=(Q,∑,q0,A,δ), where

Q - is a set of finite states,

∑ - finite set of input symbols

q0εQ,

A ⊆ Q and

δ:Qx(∑U{Λ})->2^Q

4.2 Epsilon (NULL) Closure of a state -Λ(q):

Λ(q) is a set of states can be defined as follows:

- It is a set of all states that can be reached from any state on Λ symbol
- Let M=(Q, ∑, q0, δ , A) be a NFA- Λ machine and let S be any subset of Q.
- The Λ(S) is the set defined as follows:
  1. Every element of S is in element of Λ(S)
  2. For any q ***ε*** Λ(S), every element of δ(q, Λ) is in Λ(S)

4.3 Problems on Λ Closure:

1.

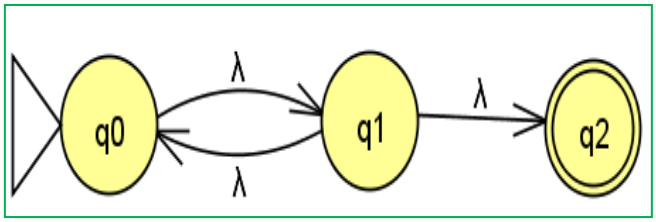


Figure 1.19 State Diagram

Λ(q0)={q0, q1, q2}

Λ(q1) ={q1, q0, q2}

Λ(q2)={q2}

2.

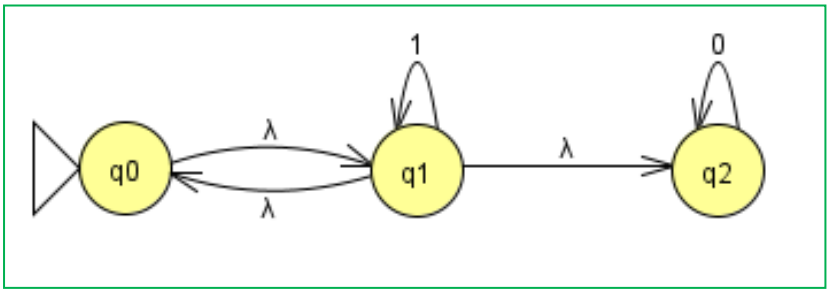


Figure 1.20 State Diagram

Λ(q0)={q0, q1, q2}

Λ(q1) ={q0, q1, q2}

Λ(q2)={q2}

4.4. Extended Transition Function of NFA- Λ (δ*):

- δ*
  - Describes what happens when we start in any state and follow any sequence of inputs.
  - Definition of δ*:

Let M= (Q,Σ,δ,q0,A) be a NFA with Λ .

We define the function δ* : Q x ∑ * {Λ }→2^Q as follows:

1. For any state q ***ε*** Q , δ* (q, Λ) = Λ (q)
2. for any state q ***ε*** Q , y ***ε***∑ *, a ***ε*** ∑

$$\delta^*(q, ya) = \Lambda(\bigcup_{r \varepsilon \delta^*(q,y)}^{\delta(r,a)})$$

Problems on δ* for NFA-Λ

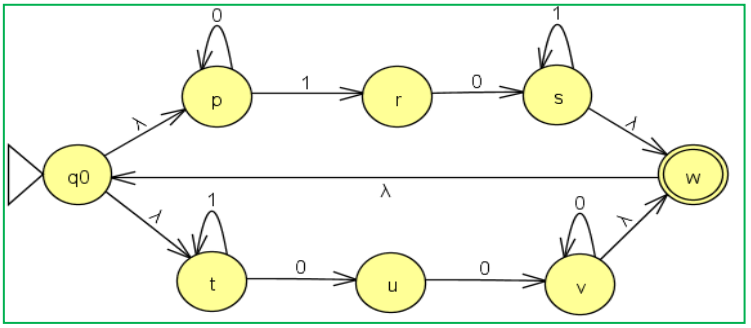


Figure 1.21 State Diagram

(i) Find δ*(q0, Λ)

δ*(q0, Λ) = Λ({q0})

={q0,p,t}

(ii) Find δ*(q0, 0)

$\delta^*(q0, \Lambda 0) = \Lambda(\bigcup_{r \varepsilon \delta^*(q0,\Lambda)}^{\delta(r,0)})$

$= \Lambda(\bigcup_{r \varepsilon \{q0,p,t\}}^{\delta(r,0)})$

= Λ(δ(q0,0) U δ(p,0)U δ(t,0))

= Λ(∅ U{P}U{u})

= Λ({p,u})

={p,u}

String not accepted.

(iii) δ*(q0, 01)

$\delta^*(q0, 01) = \Lambda(\bigcup_{r \varepsilon \delta^*(q0,0)}^{\delta(r,1)})$

$= \Lambda(\bigcup_{r \varepsilon \{p,u\}}^{\delta(r,1)})$

= Λ(δ(p,1) U δ(u,1))

= Λ({r}U∅)

= Λ({r})

={r}

String not accepted.

(iv) δ*(q0, 010)

$\delta^*(q0, 010) = \Lambda(\bigcup_{r \varepsilon \delta^*(q0,01)}^{\delta(r,0)})$

$= \Lambda(\bigcup_{r \varepsilon \{r\}}^{\delta(r,0)})$

= Λ(δ(r,0))

= Λ({s})

={s,u,q0,p,t}

String is accepted.

4.5 Conversion of NFA-Λ to an NFA:

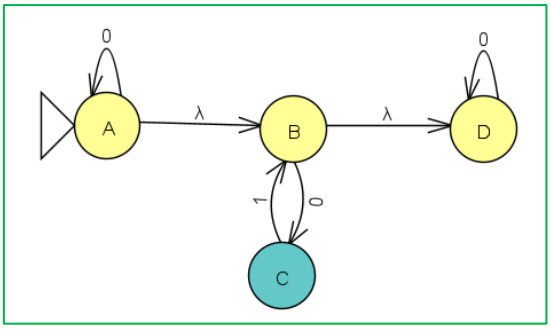


Figure 1.21 State Diagram

Λ-closure(A)={A,B,D}

Λ-closure(B)={B,D}

Λ-closure(C)={C}

Λ-closure(D)={D}

(i) **δ*(A,0)= δ*(A,Λ0)**

$=\Lambda(\cup_{r\varepsilon\delta^*(A,\Lambda)}^{\delta(r,0)})$

$=\Lambda(\cup_{r\,\varepsilon\,\Lambda(A)}^{\delta(r,0)})$

$=\Lambda(\cup_{r\varepsilon\,\{A,B,D\}}^{\delta(r,0)})$

=Λ(δ(A,0) U δ(B,0) U δ(D,0))

= Λ(A,C,D)

={A,B,C,D}

(ii) **δ*(A,1)= δ*(A,Λ1)**

$=\Lambda(\cup_{r\varepsilon\delta^*(A,\Lambda)}^{\delta(r,1)})$

$=\Lambda(\cup_{r\,\varepsilon\,\Lambda(A)}^{\delta(r,1)})$

$=\Lambda(\cup_{r\varepsilon\,\{A,B,D\}}^{\delta(r,1)})$

=Λ(δ(A,1) U δ(B,1) U δ(D,1))

= Λ(∅)

=∅

(iii) **δ*(B,0)= δ*(B,Λ0)**

$=\Lambda(\cup_{r\varepsilon\delta^*(B,\Lambda)}^{\delta(r,0)})$

$=\Lambda(\cup_{r\,\varepsilon\,\Lambda(B)}^{\delta(r,0)})$

$=\Lambda(\cup_{r\varepsilon\,\{B,D\}}^{\delta(r,0)})$

=Λ(δ(B,0) U δ(D,0))

= Λ(C,D)

$=\{C,D\}$

(iv) $\boldsymbol{\delta^*(B,1)= \delta^*(B,\Lambda 1)}$

$=\Lambda(\bigcup_{r\varepsilon\delta^*(B,\Lambda)}^{\delta(r,1)})$

$=\Lambda(\bigcup_{r\,\varepsilon\,\Lambda(B)}^{\delta(r,1)})$

$=\Lambda(\bigcup_{r\varepsilon\,\{B,D\}}^{\delta(r,1)})$

$=\Lambda(\delta(B,1)\cup\delta(D,1))$

$=\Lambda(\emptyset)$

$=\emptyset$

(v) $\boldsymbol{\delta^*(C,0)= \delta^*(C,\Lambda 0)}$

$=\Lambda(\bigcup_{r\varepsilon\delta^*(C,\Lambda)}^{\delta(r,0)})$

$=\Lambda(\bigcup_{r\,\varepsilon\,\Lambda(C)}^{\delta(r,0)})$

$=\Lambda(\bigcup_{r\varepsilon\,\{C\}}^{\delta(r,0)})$

$=\Lambda(\delta(C,0))$

$=\Lambda(\emptyset)$

$=\emptyset$

(vi) $\boldsymbol{\delta^*(C,1)= \delta^*(C,\Lambda 1)}$

$=\Lambda(\bigcup_{r\varepsilon\delta^*(C,\Lambda)}^{\delta(r,1)})$

$=\Lambda(\bigcup_{r\,\varepsilon\,\Lambda(C)}^{\delta(r,1)})$

$=\Lambda(\bigcup_{r\varepsilon\,\{C\}}^{\delta(r,1)})$

$=\Lambda(\delta(C,1))$

$=\Lambda(B)$

$=\{B,D\}$

(vii) $\boldsymbol{\delta^*(D,0)= \delta^*(D,\Lambda 0)}$

$=\Lambda(\bigcup_{r\varepsilon\delta^*(D,\Lambda)}^{\delta(r,0)})$

$=\Lambda(\bigcup_{r\,\varepsilon\,\Lambda(D)}^{\delta(r,0)})$

$=\Lambda(\bigcup_{r\varepsilon\,\{D\}}^{\delta(r,0)})$

$=\Lambda(\delta(D,0))$

$=\Lambda(D)$

$=\{D\}$

(viii) **δ*(D,1)= δ*(D,Λ1)**

$=\Lambda(\cup^{\delta(r,1)}_{r\varepsilon\delta^*(D,\Lambda)})$

$=\Lambda(\cup^{\delta(r,1)}_{r\,\varepsilon\,\Lambda(D)})$

$=\Lambda(\cup^{\delta(r,1)}_{r\varepsilon\,\{D\}})$

$=\Lambda(\delta(D,1))$

$=\Lambda(\emptyset)$

$=\emptyset$

Transition Table:

NFA- Λ				NFA	
STATES	∑			**δ*(q,0)**	**δ*(q,1)**
	Λ	0	1		
A	{B}	{A}	**∅**	{A,B,C,D}	**∅**
B	{D}	{C}	∅	{C,D}	∅
C	∅	∅	{B}	∅	{B,D}
D	∅	{D}	∅	{D}	∅

NFA:

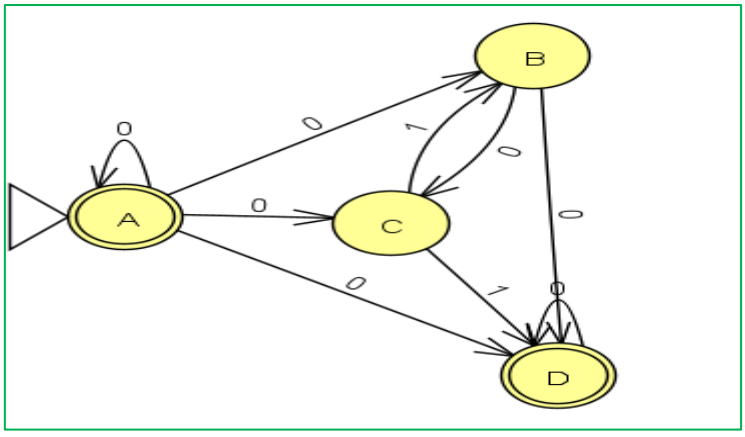


Figure 1.22 State Diagram

Note: Final state of NFA and NFA- Λ are same, moreover if initial state Λ(A)={A,B,D} has NFA-Λ final state then initial state A is also final state.

4.6.Theorem 2

- Equivalence of NFA-Λ and NFA

- If L is accepted by an NFA with Λ transitions then L is accepted by NFA without Λ transitions
- Statement:

  For any NFA NULL machine M=(Q, ∑, q0, δ , A) accepting a language L ⊆ ∑*, there is a NFA machine M1= (Q, ∑, q0, δ , A) that accepts L.

Given:

NFA-Λ, M=(Q,∑,δ,q0,A)

NFA, M1=(Q1,∑,δ1,q1,A1)

Proof:

- **By Definitions:**

$$A1=\begin{cases} A \cup \{q0\}, & if\ \Lambda(\{q0\})\ contains\ a\ state\ of\ final\ state \\ A, & otherwise \end{cases}$$

- **For, a∈∑ :**

  $\delta 1(q1,a) = \Lambda(\bigcup_{r \in \delta^*(C,\Lambda)}^{\delta(r,1)})$ **(By definition it is true if we pass a single length string)**

To Prove: $\delta_1^*(q1,x) = \delta^*(q0,x)$

Proof: By Structural Induction

Basis Step:

Take |x|=0 => x= Λ

LHS: $\delta_1^*(q1, \Lambda) = \{q1\}$

RHS: : $\delta^*(q0, \Lambda) = \Lambda \{q\}$

So the above statement is not true for |x|=Λ

Hence we begin the induction with |x|=1

Let, x=a

$\delta_1^*(q1, a) = \delta^*(q0, \Lambda)$ **(By definition it is true if we pass a single length string)**

Induction Hypothesis: Assumption:

x is a string satisfying**:** $\delta_1^*(q_1,x) = \delta^*(q_0,x)$

Induction Step

- To prove, for any a∈∑, x ∈∑*

$\delta_1^*(q_1,xa)= \delta^*(q_0,xa)$

LHS: $\delta_1^*(q_1,xa)$

$=\bigcup_{r\varepsilon}^{\delta(r,a)} \delta 1*\{q1,x\})$ ----(By Induction Hypothesis)

$= \bigcup_{r\varepsilon}^{\delta 1(r,a)} \delta*\{q0,x\})$ δ=δ1 if the length of the string

$=\bigcup_{r\varepsilon}^{\delta(r,a)} \delta*\{q0,x\})$

$=\bigcup_{r\varepsilon}^{\delta(r,a)} \Lambda(\bigcup_{s\varepsilon}^{\delta(s,\Lambda)} \delta*\{q0,x\})$

$= \Lambda(\bigcup_{s\varepsilon}^{\delta(s,a)} \delta*\{q0,x\})$

= δ*(q, xa)

=RHS Hence the theorem is proved

5.Kleen's Theorem Part-I:

Theorem Statement:

Any RL can be accepted by a FA.

Or

Let 'r' be a RE, then there exists an NFA with Λ transitions that accepts L(r).

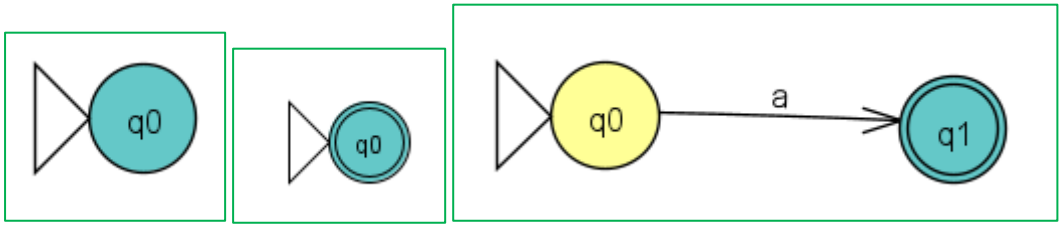
Proof:

By induction on the number of operators in the RE 'r' that there is an NFA 'M' with Λ-transitions having one final state, and no transitions out of this final state, such that L(M)=L(r).

Basis Step: 'r' has Ø operators

'r' must be Ø, Λ or a where a∈∑.

The NFA for 'r' are



r=Φ r=Λ r='a'

Induction Hypothesis:

Assume that theorem is true for 'r' with fewer than 'i' operators, i>=1=>1<=n<i

Induction Step:

Let 'r' have 'i' operators.

Case 1: r=r1+r2

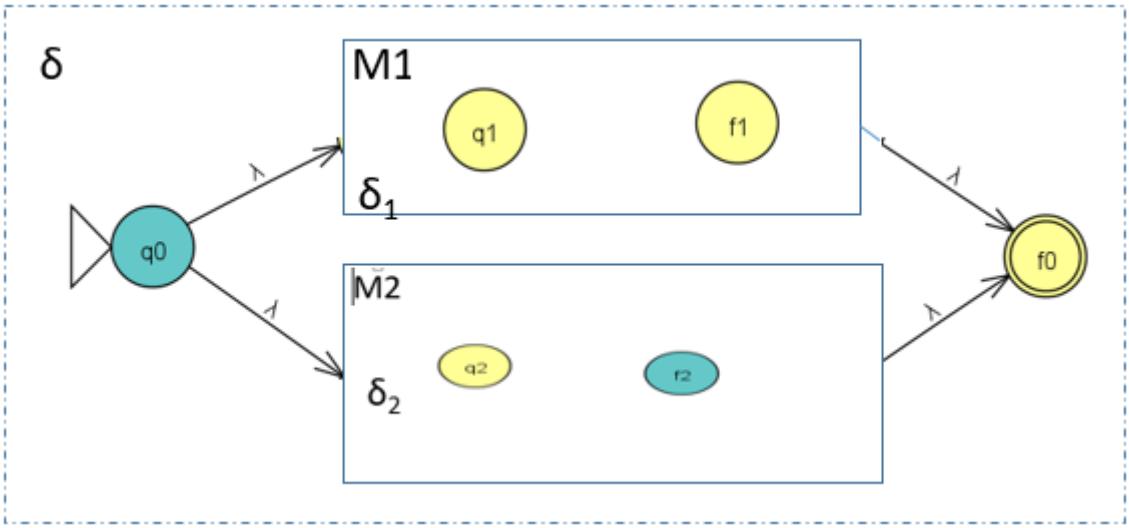


Figure 5.1 State Diagram

Let NFA's machine is $M_1=(Q_1,\sum_1,\delta_1,q_1,\{f1\})$ & $L(M_1)=L(r_1)$

Let DFA's machine is $M_2=(Q_2,\sum_2,\delta_2,q_2,\{f2\})$ & $L(M_2)=L(r_2)$

Where Q1 and Q2 are disjoint.

q0->new initial state

f0->new final state

Construct $M= (Q1UQ2U\{q0,f0\}, \sum_1U\sum_2,\delta,q0,\{f0\})$

Where δ is defined by,

i) $\delta(q0,\Lambda)=\{q1,q2\}$

ii) $\delta(q,a)=\delta_1(q,a)$ for q in $Q_1-\{f1\}$ & a in $\sum_1U\{\Lambda\}$

iii) $\delta(q,a)=\delta_2(q,a)$ for q in $Q_2-\{f2\}$ & a in $\sum_2U\{\Lambda\}$

iv) $\delta(f1,\Lambda)=\delta(f2,\Lambda)=\{f0\}$

Thus all the moves of M1 and M2 are in M.

There is a path labelled x in M from q1 to f1 or a path in M2 from q2 to f2.

Hence L(M)=L(M1) U L(M2)

L(M)={x|x is in L(M1) or x is in L(M2)}

Case 2: r=r1 . r2

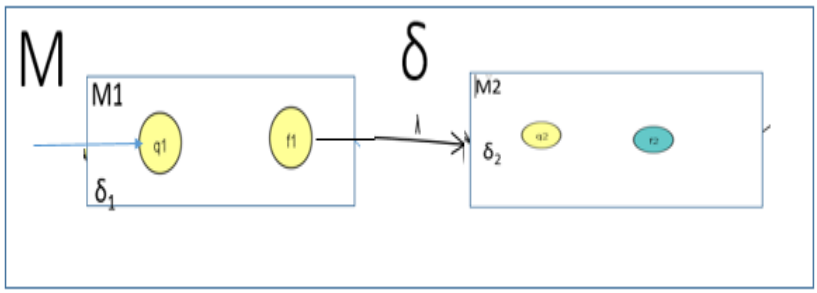


Figure 5.2 State Diagram

Let M1 and M2 be as in case2.

Construct M=(Q1UQ2, ∑1U∑2, δ, {q1}, {f1})

Where δ is defined by,

i) $\delta(q,a)=\delta_1(q,a)$ for q in Q_1-{f1} & a in $\sum_1 U\{\Lambda\}$

ii) $\delta(f1,\Lambda)=\{q2\}$

iii) $\delta(q,a)=\delta_2(q,a)$ for q in Q_2-{f2} & a in $\sum_2 U\{\Lambda\}$

Every path in M from q1 to f2 is a path labelled by some string x from q1to f1 followed by the edge from f1 to q2 labelled Λ followed by a path labelled by some string 'y' from q2 to f2.

Thus L(M)=L(M1) . L(M2)

Case 3:

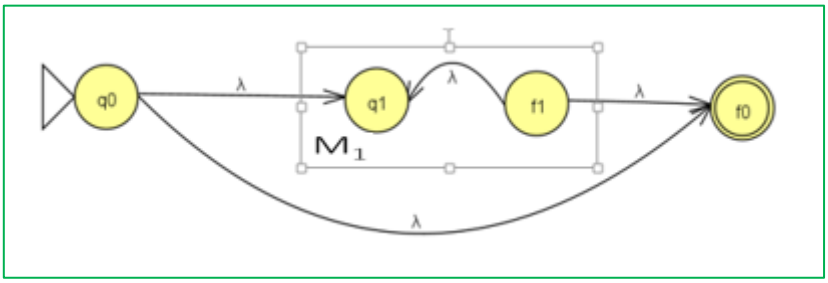


Figure 5.3 State Diagram

Let $M_1=(Q_1,\sum_1,\delta_1,q_1,\{f1\})$ & $L(M_1)=L(r_1)$

Construct M= $(Q1U\{q0,f0\}, \sum_1,\delta_1,q0,\{f0\})$

Where δ is defined by

i) $\delta(q0,\Lambda)=\delta(f1,\Lambda) = \{q1,f0\}$

ii) $\delta(q,a)=\delta_1(q,a)$ for q in Q_1-{f1} & a in $\sum_1 U\{\Lambda\}$

Any path from q0 to f0 consists either of a path from q0 to f0 on Λ or a path from q0 to q1 on Λ, followed by some number of paths from q1 to f1, then back to q1 on Λ, each labelled by a string in L(M1) followed by a path from q1 to f1 on a string in L(M1).

Hence L(M)= L(M1)*

Regular Expressions to NFA-Λ Kleen's Theorem Part-I:

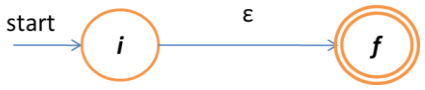
For each kind of RE, define an NFA-Λ

Input: A Regular Expression *r* over an alphabet ∑

Output: An *NFA*-Λ accepting *L(r)*

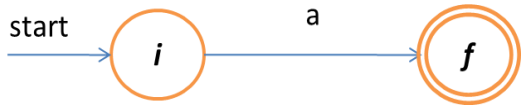
Method:

Step 1: For ε



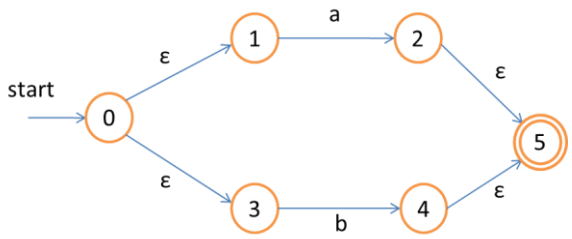
The NFA-Λ recognizes {ε}

Step 2: For a in ∑



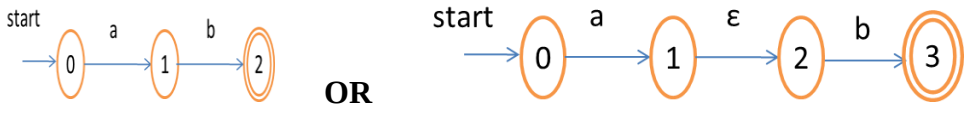
The NFA-Λ recognizes {a}

Step 3: RE=a |b



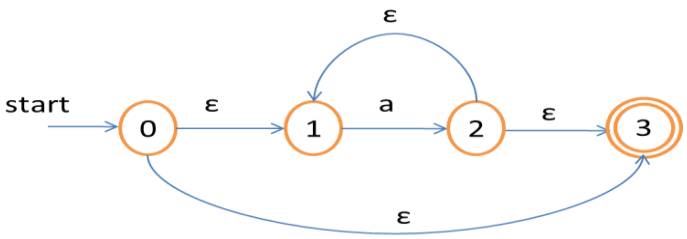
The NFA-Λ recognizes {a,b}

Step 4: RE=ab



The NFA-Λ recognizes {ab}

Step 5: RE=a*



The NFA-Λ recognizes {ε,a,aa,aaa,.....}

Step 6: RE= a^+

= a.a*

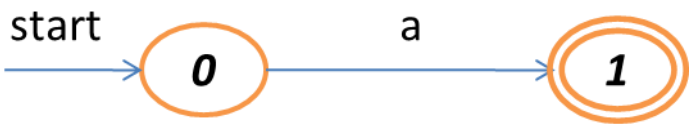
Follow step 4 for construction.

Problems:

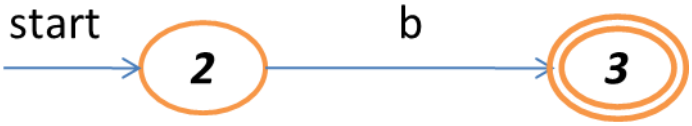
Construct NFA-Λ for the following regular expression using Thompson's Construction method.

a. (a|b)*abb

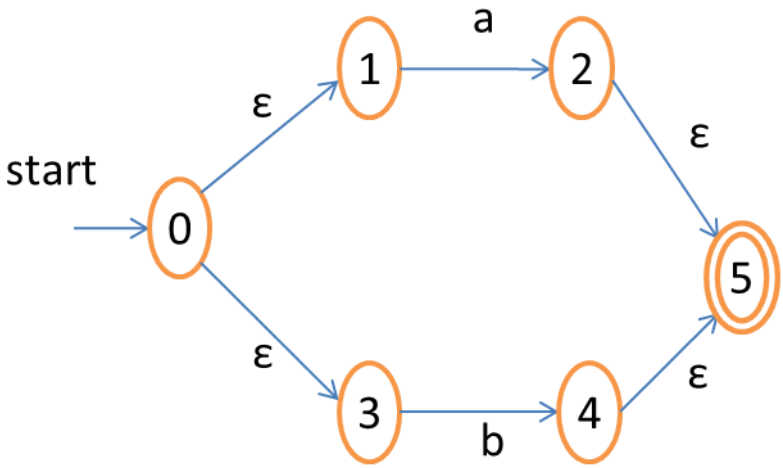
a

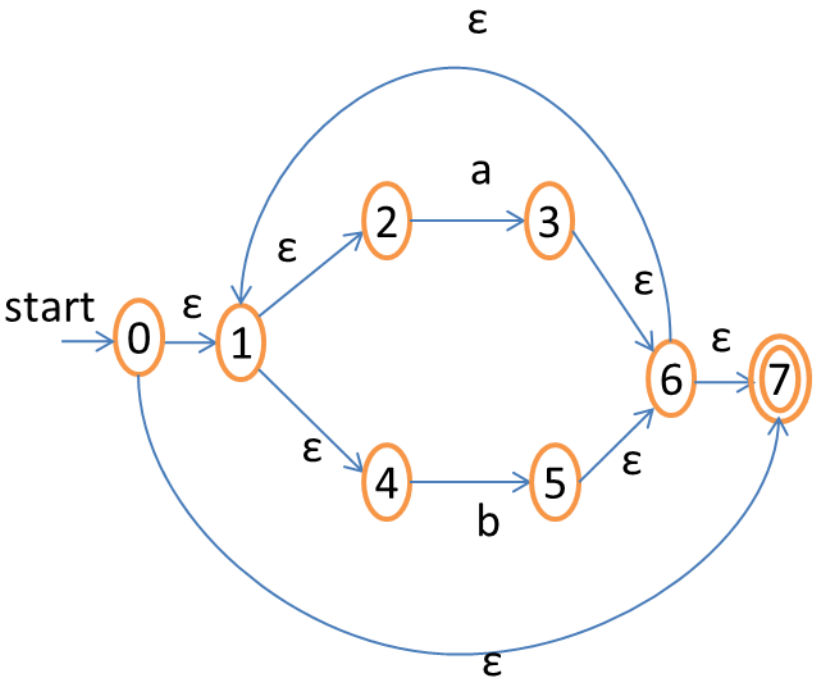


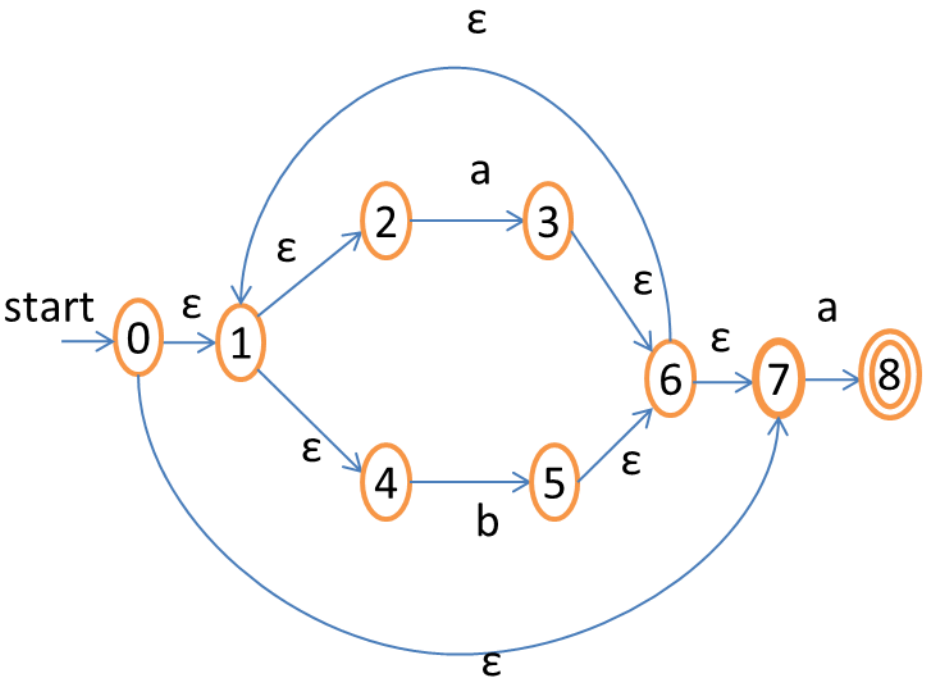
b



a/b




ε
a
2
3
ε
ε
start
ε
0
1
6
ε
7
ε
ε
4
5
ε
b
ε


ε
a
2
3
ε
ε
start
ε
0
1
6
ε
a
7
8
ε
ε
4
5
ε
b
ε

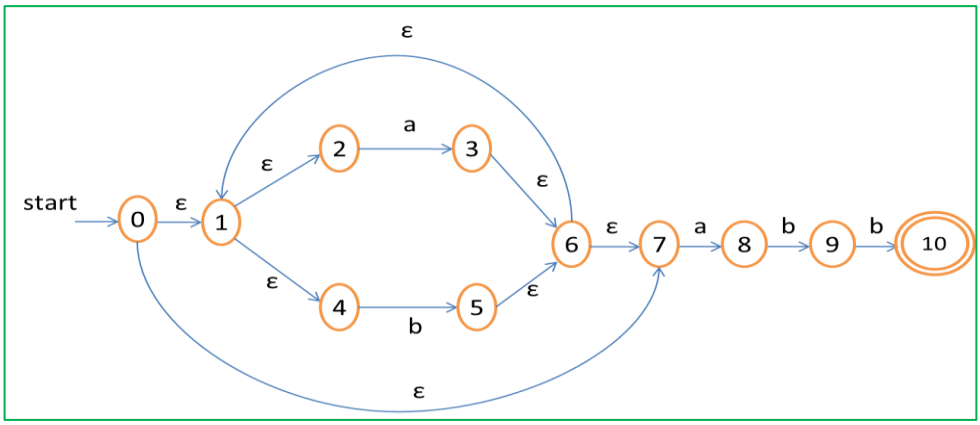


Figure 5.4 State Diagram of NFA-NULL

6.Kleene's Theorem Part -II

- Any language accepted by a finite automaton is regular.

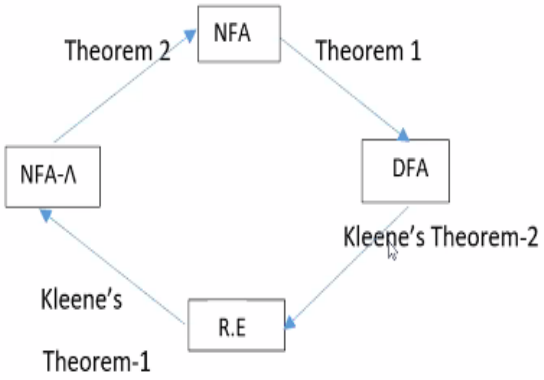


Figure 6.1. Finite Automata Conversion

6.1Conversion of DFA to Regular Expression

- Formula:

$$R_{ij}^{k} = R_{ij}^{(k-1)} + R_{ik}^{(k-1)} (R_{kk}^{k-1})^{*} R_{kj}^{(k-1)}$$

- Where
  - i= Start state
  - j = Final state
  - k = No.of states

Identity rules for Regular Expression

1. Φ+R=R
2. Φ.R=Φ
3. ƐR=RƐ=R
4. R+R=R
5. RR*=R*R=R⁺
6. Ɛ+R⁺=R*
7. Ɛ*=Ɛ
8. Φ*=Ɛ
9. (R*)*=R*
10. (PQ)*.P=p(QP)*
11. (P+Q)R=PR+QR
12. R(P+Q)=RP+RQ
13. (P+Q)*=(P*Q*)*=(P*+Q*)*
14. Ɛ+RR*=Ɛ+R*R=R*

Problem 1: Obtain the regular expression for the finite automata shown below:

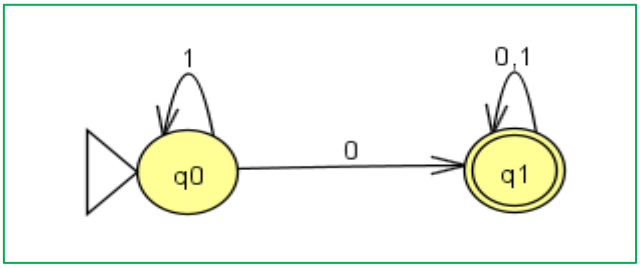


Figure 6.1 DFA State Diagram

Formula for RE:

$$R_{ij}^{k} = R_{ij}^{(k-1)} + R_{ik}^{(k-1)}(R_{kk}^{k-1})^{*}R_{kj}^{(k-1)}$$

Step 1: Rename the states:

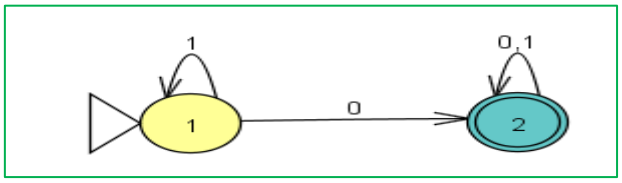


Figure 6.2 DFA State Diagram

Step 2: Find the values of i,j,k

i=1 (start state)

j=2 (final state)

k=2 (no. of states)

Substitute I,j,k in formula:

Find:

$$R_{12}^{2} = R_{12}^{(1)} + R_{12}^{(1)}(R_{22}^{1})^{*}R_{22}^{(1)} \longrightarrow 1$$

$\boldsymbol{R_{12}^{(1)}}$ **=> i=1, j=2, k=1**

$$R_{12}^{1} = R_{12}^{(0)} + R_{11}^{(0)}(R_{11}^{0})^{*}R_{12}^{(0)} \longrightarrow 2$$

$\boldsymbol{R_{22}^{1}}$**=> i=2, j=2, k=1**

$$R_{22}^{1} = R_{22}^{(0)} + R_{21}^{(0)}(R_{11}^{0})^{*}R_{12}^{(0)} \longrightarrow 3$$

In:

$$R_{12}^{1} = R_{12}^{(0)} + R_{11}^{(0)}(R_{11}^{0})^{*}R_{12}^{(0)} \longrightarrow 2$$

$$R_{22}^{1} = R_{22}^{(0)} + R_{21}^{(0)}(R_{11}^{0})^{*}R_{12}^{(0)} \longrightarrow 3$$

When K=0:

$R_{11}^{(0)} = 1+\Lambda$

$R_{12}^{(0)} = 0$

$R_{21}^{(0)} = \Phi$

$R_{22}^{(0)} = 0+1+\Lambda$

Substitute in equation 2 and 3:

$$2 \Rightarrow R_{12}^{1} = R_{12}^{(0)} + R_{11}^{(0)} (R_{11}^{0})^{*} R_{12}^{(0)}$$

$$R_{12}^{1} = 0 + (1+\Lambda)(1+\Lambda)^{*}0$$

$$= 0 + (1+\Lambda)^{+}0$$

$$= \Lambda 0 + 1^{*}0 \qquad = (\Lambda + 1^{*})0 \qquad = 1^{*}0$$

Substitute in equation 3:

$$R_{22}^{1} = R_{22}^{(0)} + R_{21}^{(0)} (R_{11}^{0})^{*} R_{12}^{(0)} \longrightarrow 3$$

$$3 \Rightarrow R_{22}^{1} = R_{22}^{(0)} + R_{21}^{(0)} (R_{11}^{0})^{*} R_{12}^{(0)}$$

When K=0:

$$= (0+1+\Lambda) + \Phi(1+\Lambda)^{*}0$$

$$= (0+1+\Lambda)$$

Substituting in 1:

$$R_{12}^{2} = R_{12}^{(1)} + R_{12}^{(1)} (R_{22}^{1})^{*} R_{22}^{(1)} \longrightarrow 1$$

$$= 1^{*}0 + 1^{*}0(0+1+\Lambda)^{*}(0+1+\Lambda)$$

$$= 1^{*}0 + 1^{*}0(0+1+\Lambda)^{+}$$

$=1*0\Lambda+1*0(0+1)^*$

$=1*0(\Lambda+((0+1)^*)$

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

SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE
www.sathyabama.ac.in

SCHOOL OF COMPUTING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

UNIT – II – Theory of Computation – SCSA1302

CONTEXT FREE LANGUAGES AND NORMAL FORMS

Context-free grammars -More examples -Union, concatenations, and *'s of CFLs-Derivation trees and ambiguity -Unambiguous CFG for algebraic expressions-Normal Forms – CNF – GNF

1. Context Free Grammar

Definition − A context-free grammar (CFG) consisting of a finite set of grammar rules is a quadruple

G= **(N, T, P, S)**

Where,

- **N** is a set of non-terminal symbols (N is also represented as V- the set of variables).
- **T** is a set of terminals where **N ∩ T = NULL.**
- **P** is a set of rules, **P: N → (N ∪ T)***, i.e., the left-hand side of the production rule **P** does have any right context or left context.
- **S** is the start symbol.

Terminals:

- Defines the basic symbols from which a string in a language are composed.
- Represented in lower case letters.

Non Terminals :

- They are special symbols that are described recursively in terms of each other and terminals. Represented in upper case letters.

Production rules:

- It Defines the way in which NTs may be built from one another and from terminal.

Start Symbol:

- It is a special NT from which all the other strings are derived. It is always present in the first rule on the LHS.

Example:

Consider the set of productions

<exp> →<exp>+<exp>
<exp>→<exp>*<exp>
<exp>→(<exp>)
<exp>→id

We apply productions repeatedly (id+id)*id

<exp> ⇒ <exp>*<exp>
⇒ (<exp>)*<exp>
⇒ (<exp>+<exp>)*<exp>
⇒ (id+id) * id

Where (id+id) is the word in the language of <exp>

- Use E instead of

<exp> Productions:

E→ E+E
E→
E*E
E→
(E)
E→id
G=({E},{+,*,id,(,)},P,E)

Definition 2: The language generated by a CFG

Let G=(N,T,P,S) be a CFG. The language generated by G is

$$L(G) = \{x \in \Sigma^* | s \Rightarrow G^* x\}$$

Definition 3:

A language L is a CFL if there is a CFG 'G' so that L = L(G)

1.1 Problems:

1. **Given a CFG, find the language generated by G**

(i) G=(N,T,P,S) where N={S}, T={a,b}
P={S→aSb, S→ab}

S=>ab | S=>aSb =>aaSbb =>aaabbb =>aaaabbbb | S=>aSb =>aabb | S=>aSb =>aaSbb =>aaaSbbb

W= { ab, aabb,aaabbb,aaaabbbb,……..}

$L(G)=\{a^n b^n | n \geq 1\}$

(ii) G=(P,{Ɛ,0,1},P,P}
P: P→0 | 1 | Ɛ |0P0 |1P1
P=>0
P=>1
P=>Ɛ
P=>0P0 | P=>0P0 P=>0P0 | P =>0P0

=>010 =>00 =>000 =>00P00

=>00100

P =>1P1 | P =>0P0 |

=>10P01 | =>00P00 |

=>10101 | =>001P100 |

=>0010100 |

W={Λ, 0,1,00,010,000, 00100, 0010100, 10101,….}

L(G)={language of palindromes}

1.2 CFG Corresponding To A Language

1.2.3 For the given L(G), design a CFG.

i.Language consisting of even number of 1's

T={1,Ɛ }

W={Ɛ ,11,1111,111111,….}

P:

S -> Ɛ

S -> 1S1

G=({S},{1,Ɛ },P,S)

ii. Design a CFG for a language consisting of arithmetic expression.

T={id,+,-,*,/,(,)}

W={id,id+id,id-id,id*id,id/id, id+id*id,(id-id)/id,……}

P: S -> id

S -> S+S

S -> S-S

S ->S*S

S ->S/S

(or)

S->id |S+S |S-S|S*S |S/S |(S)

G=({S},{+,-,*,/,id},P,S)

iii. Design a CFG for a language accepting balanced parenthesis

T={{,},[,],(,)}

W=(Ɛ ,{},[],(), {()},([]),{{}},(()),[[()]],….,{}[]{},()(),{}(),….}

P: S->{ S} |[S] |(S) |SS | Ɛ

G=({S},{ Ɛ ,[,],{,},(,)},P,S)

iv. L={$a^n b^m$| $m > n$ and $n \geq 0$}

T={a,b}
W={b,bb,bbb,…,abb,abbb,…,aabbb,aaabbbb,…}

n=0, m=1=> b
n=0, m=2=>b, bb, bbb,bbbb,….
n=1, m=2 =>abb,abbb,abbbb,…
n=2, m=3=>aabbb,aaabbbb,….
P:
B->b |bB
S->aSb |B
G=({S,B}, {a,b} , P,S)

v. L={w|wϵ{a,b}*, $n_a(w) = n_b(w)$}

W={ε ,ab,ba,aabb,abab,baba,abba,aaabbb,bbaa,baab,….}
P: S-> ε |aSb |bSa | SS
G=({S}, {a,b, ε } , P,S)

vi. L={w|wϵ{a,b}*, $n_a(w) \neq n_b(w)$}

The problem is split into 2 cases:

(i) $n_a(w) > n_b(w)$}
(ii) $n_a(w) < n_b(w)$}

L1= $n_a(w) > n_b(w)$

W={a, aba, aab ,baa,aaabb,aaa,aa,aaaab,baaa,…..}

P1:

A->a| aA |AbA |AAb |bAA

G1=({A},{a,b},P1,A)

L2= $n_b(w) > n_a(w)$

W={b,bb,bbb, abb,bab,…….}

P2:

B->b| bB |BaB |BBa |aBB

G2=({B},{a,b},P2,B}

L=L1∪ L2 $n_a(w) \neq n_b(w)$

P:

S->A|B

A->a| aA |AbA |AAb |bAA

B->b| bB |BaB |BBa |aBB

G=({S,A,B},{a,b},P, S}

vii. Construct the CFG for the language having any number of a's followed by any number of b's over the set ∑= {a}

W={ ε ,aaaa,bbbb,aabb,abbb,…..}
a*.b*
x=aabb
P:
S->A.B
A->aA|ε
B->bB| ε

G=({S,A,B},{a,b, ε },P,S)

1.3 Regular Expression to CFG

i. Find the CFG equivalent to a Regular Expression

RE=ab. (a+bb)*

Generate the production for the language L1={ab}

A->ab

Generate the production for the language L2=a+bb

B->a | bb

Generate the production for the language L2*=(a+bb)*

C->ε |BC

RE=ab. (a+bb)*

P: S->A.C

C->ε |BC

A->ab

B->a | bb

G=({S,A,B,C}, {a,b, ε },P, }

ii. Obtain the CFG for RE=(011+1)*(01)*

L1=(011+1)
L2=>(011+1)*
Generate the production for the language L1=(011+1)

A->011|1

Generate the production for the language L2=(011+1)*
B->AB|ε

A->011|1

Similarly derive for (01)*

C->DC|ε

D->01

Finally generate the concatenation of the 2 languages by adding the production

S->BC

B->AB|ε

A->011|1

C->DC|ε

D->01

Definition 4:

Regular Grammar
A Grammar G=(N,T,P,S) is regular if every production takes one of the following forms:
B→aC
B→a
Where B & C are NT and 'a' is a T

2.Derivation Trees and Ambiguity

2.1 Derivation: Process of deriving a string using the grammar

- **Types :**
  - Left Most Derivation (LMD)
  - Right Most Derivation (RMD)

2.2 Derivation tree is a graphical representation for the derivation. Also called as Parse Tree.
Properties:

- The root node is always a node indicating start symbols.
- Every vertex has a label which is in (N U T U Λ)
- The leaf node has a label from T (terminal).
- The interior nodes are always the non-terminal nodes.
- If a vertex is labeled A & if x1,x2,x3,…xn are all children of A from left then A->x1x2x3….xn be a production in P.

Leftmost derivation & Rightmost Derivation

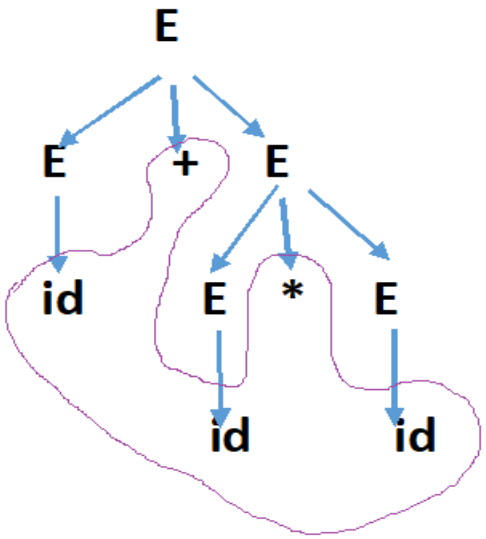
2.3 Leftmost derivation

In the derivation process, if the leftmost variable is replaced at every step then the derivation is leftmost derivation.

E->E+E | E*E | (E) |id

String:id+id*id

$E \overset{lmd}{\Longrightarrow}$ **E+E**	**E->E+E**
$\overset{lmd}{\Longrightarrow}$ **id+E**	**E->id**
$\overset{lmd}{\Longrightarrow}$ **id+E*E**	**E->E*E**
$\overset{lmd}{\Longrightarrow}$ **id+id*E**	**E->id**
$\overset{lmd}{\Longrightarrow}$ **id+id*id**	**E->id**



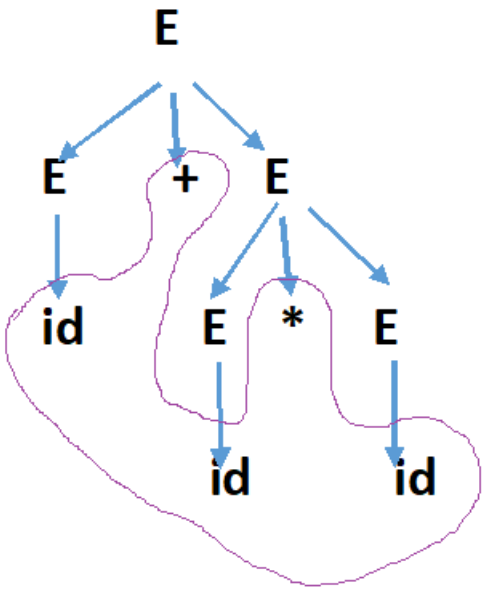
2.4 Rightmost Derivation

In the derivation process, if the rightmost variable is replaced at every step then the derivation is rightmost derivation.

E->E+E | E*E | (E) |id

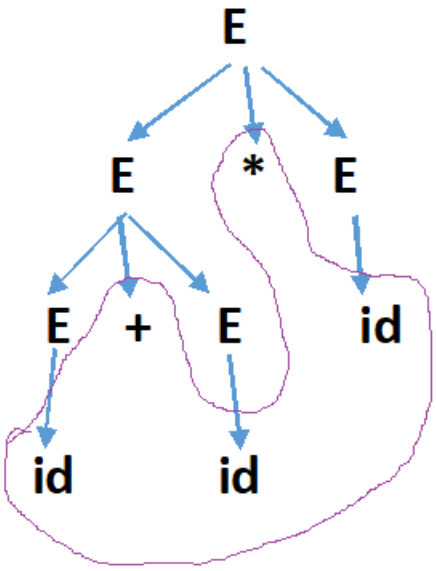
String:id+id*id

$E \overset{rmd}{\Longrightarrow} E+E$	E->E+E
$\overset{rmd}{\Longrightarrow} E+E*E$	E->E*E
$\overset{rmd}{\Longrightarrow} E+E*id$	E->id
$\overset{rmd}{\Longrightarrow} E+id*id$	E->id
$\overset{rmd}{\Longrightarrow} id+id*id$	E=>id



Definition: Yield of a tree:

Is the string of symbols obtained by only reading the leaves of the tree from left to right without considering the □ symbols called sentinal function.



Yield of the tree=id+id*id

2.5 Problems:

1. **For the Grammar G defined by**
   S->AB
   B->a|Sb
   A->Aa|bB
   Give the derivation trees for the following sentential forms

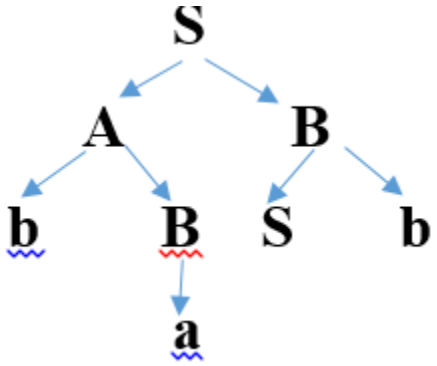
(i) baSb

S=>AB | S->AB

=>bBB | A->bB

=>baB | B->a

=>baSb | B->Sb



(ii) bAaBbB

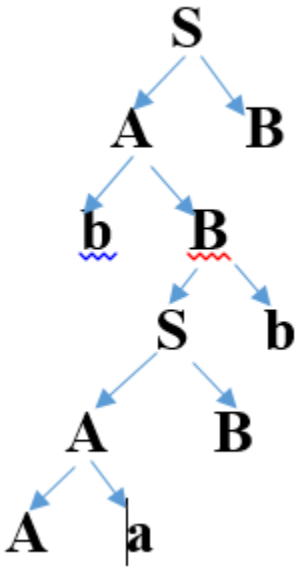
S=>AB **S->AB**

=>bBB **A->bB**

=>bSbB **B->Sb**

=>bABbB **S->AB**

=>bAaBbB **A->Aa**



3. Ambiguity

Definition: An Ambiguous CFG

A CFG G is ambiguous if there is atleat one string in L(G) having two or more distinct derivation trees(or equivalently two or more distinct LMD).

i.Is the following grammar ambiguous:

E->E+E | E*E | (E) |id

Consider the String: id+id*id

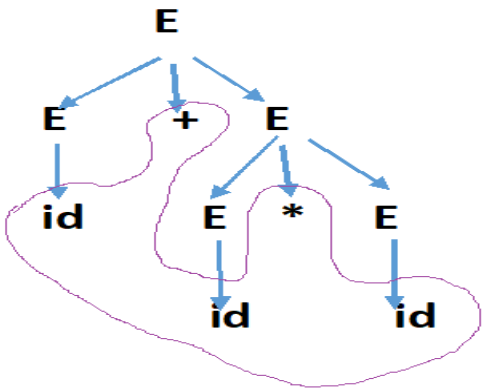
$E \overset{lmd}{\Longrightarrow}$ E+E | E->E+E

$\overset{lmd}{\Longrightarrow}$ id+E |E->id

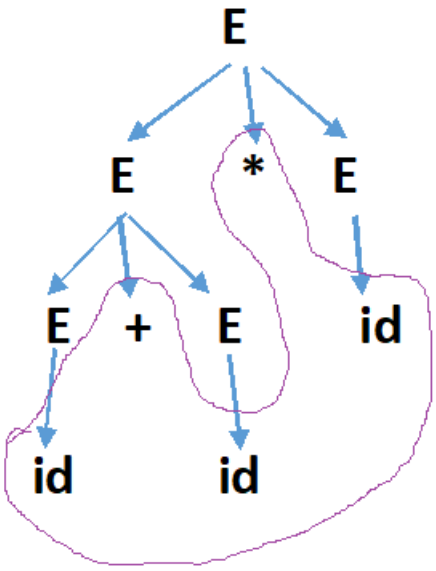
$\overset{lmd}{\Longrightarrow}$ id+E*E |E->E*E

$\overset{lmd}{\Longrightarrow}$ id+id*E |E->id

$\overset{lmd}{\Longrightarrow}$ id+id*id |E->id



$E \overset{lmd}{\Longrightarrow}$ E*E	E-> E*E
$\overset{lmd}{\Longrightarrow}$ E+E*E	E->E+E
$\overset{lmd}{\Longrightarrow}$ id+E*E	E->id
$\overset{lmd}{\Longrightarrow}$ id+id*E	E->id
$\overset{lmd}{\Longrightarrow}$ id+id*id	E->id



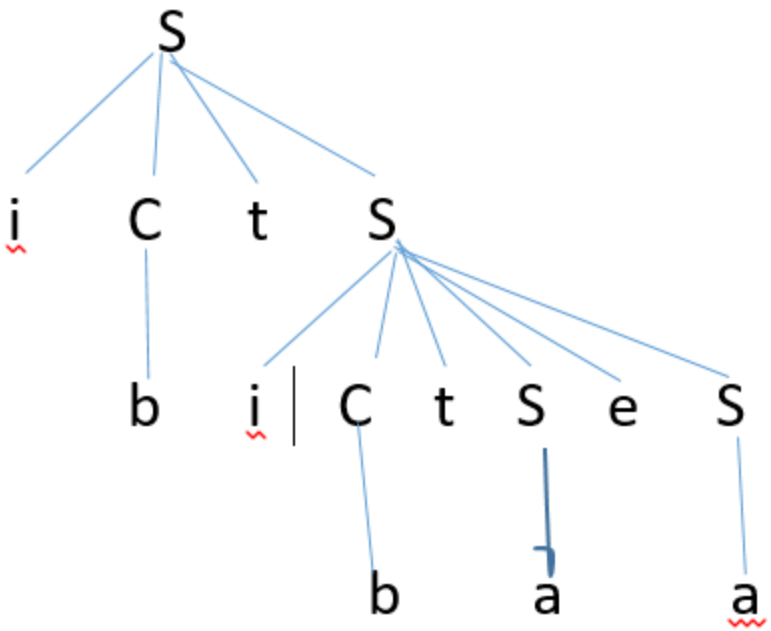
There are 2 parse trees or 2 leftmost derivations for the string ‘id+id*id’. So the given grammar is ambiguous.

ii.Is the following grammar ambiguous:

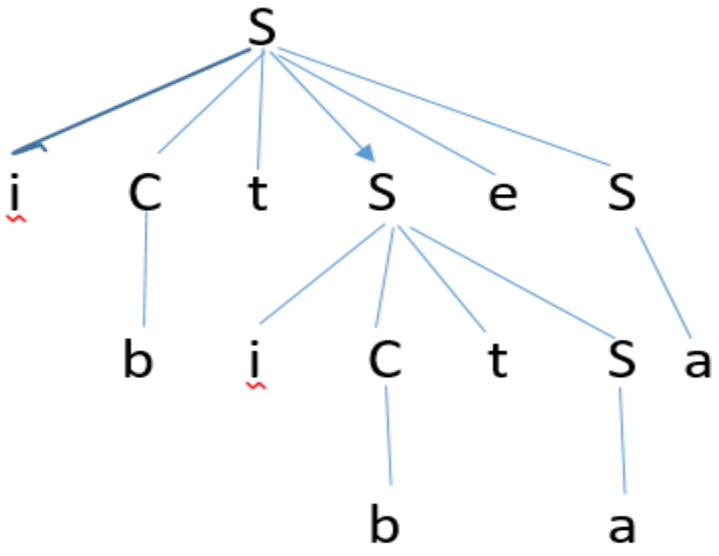
S->iCtS |iCtSeS |a
C->b

Consider the String: **ibtibtaea**

S $\overset{lmd}{\Longrightarrow}$ iCtS	S->iCtS
$\overset{lmd}{\Longrightarrow}$ ibtS	C->b
$\overset{lmd}{\Longrightarrow}$ ibt iCtSeS	S-> iCtSeS
$\overset{lmd}{\Longrightarrow}$ ibtibtSeS	C->b
$\overset{lmd}{\Longrightarrow}$ ibtibtaeS	S->a
$\overset{lmd}{\Longrightarrow}$ ibtibtaea	S->a



S$\overset{lmd}{\Longrightarrow}$ iCtSeS
$\overset{lmd}{\Longrightarrow}$ibtSeS
$\overset{lmd}{\Longrightarrow}$ibtiCtSeS
$\overset{lmd}{\Longrightarrow}$ibtibtSeS
$\overset{lmd}{\Longrightarrow}$ibtibtaeS
$\overset{lmd}{\Longrightarrow}$ibtibtaea

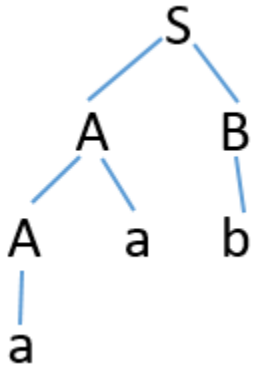


There are 2 parse trees or 2 leftmost derivations for the string ‘ibtibtaea’. So the given grammar is ambiguous.

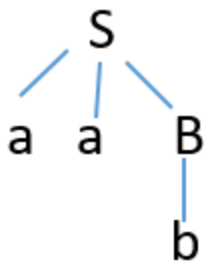
iii. Is the following grammar ambiguous:

S->AB | aaB
A->a | Aa
B->b
String: aab

$S \overset{lmd}{\Longrightarrow}$ AB	S->AB
$\overset{lmd}{\Longrightarrow}$ AaB	A->Aa
$\overset{lmd}{\Longrightarrow}$ aaB	A->a
$\overset{lmd}{\Longrightarrow}$ aab	B->b



$S \overset{lmd}{\Longrightarrow}$ aaB	S->aaB
$\overset{lmd}{\Longrightarrow}$ aab	B->b



There are 2 parse trees or 2 leftmost derivations for the string 'aab'. So the given grammar is ambiguous.

An unambiguous CFG for Algebraic expression

If a CFG is ambiguous , it is often possible and usually desirable to find an equivalent unambiguous CFG.

4. Normal Forms:

Chomsky Normal Form(CNF)

Greibach Normal Form(GNF)

4.1Chomsky Normal Form(CNF):

A CFG is in CNF if every production is one of two types

A->BC

A->a

Where A,B and C are Non terminals and 'a' is a terminal

4.1.1.Converting a CFG to CNF

i.Let G be the CFG with productions

S->AACD
A->aAb | Λ
C->aC |a
D ->aDa | bDb |Λ

Step 1:

Eliminating Λ productions

Any production A for which P contains the production A->Λ is nullable

Nullable variable: A->Λ D->Λ

S->AACD | ACD |CD | AAC |AC |C

A->aAb | ab

C->aC |a

D ->aDa | bDb |aa |bb

Step 2:

Eliminating unit productions S->C

S->AACD | ACD |CD | AAC |AC | aC |a

A->aAb | ab

C->aC |a

D ->aDa | bDb |aa |bb

Step 3:Restricting the right sides of the productions to single terminals or strings of two or more variables(NON TERMINALS).

S->AACD | ACD |CD | AAC |AC | X_a C |a

A-> X_a A X_b | X_a X_b

C-> X_a C |a

D -> X_aD X_a | X_b D X_b | X_a X_a | X_b X_b

X_a->a

X_b->b

Step 4:

S->AT_1 |AT_2 |CD |AT_3 |AC | X_a C |a

T_1->AT_2

T_2->CD

T_3->AC

A-> $X_a$$T_4$ | X_a X_b

T_4-> A X_b

C-> X_a C |a

D->X_aT5 | $X_b$$T_6$| X_a X_a | X_b X_b

T_5->D X_a

T_6-> D X_b

X_a->a

X_b->b

4.3 Greibach Normal Form (GNF)

Let G=(N, T, P, S) be a CFG. The CFG 'G' is said to be in GNF, if all the production are of the form:

A->aα

Where a Ɛ T and α Ɛ N*

A non-terminal generating a terminal which is followed by any number of non-terminals.

For example, A → a.

S → aASB.

Step 1: Convert the grammar into CNF.

Step 2: Rename the non-terminals to A1, A2, A3,…

Step 3: In the grammar, convert the given production rule into GNF form.

Problem 1:

S-> AB1 |0S | ε

A->00A | B

B-> 1A1

Step 1: Eliminate null productions.

S-> ε

S->AB1 | 0S | 0

A->00A |B

B->1A1

Step 2: Eliminate unit productions

A->B is the unit production.

S->AB1 |0S |0

A->00A |1A1

B->1A1

Step 3: Restricting right hand side production with single terminal symbol or two or more non terminals.

X->0

Y->1

S->ABY | XS |0

A->XXA |YAY

B->YAY

Step 4: Final CNF

X->0 Y->1

S->AT1 T1->BY

S->XS | 0

A->XT2 T2->XA A->YT3 T3->AY

B->YT3

Step 5: Rename Non terminal as A1, A2, A3,……..

S=A1, A= A2, B=A3, X=A4, Y=A5, T1=A6, T2=A7, T3=A8
X->0 Y->1
S->AT1 T1->BY
S->XS | 0
A->XT2 T2->XA A->YT3 T3->AY
B->YT3

A4->0 A5->1
A1->A2A6 | A4A1 | 0
A2->A4A7 | A5A8
A3->A5A8
A6->A3A5
A7->A4A2
A8->A2A5

Step 6: Obtain productions to the form A->aα

Final CFG is

A4->0 A5->1

A2->0A7 |1A8

A3->1A8

A7->0A2

A8->0A7A5 | 1A8A5

A6->1A8A5

A1->0A7A6 | 1A8A6 | 0A1 |0

SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE
www.sathyabama.ac.in

SCHOOL OF COMPUTING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

UNIT – III – THEORY OF COMPUTATION – SCSA1302

PUSH DOWN AUTOMATA (PDA)

Pushdown automata - Introduction - Definition - Deterministic pushdown automata - PDA corresponding to a given context-free grammar – Context-free Grammar corresponding to PDA - Pumping Lemma for CFG

1. Push Down Automata

1.1. Drawback of Finite Automata

- can remember only a finite amount of information
- No memory is used in FA

2. Introduction

- PDA can remember an infinite amount of information.
- Memory used – Stack
- A PDA is more powerful than FA.
- Any language which can be acceptable by FA can also be acceptable by PDA.
- PDA also accepts a class of languages which cannot be accepted by FA.
- PDA recognizes CFL.

FA + stack = PDA

3. Definition

The PDA can be defined as a collection of 7 tuples:

$$M=(Q, \sum, \Gamma, q_0, Z_0, F, \delta)$$

Q: the finite set of states

$\sum$: the input set

Γ: a stack symbol which can be pushed and popped from the stack q_0:

the initial state

Z_0: a start symbol which is in Γ.

F: a set of final states.

δ: Transition function which is used for moving from current state to next state.

δ: Q x $\{\Sigma \cup \varepsilon\}$ x Γ -> Q x Γ^*

(i.e) $\delta(q,a,x)=(p,\alpha)$

from state ‘q’ for an input symbol ‘a’, and stack symbol ‘x’, goto state ‘p’ and x is replaced by string ‘α’.

3.1. Instantaneous Description (ID)

- An instantaneous description is a triple ID

  (q, w, α)

Where:

- **Q** describes the current state.
- **w** describes the remaining input.
- ***α*** describes the stack contents, top at the left.

Example Derivation: (p, b, T) ⊢ (q, w, α)

3.2. Definition: Acceptance by a PDA

1. Acceptance by Final State:
   If M =(Q, ∑, Γ, δ, q0, Z, F) is a PDA and the language L(M) accepted by the final state is given by: x∈∑* and x is accepted by M if,
   $L(M) = \{x \mid (q0, x, Z_0) \vdash^* (q, \Lambda, \alpha)\}$
   Where q∈Aα∈Γ*

2. Acceptance by Empty Stack:
   For each PDA, M=(Q, ∑, Γ, δ, q0, Z, F) the language accepted by empty stack is given by
   $L(M) = \{x \mid (q0, x, Z_0) \vdash^* (q, \Lambda, \Lambda)\}$
   For any state q∈A and x∈∑*

Language Acceptance:
A language L ⊆∑* is said to be accepted by M, if L is precisely the set of string accepted by M.
L=L(M)

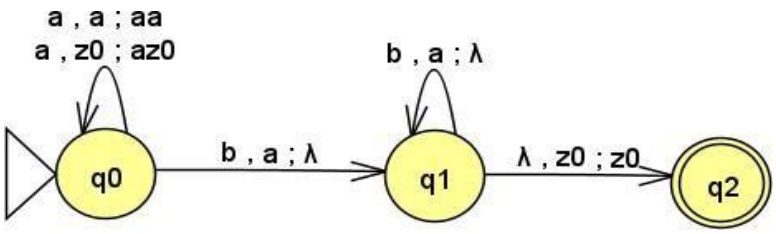
4. Construction of PDA

1. Design a PDA for accepting a language $\{a^n b^n \mid n>=1\}$.

Solution:

1. Decide the nature of the language b’s followed by ‘a’.
2. Execution procedure using stack:
   - Push all a’s on to stack.
   - For every ‘b’ pop out an ‘a’.
3. Define the states
   - q0 – push all a’s on to the stack.
   - q1 – when a ‘b’ encounters, pop ‘a’ from stack.
   - q2 – accepting state.

PDA Diagram



Transition Table:

Move No.	**State**	**Input Symbol**	**Top of stack**	**Moves**
1	q_0	a	Z0	(q_0,az_0)
2	q_0	a	a	(q_0,aa)
3	q_0	b	a	(q_1,Λ)
4	q_1	b	a	(q_1,Λ)
5	q_1	Λ	Z0	(q_2,z_0)
All other combinations		None		

Trace the moves: a^3b^3 => aaabbb

Move No.	Resulting state	Input	Stack
-	q_0	aaabbb	Z_0
1	q_0	aabbb	az_0
2	q_0	abbb	aaz_0
2	q_0	bbb	$aaaz_0$
3	q_1	bb	aaz_0
4	q_1	b	az_0
4	q_1	Λ	Z_0
5	q_2	Λ	Z_0
Accepted			

Trace the moves: a^2b=> aab

Move No.	Resulting state	Input	Stack
-	q_0	aab	Z_0
1	q_0	ab	az_0
2	q_0	b	aaz_0
3	q_1	Λ	az_0
Rejected			

Instantaneous Description (ID)

$(q_0, aabb,z_0)$ |- (q_0, abb, az_0)

|- (q_0, bb, aaz_0)

|-(q_1, b, az_0)

|-(q_1,Λ, z_0)

|-(q_2, Λ, z_0)

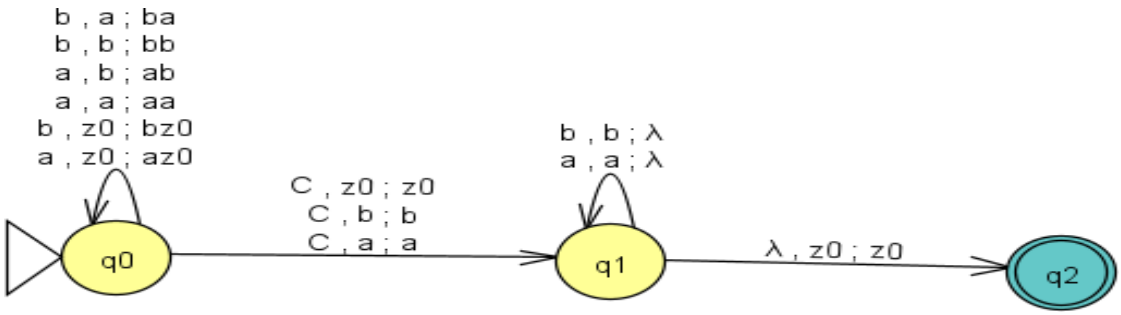
String Accepted

2. Construct PDA for the language L={xCx^r | x Ɛ {a,b}*}

Solution

1. Nature of the language:
   A string 'x' followed by constant 'C' followed by reversed string.
2. Execution procedure:
   - Push the string 'x' into the stack until 'C' is encountered.
   - When 'C' is encountered, don't do any operation.
   - After that, pop out each element from the stack for the string x^r.
3. Define the states:
   - q0 - Push all input symbols until 'C'.
   - q1 – when 'C' is encountered, Pop.
   - q2 – accepting state.

PDA



Transition Table

Move No.	State	Input Symbol	Top of stack	Moves
1	q_0	a	Z_0	(q_0,az_0)
2	q_0	b	Z_0	(q0,bz_0)
3	q_0	a	a	(q_0,aa)
4	q_0	a	b	(q_0,ab)
5	q_0	b	b	(q_0,bb)
6	q_0	b	a	(q_0,ba)
7	q_0	C	a	(q_1,a)
8	q_0	C	b	(q_1,b)
9	q_0	C	Z_0	(q_1,z_0)

10	q_1	a	a	(q_1,Λ)
11	q_1	b	b	(q_1, Λ)
12	q_1	Λ	Z_0	(q_2,z_0)
All other combinations - No moves				

Trace the moves: abCba

Move No.	Resulting state	Input	Stack
-	q_0	abCba	Z_0
1	q_0	bCba	az_0
6	q_0	Cba	baz_0
8	q_1	ba	baz_0
11	q_1	a	az_0
10	q_1	Λ	Z_0
12	q_2	Λ	Z_0
Accept			

Trace the moves: abCa

Move No.	Resulting state	Input	Stack
-	q_0	abCa	Z_0
1	q_0	bCa	az_0
6	q_0	Ca	baz_0
8	q_1	a	baz_0
Rejected			

3. Consider the CFG

S->[S] | {S} | Λ

Generate the CFL and PDA.

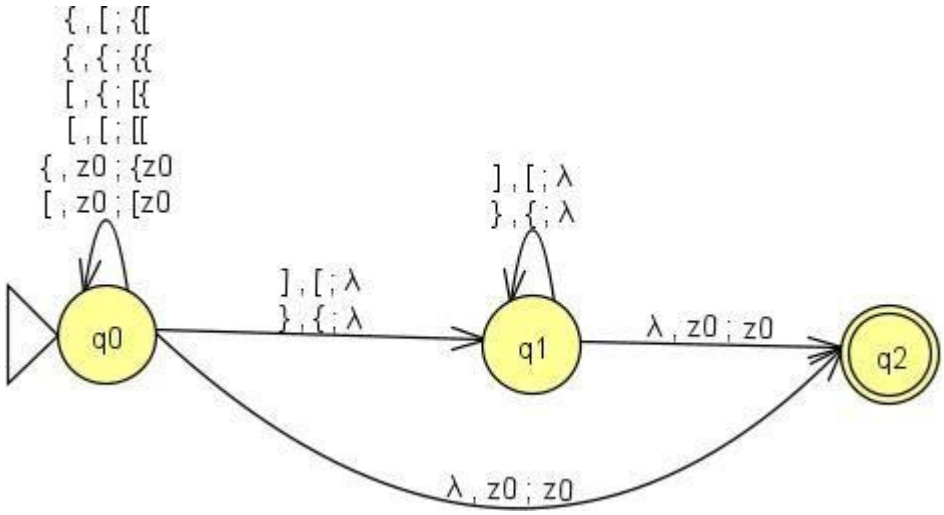
Solution

1. Nature of the language

CFL = {Λ, [], {}, [{}], {[]}, [{{}}],….}

Open parenthesis followed by symbols and then closed parenthesis.

2. Define the states:
   q0 – When all the open parenthesis are encountered.
   q1 – when all the closed parenthesis are encountered
   q2 – accepting state.



Transition Table:

Move No.	State	Input Symbol	Top of stack	Moves
1	q_0	[	Z_0	$(q_0,[z_0)$
2	q_0	{	Z_0	$(q_0,\{z_0)$
3	q_0	[	[	$(q_0,[[)$
4	q_0	[	{	$(q_0,[\{)$
5	q_0	{	{	$(q_0,\{\{)$
6	q_0	{	[	$(q_0,\{[)$
7	q_0	}	{	(q_1, Λ)
8	q_0	]	[	(q_1, Λ)
9	q_1	}	{	(q_1, Λ)
10	q_1	]	[	(q_1, Λ)
11	q_1	Λ	Z_0	(q_2,z_0)
12	q_0	Λ	Z_0	(q_2,z_0)
All other combinations			No moves	

Trace the moves: {[{}]}

Move No.	Resulting state	Input	Stack
-	q_0	{[{}]}	Z_0
2	q_0	[{}]}	$\{z_0$
4	q_0	{}]}	$[\{z_0$
6	q_0	}]}	$\{[\{z_0$
7	q_1	]}	$[\{z_0$
10	q_1	}	$\{z_0$

9	q_1	Λ	Z_0
11	q_2	Λ	Z_0
Accept			

Trace the moves: {[{]

Move No.	Resulting state	Input	Stack
-	q_0	{[{]	Z_0
2	q_0	[{]	{z_0
4	q_0	{]	[{z_0
6	q_0	]	{[{z_0
	No Move		
Rejected			

4.

5. Parsing:

- To derive a string using the production rules for a grammar.
- It is used to check whether or not a string is syntactically correct.
- Parser takes the inputs and builds a parse tree.

5.1. Types of parser

Top down Parser- Parsing starts from the top with the start symbol and derives a string using a parse tree.

Bottom up parser- Starts from the bottom with the string and comes to the start symbol using a parse tree.

5.2. Design of Top down parser

1. Push the start symbol onto the stack.
2. If the top of the stack contains a NT, pop it out of the stack and push its right hand side of the production.
3. If the top of the stack matches with input symbol being read, pop it.
4. If the input string is fully read and the stack is empty go to final state.

5.3. PDA Corresponding to CFG

In 2 ways, PDA can simulate a derivation in the grammar.

- Top Down Parsing - LMD
- Bottom Up Parsing– RMD

5.3.1. Top Down Parsing: PDA corresponding to CFG:

Left Most Derivation is used

Statement:

Let G=(N,T,P,S) be a Context Free G, then there is a push down automata M, so that

L(M)=L(G)

Define 'M' as:

$M=(Q,\sum , \Gamma, \delta, q_0, Z_0, F)$

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

$\Gamma = N \cup \sum \cup \{z_0\}$

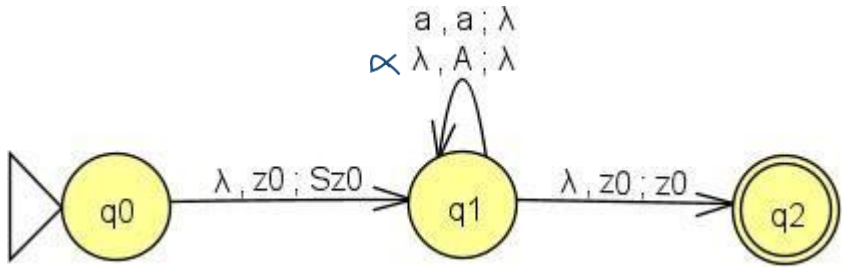
$A = \{q_2\}$

a) $\delta(q_0,\Lambda,z_0)=\{(q_1,Sz_0)\}$

b) for every $A \in N$, $\delta (q_1,\Lambda,A)=\{(q_1, \alpha) \mid A->\alpha$ is a production in G}

c) for every $a \in \sum$, $\delta (q1,a, a)=\{(q, \Lambda)\}$

d) $\delta (q1,\Lambda,z0)=\{(q2,z0)\}$



Q={q0,q1,q2}

A={q2}

5.3.2. Construction of PDA

1. Construct PDA for the language

$L=\{x \in \{a,b\}^* \mid n_a(x) > n_b(x)\}$

S->a | aS | bSS | SSb| SbS

Solution:

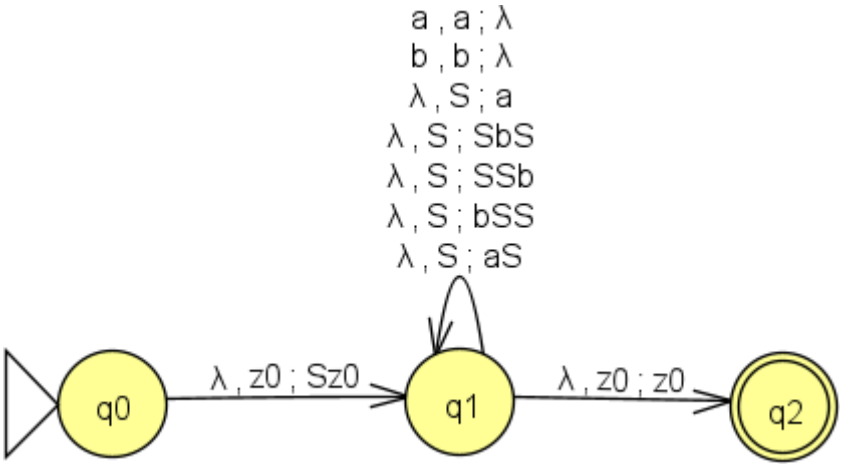
Let M=(Q, ∑, Γ, δ, q0, Z, F)

Q={q0,q1,q2}

∑={a,b}

Γ={S,a,b,z0}

F={q2}



Transition Table:

State	Input	Stack Symbol	Moves
q0	Λ	Z0	(q1,Sz0)
Q1	Λ	S	(q1,a),(q1,aS),(q1,bSS),(q1,SSb),(q1,SbS)
Q1	a	a	(q1, Λ)
Q1	b	b	(q1, Λ)
Q1	Λ	Z0	(q2,z0)
All other combination none			

S->a | aS | bSS | SSb| SbS

Example: abbaaa
S->a | aS | bSS | SSb| SbS

S=>SbS S->SbS
=>abS S->a
=>abbSS S->bSS
=>abbaS S->a
=>abbaaS S->aS
=>abbaaa S->a

Sequence of moves: parsing
(q0, Λ abbaaa,z0) -| (q1, Λ abbaaa ,SZ0)

-| (q1, Λ abbaaa, SbSZ0)
(q1, abbaaa, abSZ0)
(q1, bbaaa, bSz0)
(q1, baaa, Sz0)
(q1, baaa, bSSz0)
(q1, aaa, SSz0)

(q1, aaa, aSz0)
(q1,aa, Sz0)
(q1,aa, aSz0)
(q1.a, Sz0)
(q1,a , az0)
(q1, Λ, z0)
(q2)

2. Construct PDA

S->S+X | X
X->X*Y | Y
Y->(S) | id
String: id+id*id

Solution:

Let M=(Q, ∑, Γ, δ, q0, Z0, F)

Q={q0,q1,q2}
∑={ +,*,id,(,)}
Γ={ z0,S,X,Y, +,*,id,(,)}
A={q2}

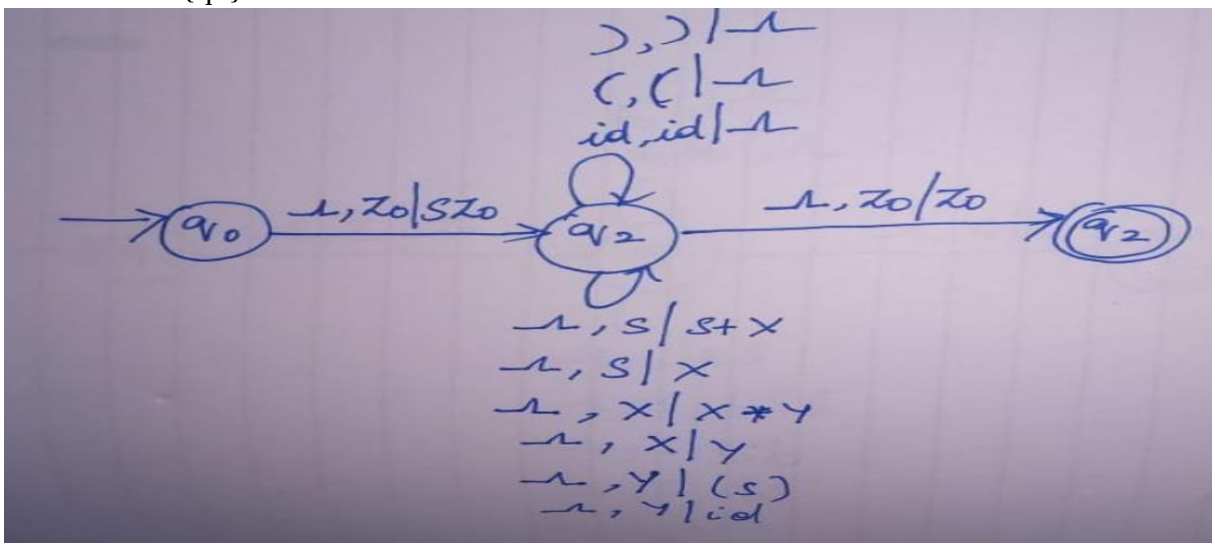


Figure 3.6 State Diagram

String: id*id+id

S=>S+X S->S+X
=>X+X S->X
=>X*X+X X->X*X
=>Y*X+X X->Y
-> id*X+X Y->id
=>id*Y+X X-> Y
=>id*id+X Y->id

=>id*id+Y X->Y
=>id*id+id Y->id
(q_0,id*id+id,z_0) -|(q_0,id*id+id,Sz_0)
(q_1,id*id+id, S+X)
(q_1, id*id+id, X+X)
(q_1, id*id+id, X*X+X)
(q_1, id*id+id, Y*X+X)
(q_1, id*id+id, id*X+X)
(q_1, *id+id, *X+X) (q_1,
id+id, X+X) (q_1,id+id,
Y+X)
(q_1, id+id, id+X)
(q_1, +id, +X)
(q_1, id, X)
(q_1, id, Y)
(q_1, id, id)

5.3.4. Bottom-Up PDA

- Right Most Derivation in reverse is used.

Steps:

- Push the current input symbol onto the stack.
- Replace the right-hand side of a production at the top of the stack with its left-hand side.
- If the top of the stack element matches with the current input symbol, pop it.
- If the input string is fully read and only if the start symbol 'S' remains in the stack, pop it and go to the final state 'F'.

Problem 1:

P: S->S+T
S->T
T->T*a
T->a

String: a+a*a

Right Most Derivation:

S=>S+T [S->S+T]
=>S+T*a [T->T*a]
=>S+a*a [T->a]
=>T+a*a [S->T]
=>a+a*a [T->a]

Move	Production	Stack	Unread Input
-	-	Z_0	a+a*a
shift	-	aZ_0	+a*a
Reduce	T->a	TZ_0	+a*a
Reduce	S->T	SZ_0	+a*a
Shift	-	+SZ_0	a*a
Shift	-	a+SZ_0	*a

Reduce	T->a	$T+SZ_0$	*a
Shift	-	$*T+SZ_0$	A
Shift	-	$a*T+SZ_0$	-
Reduce	T->T*a	$T+SZ_0$	-
Reduce	S->S+T	$S\ Z_0$	-
Accept			

6. Deterministic Push down automata

Let M=(Q,∑ , Γ, δ, q0, Z, F) be a PDA, M is deterministic if there is no configuration for which M has a choice of more than one move.

If M is deterministic it satisfies the following condition:

i) For every q ∈ Q, a ∈ ∑ U { Λ } and x ∈ Γ then the set δ(q, a, x) has at most one element

ii) For any q ∈ Q, x ∈ Γ, if δ(q, Λ, x) ≠ φ then δ(q, a, x) = φ for every a ∈ ∑.

Problem :

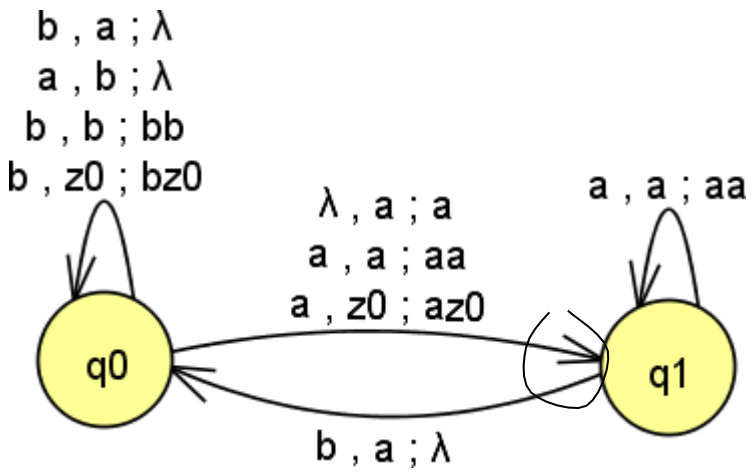
Construct DPDA for the language

$L=\{x\in\{a,b\}^* | n_a(x)>n_b(x)\}$

solution

DPDA with Null transition :

W= {a, aa, aaa, aab, aba,....., baa, bbaaa, aabba, aaab,...}

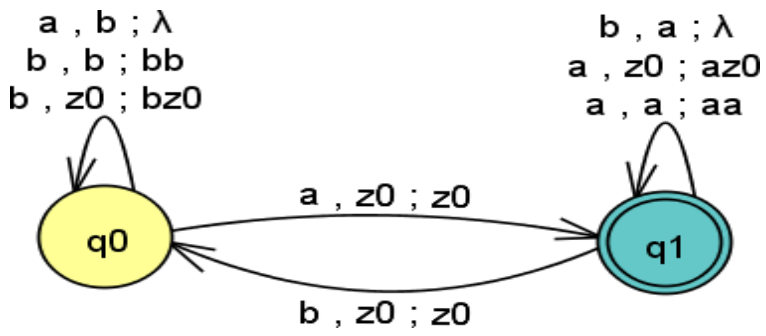


W= {a, aa, aaa, aab, aba,....., baa, bbaaa, aabba, aaab,...}

Move No.	State	Input Symbol	Top of stack	Moves
1	q0	a	Z0	(q1,az0)
2	q0	a	a	(q1,aa)
3	q0	Λ	a	(q1,a)
4	q0	b	Z0	(q0,bz0)
5	q0	b	b	(q0,bb)
6	q0	a	b	(q0, Λ)
7	q0	b	a	(q0, Λ)
8	q1	a	a	(q1,aa)
9	q1	b	a	(q0, Λ)
All other combinations	None			

DPDA without Λ

W= {a, aa, aaa, aab, aba,....., baa, bbaaa, aabba, aaab,...}



Transition Table:

Move No.	State	Input Symbol	Top of stack	Moves
1	q_0	b	Z_0	(q_0,b Z_0)
2	q_0	a	b	(q_0,Λ)
3	q_0	b	b	(q_0,bb)
4	q_0	a	Z_0	(q_1, Z_0)
5	q_1	b	a	(q_1, Λ)

6	q_1	a	Z_0	(q_1,aZ_0)
7	q_1	a	a	(q_1,aa)
8	q_1	b	Z_0	(q_0, Z_0)
All other combinations	None			

Trace the moves: aaaba

Move No.	Resulting state	Input	Stack
-	q0	aaaba	Z0
4	Q1	aaba	Z0
6	Q1	aba	Az0
7	Q1	ba	Aaz0
5	Q1	a	Az0
7	Q1	-	Aaz0
Accept			

7. Pumping Lemma

CFL- We can always find two pieces of any sufficiently long string to pump in tandem .i.e. if we repeat each of the two pieces the same number of times, we get another string of the language.

- Pumping Lemma is used to prove that a language is not CFL.
- It should never be used to show a language is regular.

For any language L, we break its strings into five parts and pump second and fourth substring.

Let 'L' be any CFL. Then there is a constant 'n' depending on L, such that if 'Z' is in L and |z|>=n, then we may write,

Z=uvwxy

uv^iwx^iy Ɛ L, For all i>0,

|vx|>=1

|vwx|<=n

7.1. Procedure

- Assume that L is context free.

- It has to have a pumping length(say n)
- Find a string 'z' in L such that |z]>=n.
- Divide z into uvwxy.
- Show that $uv^iwx^iy \notin L$

Problem 1

$L=\{a^nb^nc^n,\ n>=0\}$ is not a CFL.

Solution:

i. Assume 'L' is a CFL.

ii. Let 'n' be a natural number obtained by using pumping lemma.

iii. Let $z= a^nb^nc^n$ $|z|=n+n+n=3n$

iv. Split z into uvwxy such that |vx|>=1, |vwx|<=n

Assume $z=a^{n-i}a^ib^{n-j-k}b^jb^kc^n$

$u=a^{n-i}$ $v=a^i$ $w=b^{n-j-k}$ $x=b^j$ $y=b^kc^n$

for i=2

$=> uv^2wx^2y =>a^{n-i}a^ia^ib^{n-j-k}b^jb^jb^kc^n$

$=>a^{n+i}b^{n+j}c^n \notin L$

Eg) n=4

$Z=a^4b^4c^4$=>aaaabbbbcccc

u=a v=aa w=abbbbc x=c y=cc

Let i=2

$uv^iwx^iy =>uv^2wx^2y$=>aaaaaabbbbccccc=>$a^6b^4c^5 \notin L$

Therefore the given language is not CFL.

Problem 2:

$L=\{0^p|p$ is prime is not CFL.

Solution:

i. Assume 'L' is a CFL.

ii. Let 'n' be a natural number obtained by using pumping lemma.

iii. Let P be a prime no. such that p>=n

$Z=0^p \epsilon L$ $|z|=p>=n$

iv. Split z into uvwxy such that |vx|>=1, |vwx|<=n

Let $v=0^k$ $x=0^l$ such that k+l>=1 and <=n

Hence |uwy|=p-k-l

If we pump v and x p+1 times

$|uvwxy|=|uwy|+|v^{(p+1)}.x^{(p+1)}|$

$=p-k-l+k(p+1)+l(p+1)$

$=p-k-l+pk+k+pl+l$

$=p+pk+pl$

$=p(k+l+1)$

Which is not prime.

SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE
www.sathyabama.ac.in

SCHOOL OF COMPUTING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

UNIT – IV – THEORY OF COMPUTATION – SCSA1302

TURING MACHINE

Turing machines - Models of computation and the Turing thesis - Definition of TM and TM as language acceptor - Non-deterministic TM and Deterministic TM – Universal TM

1. Introduction

1.1. Need for Turing Machine

Describe an abstract machine™ that is widely accepted as a general model of computation

Model of Computation

EX: $a^n b^n c^n$- this kind of computation PDA needs 2 or 3 stacks

But Turing machine can handle this type of computation using queue (Tape)

Ex: L = {SS | S ∈ {a,b}* }

-Compare the first half of the string to the 2nd half then queue is more appropriate than stack.

- TM is powerful than PDA

- Recognizes all types of languages like RL, CFL, CSL

1.2. Church Turing thesis

-By Alonzo church

"Any algorithmic procedure that can be carried out by a human, a team of humans or a computer- can be carried out by some Turing machine"

1.3. Turing machine proposal

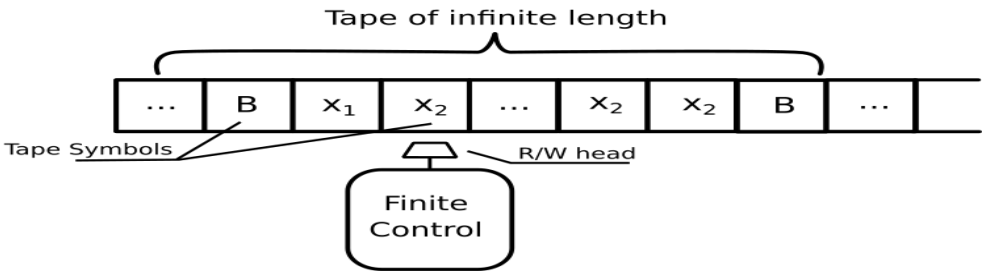


Figure 4.1 Tape

- Tape head is centered on one of the squares of the tape.
- Tape head reads the symbol in the current square(Fig 4.1)
- Moving the tape head one square to the

  Left | Right | Stationary => I | R | S

2. Definition of TM

A TM can be formally described as a 7-tuple abstract machine

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

where –

Q- Finite set of states

Γ – Finite set of allowable tape symbol

∑ - Set of input symbol

B – Symbol of Γ - blank symbol(Δ)

q0 – Start state

F – Final state

$\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R, S\}$

EX: δ (q1, x) = (q2, y, D)

Figure 4.2 Sample Transition

- From state q1 with x, replace X |Y , go to state q1, and move the tape head either D = { L,R, S}(Fig 4.2)

Turing machine can

(i) **Crash:** If in this situation D=L but the tape head is scanning square 0, the leftmost square, the tape head is not allowed to move.

(ii) **Halt:** r=h, the move causes the turing machine to halt.

Turing Machine can be represented by

1. Transition Table
2. Instantaneous Description
3. Transition Diagram

2.1. Instantaneous Description (ID)

Instantaneous Description of a turing machine is given by $\boldsymbol{\alpha_1 q \alpha_2}$

where, q- is the current state of M, $q \epsilon Q$

$\alpha_1, \alpha_2 \epsilon \Gamma^*$ - the contents of the tape upto the rightmost non blank symbol.

Initial ID: $q_0 \alpha_1 \alpha_2$

Final ID: $\alpha_1 \alpha_2 q_B$

Turing Machine can do one of the following things:

(i) Halt and accept by entering into the final state.

(ii) Halt and reject (δ is not defined)

(iii) Turing machine will never halt and enters into an infinite loop.

Definition: Language acceptance by Turing Machine

Let M=**$(Q, \Gamma, \sum, \delta, q_0, B, F)$** be a turing machine. The language L(M) accepted by M is defined as :

$L(M)=\{ w \mid q0w \vdash^* \boldsymbol{\alpha_1 \alpha_2} p \}$

Where, $w \epsilon \sum^*$, $P \epsilon F$, $\boldsymbol{\alpha_1 \alpha_2 \epsilon \Gamma^*}$

The language accepted by the turing machine is REL(Recursively Enumerable Language)

2.2. construction of Turing Machines

1. Obtain TM to accept the language

$L = \{ 0^n 1^n | n >= 1\}$

Solution:

W = {01,0011,000111,……}

Δ	0	0	1	1	Δ	Δ	Δ	Δ	Δ	Δ

Execution Procedure:

Δ0 0 1 1 Δ

Δ 0 0 1 1 Δ

Δ X 0 1 1 Δ

B X 0 1 1 B

B X 0 Y 1 B

B X 0 Y 1 B

B X 0 Y 1 B

B X X Y 1 B

B X X Y 1 B

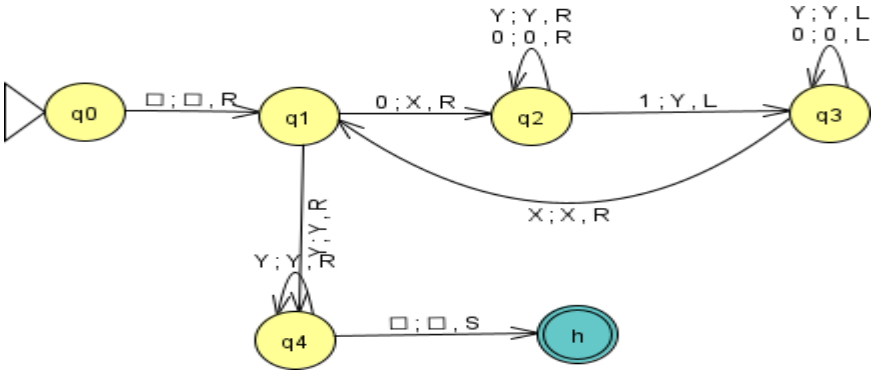
B X X Y Y B

B X X Y Y B

B X X Y Y B

B X X Y Y B

B X X Y Y B



Define Tuples

q_s-start state

Q={qs,qo,q1,q2,q3,h}

F={h}

∑={0,1}

Γ={0,1,X,Y,B}

δ : Transition table :

state	0	1	X	Y	B
Qs	-	-	-		(q0,B,R)
q0	(q1,X,R)			(q3,y,R)	
q1	(q1,0,R)	(q2,Y,L)		(q1,Y,R)	
q2	(q2,0,L)		(q0,X,R)	(q2,Y,L)	
q3				(q3,y,R)	(h,B,S)

Sequence of Moves: 0011 (ID)

(q0, B 0 0 1 1 B) |- (q1, B 0 0 1 1B)

|- (q2, B X 0 1 1 B)

|- (q2, B X 0 1 1 B)

|- (q3, B X 0 Y 1 B)

|- (q3, B X 0 Y 1 B)

|- (q1, B X 0 Y 1 B)

|- (q2, B X X Y 1 B)

|- (q2, B X X Y 1 B)

|- (q3, B X X Y Y B)

|- (q3, B X X Y Y B)

|- (q1,B X X Y Y B)

|- (q4, B X X Y Y B)

|- (q4, B X x Y Y B)

|- (q5, B X X Y Y B)

2. Construct a Turing Machine to accept palindrome over {a,b}

- Even palindrome - abba
- Odd palindrome- aba
- Not a palindrome -abb

Even Palindrome: abba

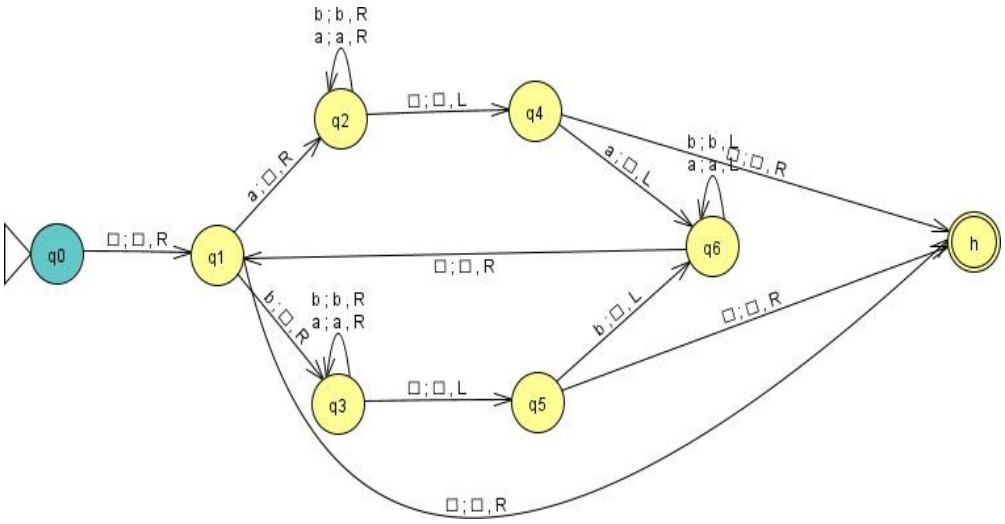
Δ	a	b	b	a	Δ	Δ	Δ	……..

Odd Palindrome: aba

Δ	a	b	a	Δ	Δ	Δ	Δ	Δ	……..

Not a Plalindrome: abb

Δ	a	b	b	Δ	Δ	Δ		……



Define Tuples

q_s-start state

Q={qs,q0,q1,q2,q3,q4,q5,h }

F={h }

∑={a,b}

Γ={ a,b,Δ}

δ : Transition table :

state	**a**	**b**	Δ
qs	-	-	**(q0, Δ,R)**
q0	**(q1, Δ,R)**	**(q4, Δ,R)**	**(h, Δ,R)**
q1	**(q1,a,R)**	**(q1,b,R)**	**(q2,B,L)**
q2	**(q3, Δ,L)**		**(h, Δ,R)**
q3	**(q3,a,L)**	**(q3,b,L)**	-
q4	**(q4,a,R)**	**(q4,b,R)**	**(q6, Δ,L)**
q5	-	**(q3, Δ,L)**	**(h, Δ,R)**

Instantaneous Description:

String: aba

Sequence of Moves for **aba**

(qs, Δ aba Δ) ⊢ (q0, Δ a b a Δ)

⊢ (q1, Δ Δ b a Δ)

⊢(q1, Δ Δ b a Δ)

⊢(q1, Δ Δ b a Δ)

⊢(q2, Δ Δ b aΔ)

⊢(q3, Δ Δ bΔΔ)

⊢(q3, Δ Δ bΔΔ)

⊢(q0, Δ Δ bΔΔ)

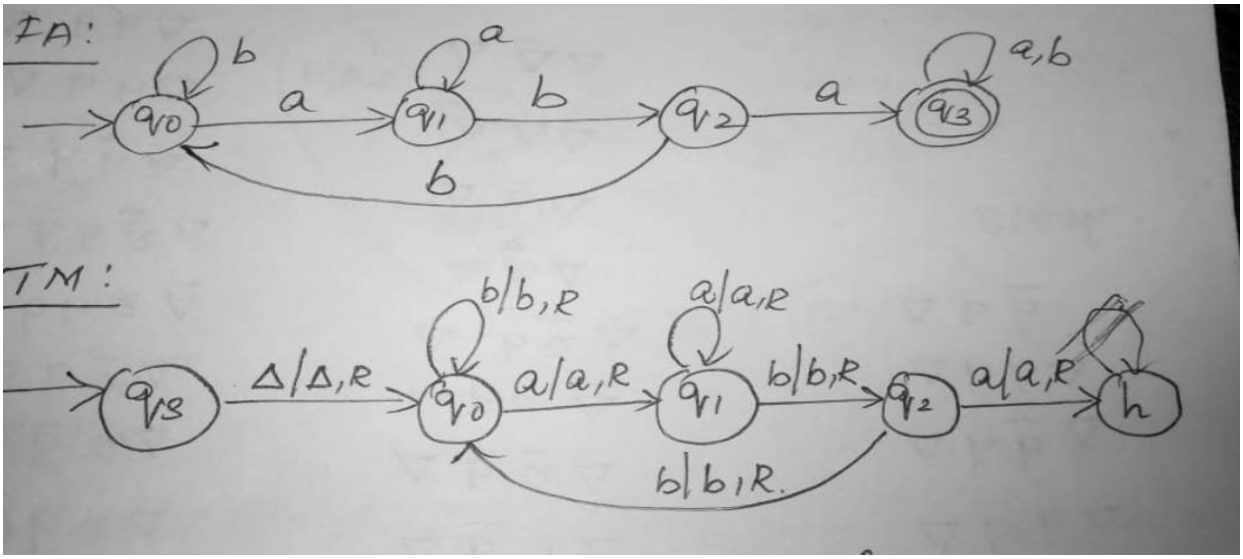
⊢(q4, Δ Δ ΔΔΔ)

⊢(q5, Δ Δ ΔΔΔ)

⊢(h, Δ Δ ΔΔΔ)

Accepted

3. L= { x ∈ {a,b} * | x contains the sub string aba }



4. L={x={a,b}* | x ends with an abb}

R.E=(a+b)*abb

5. Construct a Turing machine to copy a string

Input Tape:

Δ	**a**	**b**	**a**	Δ	Δ	Δ	Δ	Δ	**…….**

Output Tape:

Δ	**a**	**b**	**a**	Δ	**a**	**b**	**a**	Δ	**…….**

ΔabaΔΔΔΔ

Δ**a**ba**Δ**ΔΔΔ

Δ**Ab**a**Δ**ΔΔΔ

Δ**Ab**a**Δ**ΔΔΔ

Δ**Aba**ΔΔΔΔ

Δ**Aba**Δ**Δ**ΔΔ

Δ**Aba**Δ**a**ΔΔ

Δ**Ab**a**Δ**a**Δ**ΔΔ

Δ**A**b**a**Δ**a**ΔΔ

Δ**Ab**a**Δ**a**Δ**ΔΔ

Δ**AB**a**Δ**a**Δ**ΔΔ

Δ**ABa**Δ**a**ΔΔ

Δ**ABa**Δ**Δ**a**Δ**ΔΔ

Δ**ABa**Δa**Δ**ΔΔ

Δ**ABa**Δ**a**̲**b**ΔΔ

Δ**ABa**Δ̲**ab**ΔΔ

Δ**ABa**̲Δ**ab**ΔΔ

Δ**A**̲**B**a**ab**ΔΔ

Δ**ABa**̲Δ**ab**ΔΔ

Δ**ABA**Δ̲**ab**ΔΔ

Δ**ABA**Δ**a**̲**b**ΔΔ

Δ**ABA**Δ**a**̲**b**Δ̲Δ

Δ**ABA**Δ**a**̲**b**a**a**Δ

Δ**ABA**Δ̲**aba**Δ

Δ**AB**A̲Δ**aba**Δ

Δ**ABA**Δ̲**aba**Δ

Δ**ABA**Δ̲**aba**Δ

Δ**AB**A̲Δ**aba**Δ

Δ**A**̲**B**a**aba**Δ

Δ**A**̲**ba**Δ**aba**Δ

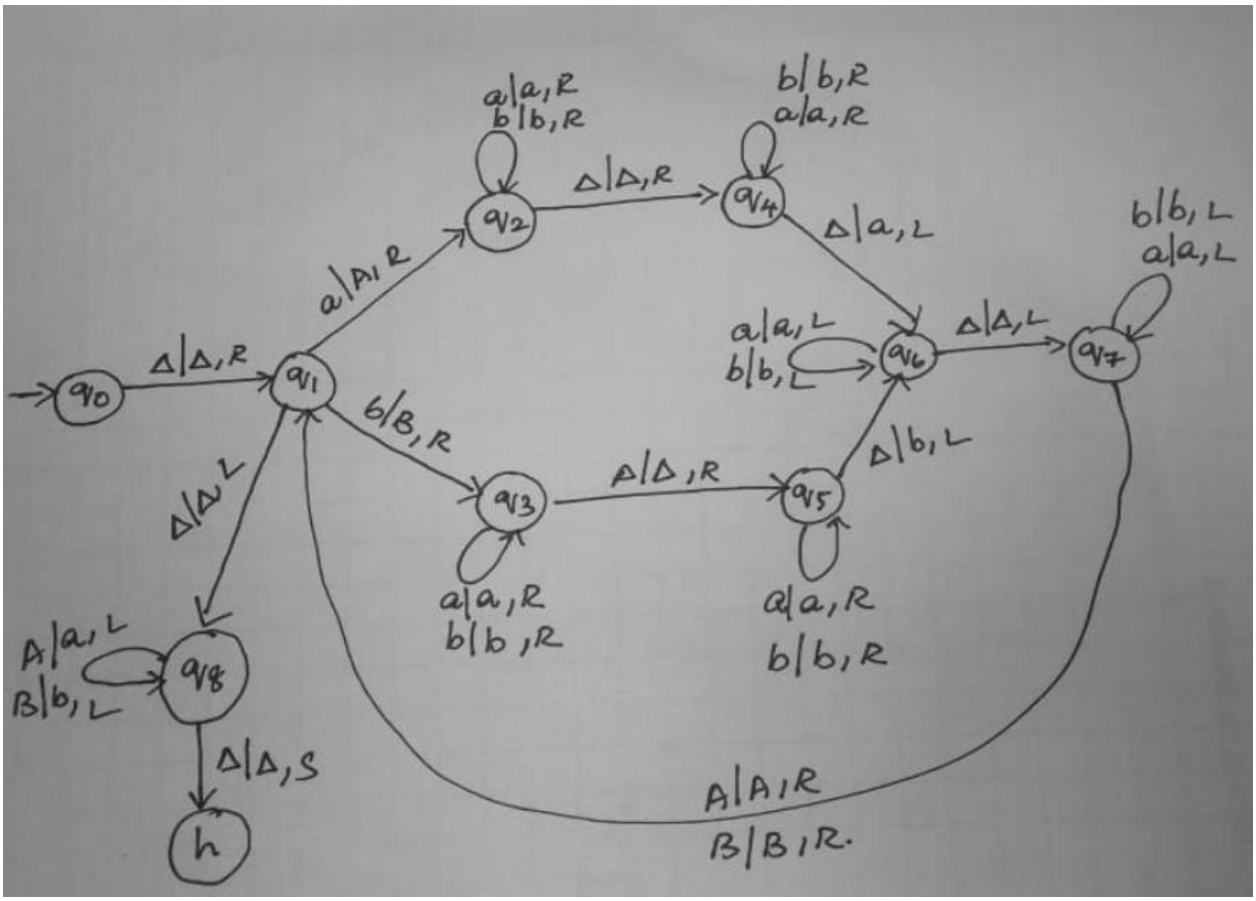
Δ̲**aba**Δ**aba**Δ

Δ**Aba**Δ**a**ΔΔ

Δ**ABa**Δab**a**Δ

ΔABAΔaba

Δaba**a**Δaba



Define Tuples

q_s-start state

Q={qs,q0,q1,q2,q3,q4,q5,q6,q7,q8,h}}

F={h }

∑={a,b}

Γ={ a,b,A.B,Δ}

δ : Transition table :

States	a	b	A	B	Δ
q0	-	-	-	-	(q1, Δ,R)

q1	(q2,A,R)	(q3,B,R)	-	-	(q8, Δ,L)
q2	(q2,a,R)	(q2,b,R)	-	-	(q4, Δ,R)
q3	(q3,a,R)	(q3,b,R)	-	-	(q5, Δ,R)
q4	(q4,a,R)	(q4,b,R)	-	-	(q6,a,L)
q5	(q5,a,R)	(q5,b,R)	-	-	(q6,b,L)
q6	(q6,a,L)	(q6,b,L)	-	-	(q7, Δ,L)
q7	(q7,a,L)	(q7,b,L)	(q1,A,R)	(q1,B,R)	-
q8	-	-	(q8,a,L)	(q8,b,L)	-

Trace the moves: aba

(q0, Δaba Δ Δ Δ Δ Δ) |-(q1, ΔabaΔΔΔΔΔ)

(q2, ΔAba Δ Δ Δ Δ) |-(q2, ΔAba ΔΔΔΔΔ)

|-(q2, ΔAba ΔΔΔΔΔ) |-(q4, ΔAba ΔΔΔΔ)

|-(q6, ΔAba Δa ΔΔ) |-(q7, ΔAba Δa ΔΔ)

|-(q7, ΔAba Δa ΔΔ) |-(q7, ΔAba Δa ΔΔ)

|-(q1, ΔAba Δa ΔΔ) |-(q3, ΔABa Δa ΔΔ)

|-(q3, ΔABa Δ a ΔΔ) |-(q5, ΔABa Δ a ΔΔ)

|-(q5, ΔABa Δ a Δ Δ) |-(q6, ΔABa Δ a b Δ)

|-(q6, ΔABa Δ a b Δ) |-(q7, ΔAB a Δ a b Δ)

|-(q7, ΔAB a Δ a b Δ) |-(q1, ΔAB a Δ a b Δ)

|-(q2, ΔAB A Δ a b Δ) |-(q4, ΔAB A Δ a b Δ)

|-(q4, ΔAB A Δ a b Δ) |-(q4, ΔAB A Δ a b Δ)

|-(q6, ΔAB A Δ a b a) |-(q6, ΔAB A Δ a b a)

|-(q6, ΔAB A Δ a b a) |-(q7, ΔAB A Δ a b a)

|-(q1, ΔAB A Δ a b a) |-(q8, ΔAB A Δ a b a)

|-(q8, ΔAB a Δ a b a) |-(q8, ΔAb a Δ a b a)

|-(q8, Δ ab a Δ a b a) |-(h, Δ ab a Δ a b a)

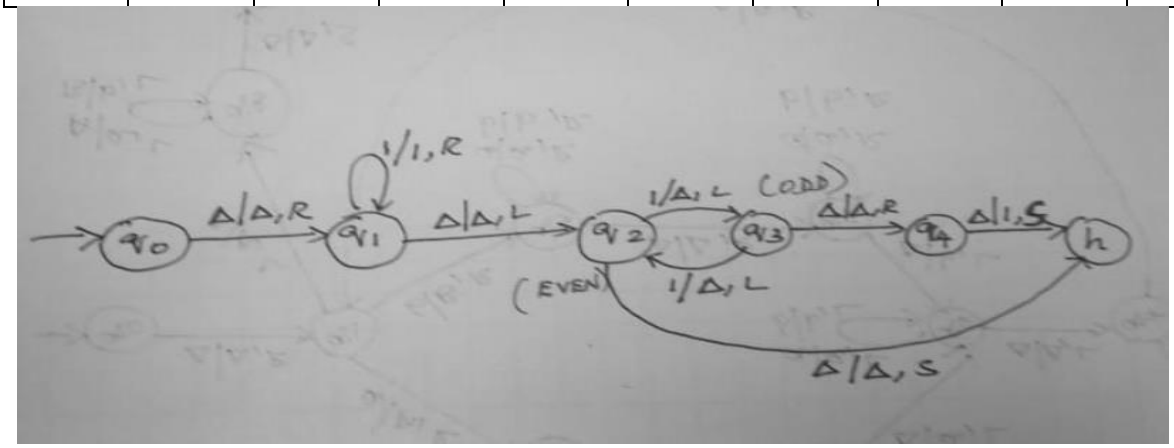
6. Turing machine to construct n mod 2 where n=|x|.

Even:

Δ	1	1	1	1	Δ	Δ	Δ	Δ	…….

Odd:

Δ	1	1	1	Δ	Δ	Δ	Δ	Δ	…….



Define Tuple:

(Q, Γ, ∑, δ, q_0, B, F)

where –

Q- { q0,q1,q2,q3,q4,h}

Γ – {Δ,1}

∑ - {1}

Δ – blank symbol

q0 – Start state

h – Final state

δ:

STATES	1	Δ
q0	-	(q1,Δ,R)
q1	(q1,1,R)	(q2,Δ,L)
q2	(q3,Δ,L)	(h,Δ,S)
q3	-	(q4,Δ,R)
q4	-	(h,1,S)

7. Construct a Turing machine to delete a symbol

Input: ababba

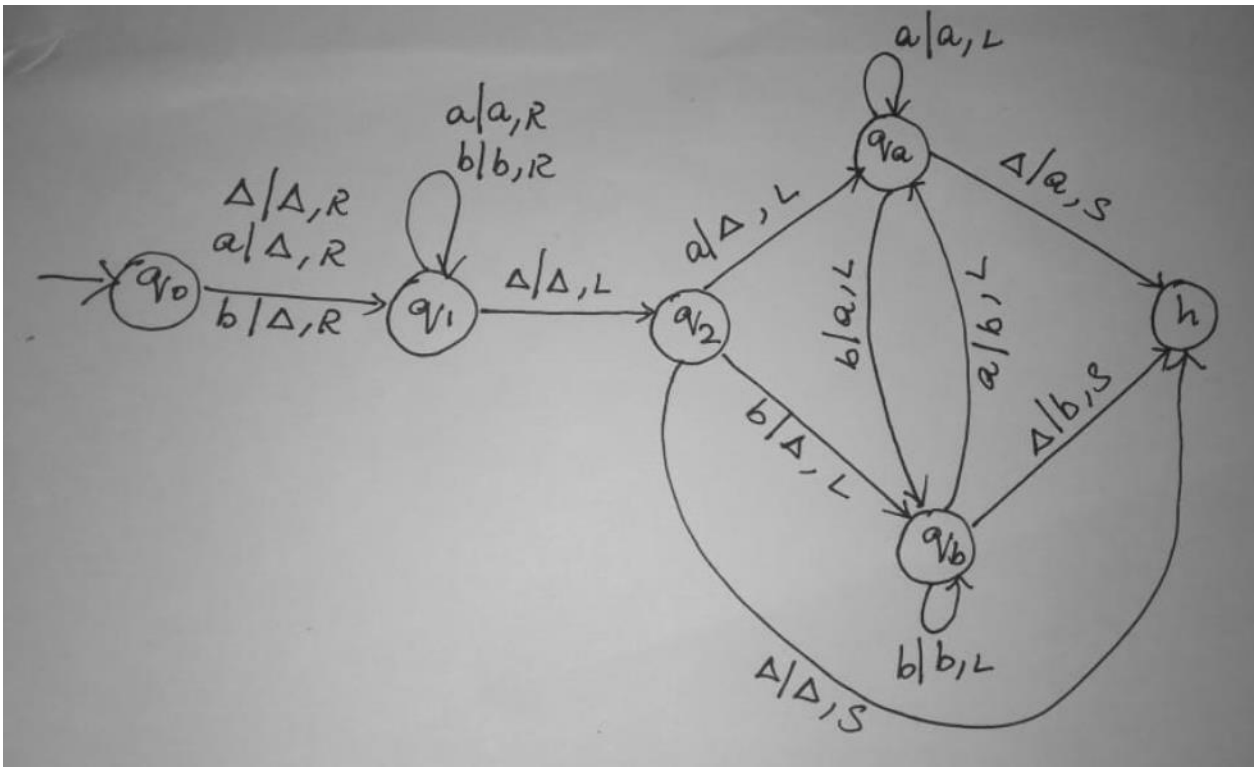
Δ	a	b	a	b	B	a	Δ	Δ	Δ	Δ	Δ	…..

Δ	a	b	Δ	b	b	a	Δ	Δ	Δ	Δ	Δ	Δ	…..

Δ	a	b	b	b	a	Δ	Δ	Δ	Δ	Δ	Δ	……

Execution Logic:

Δ bab Δ **Δ bab Δ** **Δ Δ a b Δ** **Δ Δ a b Δ** **Δ Δ a b Δ** **Δ Δ a b Δ**	**Δ Δ a Δ Δ** **Δ Δ b Δ Δ** **a b Δ Δ - HALT**



Define Tuple:

(Q, Γ, ∑, δ, q_0, B, F)

where –

Q- { q0,q1,q2,q3,q4,h}

Γ – {Δ,1}

∑ - {1}

Δ – blank symbol

q0 – Start stat**e**

h – Final state

Transition Table: δ

STATES	a	b	Δ
q0	(q1,Δ,R)	(q1,Δ,R)	(q1,Δ,R)
q1	(q1,a,R)	(q1,b,R)	(q2,Δ,L)
q2	(qa,Δ,L)	(qb,Δ,L)	(h,Δ,S)
qa	(qa,a,L)	(qb,a,L)	(h,a,S)
qb	(qa,b,L)	(qb,b,L)	(h,b,S)

Trace the moves:

(q0, Δ aba Δ) |- (q1, Δ aba Δ)

|- (q2, Δ Δ b a Δ)

|- (q2, Δ Δ b a Δ)

|- (q2, Δ Δ b a Δ)

|- (q3, Δ Δ b a Δ)

|- (q4, Δ Δ b Δ Δ)

|- (q5, Δ Δ a Δ Δ)

|- (q6, Δ b a Δ Δ) - HALT

8. Construct a Turing machine for the language , L={SS | Sϵ{a,b}*}

The problem is divided into 2 parts:

(i) Finding and marking the middle of the string

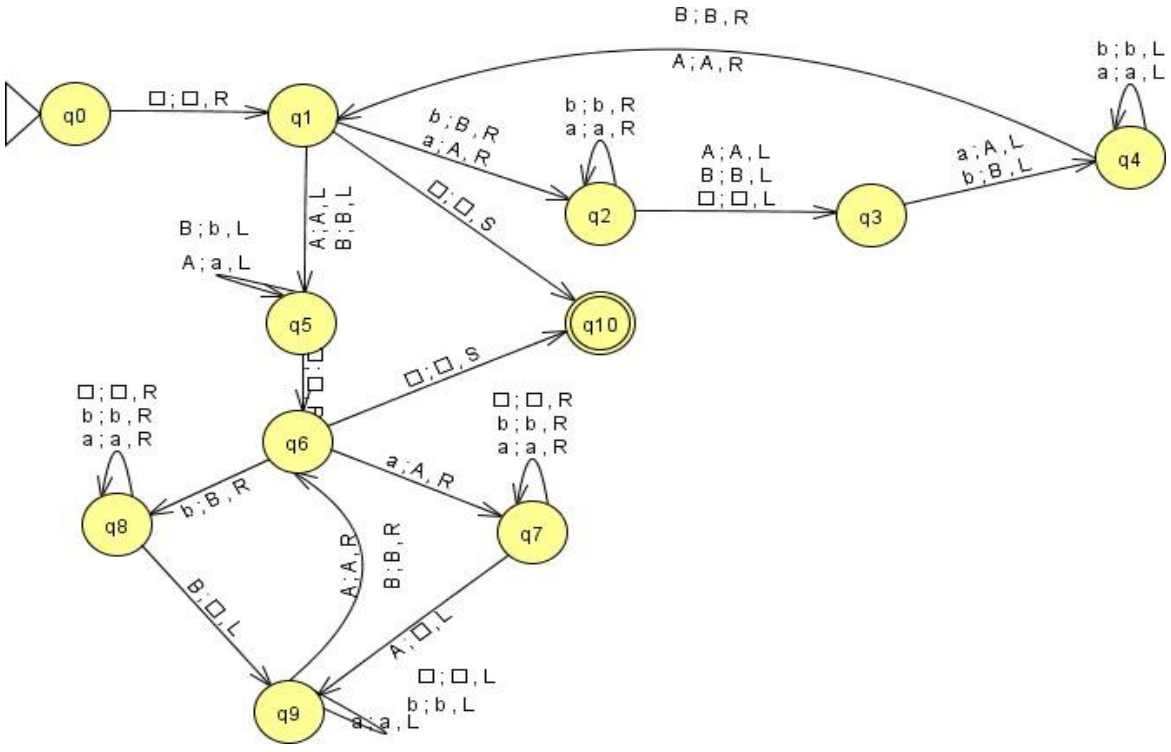
(ii) Comparing the 2 halves.

Δ	a	b	b	a	b	b	Δ	Δ	Δ	Δ	Δ	…..

Δ	a	b	b	a	b	b	Δ	Δ	Δ	Δ	Δ	…..

Δ	a	b	**b**	**A**	**B**	**B**	Δ	Δ	Δ	Δ	Δ	…..

Δ a b b a b b **Δ** **Δ** A b b a b b **Δ** **Δ** A b b a b B **Δ** **Δ** A B b a b B **Δ** **Δ** A B b a b B **Δ** **Δ** A B b a B B **Δ Δ** A B b a B B **Δ Δ** A B B a B B **Δ Δ** A B B A B B **Δ** **Δ** A B B A B B **Δ – Mid point**	**Δ** a b b A B B **Δ – Mid point** **Δ** A b b **Δ** B B **Δ** **Δ** A b b **Δ** B B **Δ** **Δ** A B b **Δ** **Δ** B **Δ** **Δ** A B B **Δ** **Δ** **Δ** **Δ** **Δ** A B B **Δ** **Δ** **Δ** **Δ**



Define Tuple:

(Q, Γ, ∑, δ, q_0, B, F)

where –

Q- { q0,q1,q2,q3,q4,q5,q6,q7,q8,q9,q10}

Γ – {Δ,a,b,A,B}

∑ - {1}

Δ – blank symbol

q0 – Start state

q10– Final state

Transition Table: δ

STATES	a	b	A	B	Δ
q0	-	-	-	-	(q1,Δ,R)
q1	(q1,A.R)	(q1.B.R)	(q5,A,L)	(q5,B,L)	(q10,Δ.S)
q2	(q2,a,R)	(q2.b.R)	(q3,A,L)	(q3,B,L)	(q3,Δ,L)
q3	(q4,A,L)	(q4,B,L)	-	-	-
q4	(q4,a,L)	(q4,a,L)	(q1,A,R)	(q1,B,R)	
q5	-	-	(q5,a,L)	(q5,b,L)	(q6,Δ,R)
q6	(q7,A,R)	(q8,B,R)	-	-	-
q7	(q7,a,R)	(q7,b,R)	(q9,Δ,L)		(q7,Δ,R)
q8	(q8,a,R)	(q8,b,R)	-	(q9,Δ,L)	(q8,Δ,R)
q9	(q9,a,L)	(q9,b,L)	(q6,A,R)	(q6,B,R)	(q9,Δ,L)

Trace the given string : abbabb

9. Construct a turing machine for subtraction

f(m,n) = m-n , m>n

0 , m≤n

Case 1: m>n

Input tape:

Δ	0	0	0	1	0	0	Δ	…..

Output Tape:

Δ	Δ	Δ	Δ	0	Δ	Δ	Δ	…..

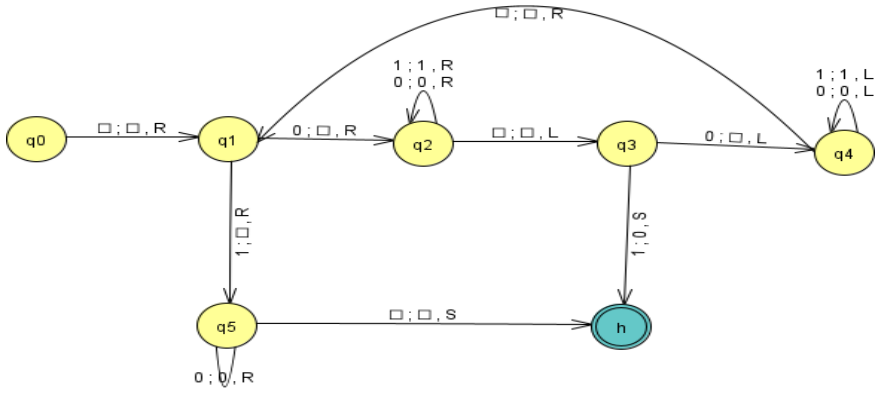
Case 2: m<=n

Input tape:

Δ	0	0	1	0	0	0	Δ	Δ	…..

Output Tape:

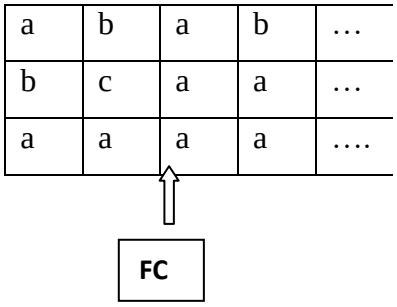
Δ	Δ	Δ	Δ	Δ	Δ	Δ	Δ	…..



3.Variations of Turing Machine

1. Multiple track Turing Machine:

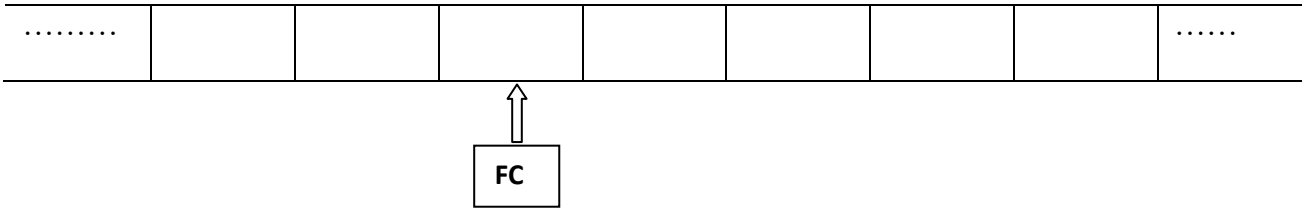
- A k-tack Turing machine(for some k>0) has k-tracks and one R/W head that reads and writes all of them one by one.
- A k-track Turing Machine can be simulated by a single track Turing machine



2. Two-way infinite Tape Turing Machine:

A two way infinite tape turing machine is a turing machine with it's a input tape infinite in both directions, the other components being the same as that of the basic model.

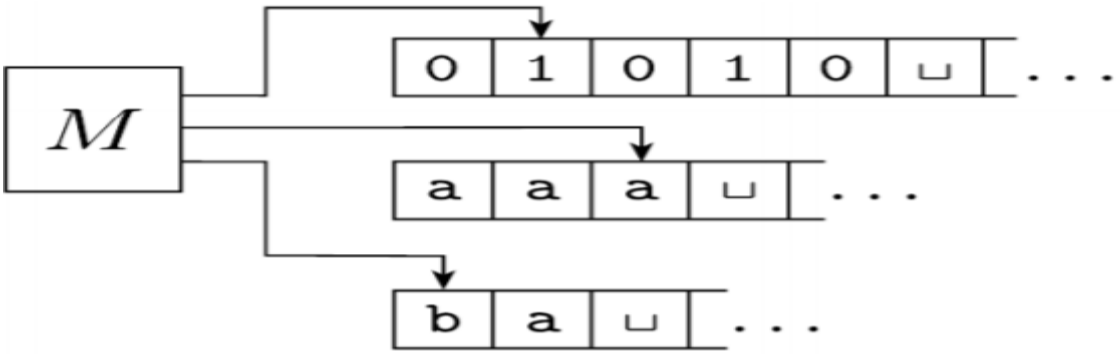
- Infinite tape of two-way infinite tape Turing machine is unbounded in both directions left and right.
- Two-way infinite tape Turing machine can be simulated by one-way infinite Turing machine (standard Turing machine).



3. Multi-tape Single-head Turing Machine:

- It has multiple tapes and controlled by a single head.
- The tape head scans the same position on all tapes.

4. Multi-tape Multi-head Turing Machine:



- The multi-tape Turing machine has multiple tapes and multiple heads
- Each tape controlled by separate head
- Multi-Tape Multi-head Turing machine can be simulated by standard Turing

5. Multi-head Turing Machine:

- A multi-head Turing machine contain 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.

6. Non-deterministic Turing Machine:

- A non-deterministic Turing machine has a single, one way infinite tape.
- For a given state and input symbol has atleast one choice to move (finite number of choices for the next move), each choice several choices of path that it might follow for a given input string.
- A non-deterministic Turing machine is equivalent to deterministic Turing machine.

$$\delta : Q \times \Gamma \to \mathcal{P}(Q \times \Gamma \times \{L, R\})$$

7. Offline Turing Machine

- It is a multitape turing machine whose input tape is read only (writing is not allowed).

- An offline Turing machine can simulate any turing machine A by adding one more tape than Turing machine A. The reason for using an extra tape is that the offline Turing machine makes a copy of its own input into the extra tape and it then simulates Turing machine A as if the extra tape were A's input.

4. Universal Turing Machine

A UTM is a specified Turing machine that can simulate the behavior of any TM.

A UTM is capable of running any algorithm.

It is a Turing Machine whose input consists of 2 parts:

- A string specifying some special purpose Turing Machine, T1.
- A string Z that is an input to T1.

The Turing Machine, Tu then simulates the processing of Z by T1.

4.1. Construction of Tu: $\sum$={0,1}

Step 1: Formulate a notational system

- For each tape symbol (including Δ) as string of 0's
- For each state (including h)
- 3 directions
- For beginning of string and ending of string – 11
- For comma, encoding is 1

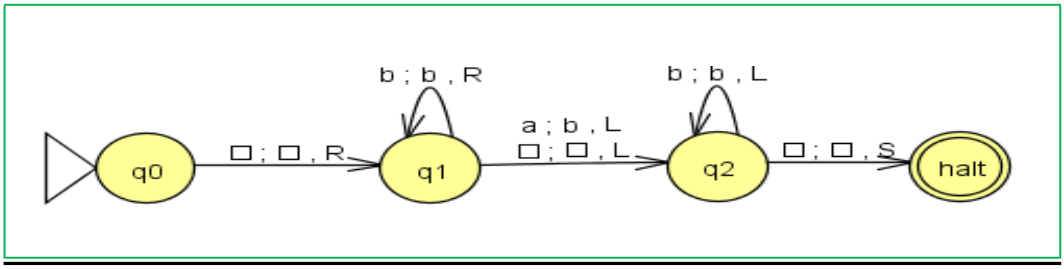
e - encoding function

Tu – represents the Universal Turing Machine

T1 – represents the name of the special Turing machine

Tu=e(T1).e(Z)

4.2. Constuct a UTM for the given Turing Machine



Encoding:

For I/P symbols:

Δ - 0

a -00

b -000

For each state:

h - 0

q0 - 00

q1 - 000

q2– 0000

For directions

S-0

L-00

R -000

Transition Function

δ(q0,Δ)=(q1,Δ,R)

00101000101000

δ(q1,b)=(q1,b,R)

0001000100010001000

δ(q1,a)=(q2,b,L)

000100100001000100

δ(q1,Δ)=(q2, Δ,L)

000101000010100

δ(q2,b)=(q2,b,L)

00001000100001000100

δ(q2,Δ)=(h,Δ,S)

00001001010

e(T1)=
00101000101000110001000100010001000110001001000010001001100010100001010011000010001000100010001001100001001010

Z=bba

e(Z)=0001000100

Tu=e(T1).e(Z)

=1100101000101000110001000100010001000110001001000010001001100010100001010011000010001000100010001001100001001010.11000100010011

For any input string Tu will halt if and only if T halts on input Z.

Output from Tu is in the encoded form of the output produced by T on input Z.

SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE
www.sathyabama.ac.in

SCHOOL OF COMPUTING
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

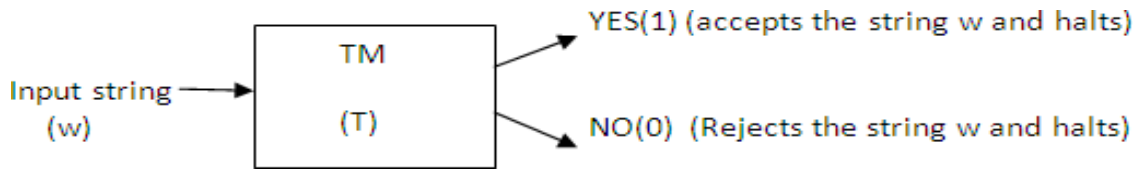
UNIT – V – Theory of Computation – SCSA1302

RECURSIVE LANGUAGES AND UNDECIDABILITY

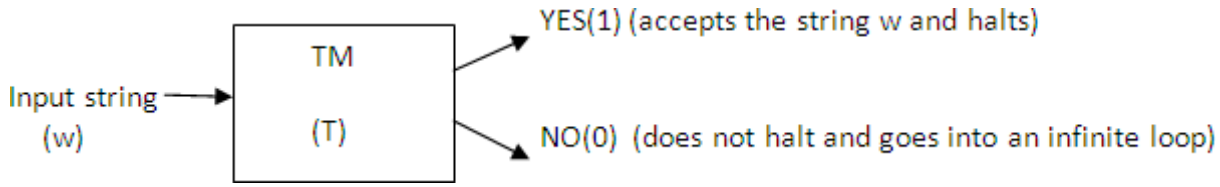
Recursively enumerable and recursive languages – Properties of Recursively enumerable and recursive languages - Enumerating a language. Introduction to Undecidability- Halting problem- Undecidability of Post correspondence problem (PCP)-Modified PCP -Rice Theorem.

1. Recursively Enumerable (REL) & Recursive Languages (RL)

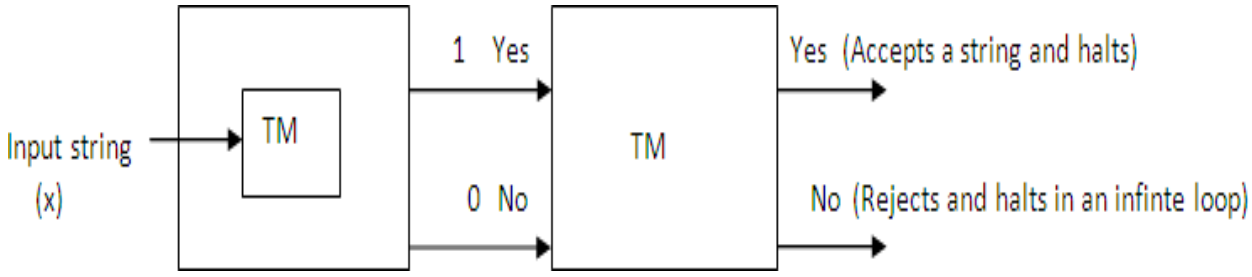
Recursive Language:



Recursively Enumerable Language:

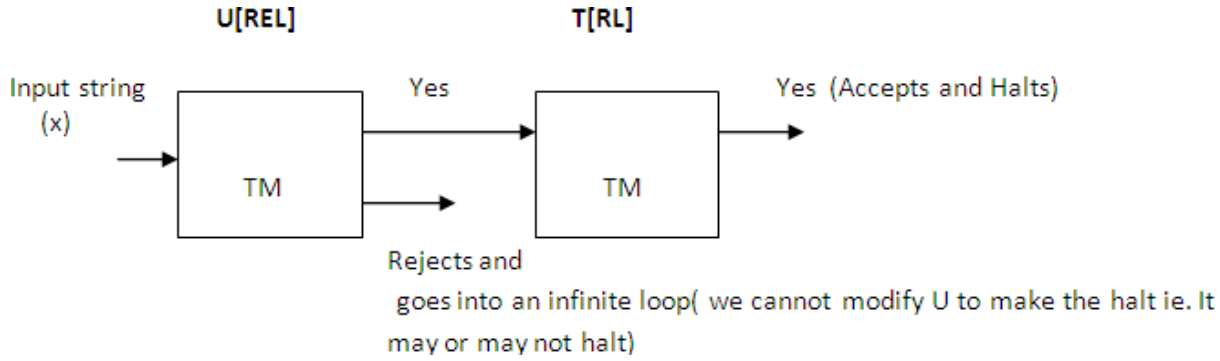


Theorem1:
Every recursive language is Recursively enumerable.

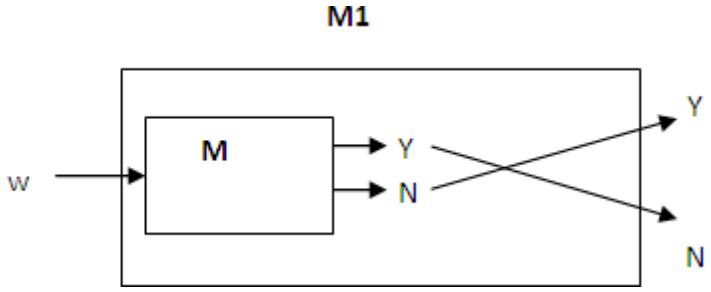


If T is TM recognizing L(RL) then we can get a TM that accepts the language L by modifying T so that when the output is 0 it does not enter the reject state but enters into an infinite loop.

Theorem 2:
Every REL is not recursive.



Theorem 3:
The complement of a RL is recursive (or) If L is recursive, so is L′.



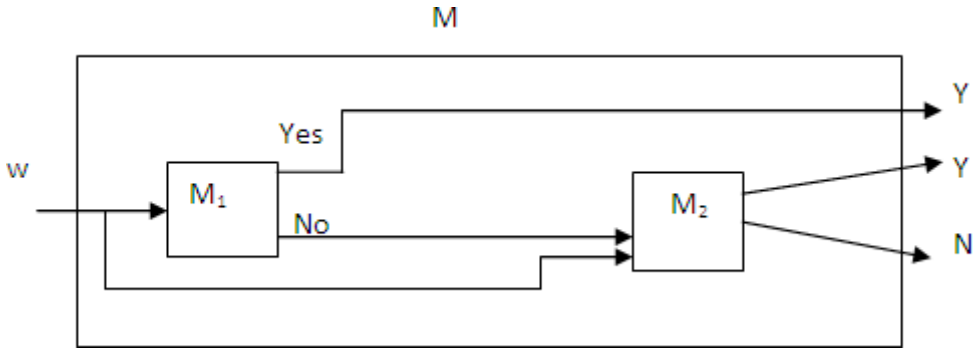
Let L be a Recursive Language and M be the TM that halts on all inputs and accepts L.

- Construct M1 for L′.
- M accepts and halts for Yes, then M′ rejects and halts.
- M rejects and halts for N, then M′ accepts and halts.

Theorem 4:

i. **Union of two recursive languages is recursive.**
ii. **Union of two recursively enumerable languages is REL.**
iii. **Intersection of two recursively enumerable languages is REL.**

i) Union of two recursive languages is recursive.

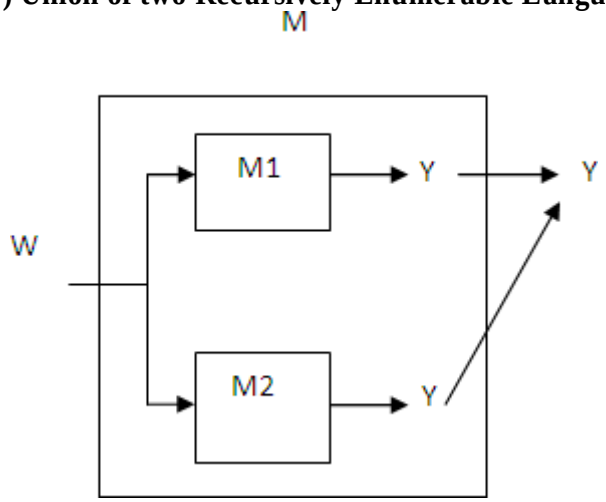


Let L1 and L2 be the Recursive languages accepted by TMs M1 and M2.

Construct M:

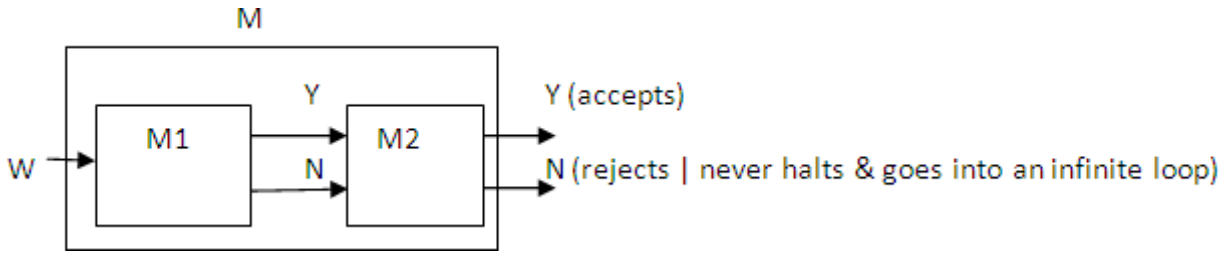
- It first simulates M1,If Yes than M accepts.
- If M1 rejects, then M simulates M2 and accepts if and only if M2 accepts.
- i.e) M accepts L1 U L2.

ii) Union of two Recursively Enumerable Languages is REL.



- Let L1 and L2 be the Recursive Enumerable languages accepted by TMs M1 and M2.
- M simultaneously simulates M1 and M2.
- If either accepts, then M accepts. i.e. M accepts L1 U L2.

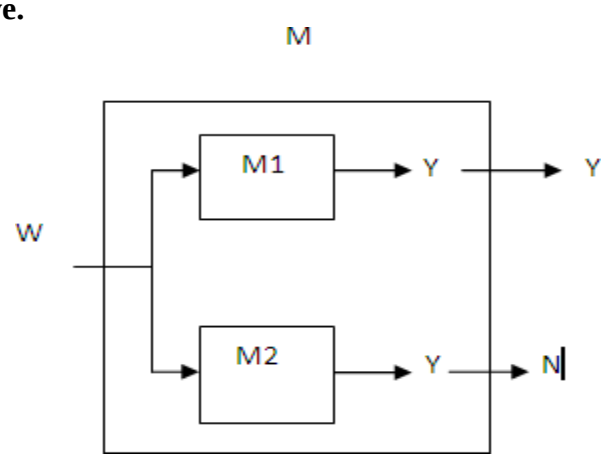
iii) Intersection of two recursively enumerable languages is REL.



- Let L1 and L2 be the Recursive Enumerable languages accepted by TMs M1 and M2.
- M halts if both M1 and M2 halts.
- M will never halt if either M1 or M2 enter into infinite loop.
- (i.e.) M accepts L1 □ L2.

Theorem 5:

If a Language L and its complement L′ are both Recursively Enumerable, then L and hence L′ are recursive.



- Let M1 and M2 be the TMs accepting L1 and L2 respectively.
- M simultaneously simulates M1 and M2.
- M1 accepts L and M2 accepts L′

2. An Unsolvable Decision Problem:

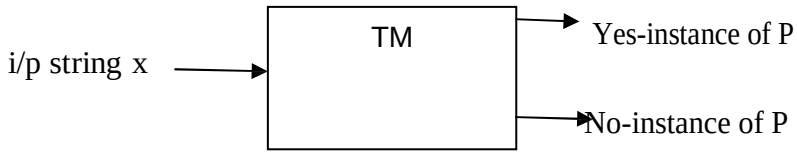
A Turing Machine can solve decision problems.

Eg: Given x ε $\{a, b\}^*$, Is x an element of Palindrome ?

Instance of a problem

- It is a particular string x , so that when the string is provided as input to the TM, the answer is "YES " or "NO"
- For more complicated problems to be solved by a TM, instances may need to be encoded over the input alphabet of the machine.

How TM solves a decision problem P :



Definition 1: Self Accepting Language (SA)

SA = {w ε $\{0,1\}^*$ | w= e(T) for some TM T and w ε L(T) }

Where **w** is any string

e(T) is the encoding function

-If the same input is given to itself (TM) and if output is 1, the TM accepts its own input. (i.e) TM accepts its own encodings e(T).

Definition 2: Non-Self Accepting Language (NSA)
NSA = {w ε { 0,1}* | w= e(T) for some TM T and w ε L(T) }.

Definition 3 : Solvable Problem

A Decision problem is solvable, if there is a algorithm capable of deciding every instance. Recursive language yields solvable DP.

Definition 4: Unsolvable Problem

A Decision problem is unsolvable, if there is no algorithm capable of deciding every instance. Non-Recursive language yields solvable DP.

Theorem 1 :
The language NSA is not recursively enumerable.

Proof by contradiction:
Assume NSA is REL.
Then, let L be a NSA and T be a TM accepting L.
Then, L(T) = NSA and w ε L(T)
But w ∉ L(T), since is not of the form e(T).
Then w ε NSA, is not possible, which implies our assumption is false. Hence,
NSA is not REL.

Theorem 2:
The language SA is Recursively Enumerable but not Recursive

Proof:
W.K.T:

1. If L is recursive, then L' is also recursive.
2. Every RL is also REL.
3. NSA is not REL.

-If SA is recursive, then NSA is also recursive, then NSA is also REL.
-But by 3, NSA is not REL.
-Then, NSA is not recursive and not REL.
-Let T simulate processing w=e(T).
-T halts if w=e(T) & T loops forever if w ε e(T).
-To conclude, SA is REL but not Recursive.

Definition 5: Reducing One Decision Problem to Another

Suppose P1 and P2 are decision problems. We say P1 is reducible to P2 ($P1 \leq P2$) if there is an algorithm that finds, for an arbitrary instance I of P1, an instance F (I) of P2, such that for every I , the answers for the two instances are the same, or I is a yes-instance of P1 if and only if F (I) is a yes-instance of P2.

Definition 6: Reducing One Language to Another

If L1 and L2 are languages over alphabets $\sum_1$ and $\sum_2$, respectively, we say L1 is reducible to L2 ($L1 \leq L2$) if there is a Turing-computable function $f: \sum_1 \rightarrow \sum_2^*$, such that for every $x \in *1, x \in L1$ if and only if f (x) $\in$ L2 .

Theorem 3:
Show that the Accepts problem is unsolvable.

In order to show Accepts is unsolvable it is sufficient to show Self-Accepting $\leq$ Accepts An instance of Self-Accepting is a TM T . A reduction to Accepts means finding a pair F (T) = (T1, y) such that T accepts e(T) if and only if

T1 accepts y. Letting T1 = T and y = e(T) gives us such a pair, and F (T) = (T , e(T)) can be obtained algorithmically from T ; therefore, F is a reduction from Self-Accepting to Accepts.

Theorem 4:
Show that Halting Problem is unsolvable.

In order to prove that Halts is unsolvable, it is sufficient to show that Accepts $\leq$ Halts . With an arbitrary instance (T, x) of Accepts where T is a TM and x is a string. The pair (T , x) = (T1, y), an instance of Halts such that the two answers are the same: T accepts x if and only if T1 halts on input y. T1 should somehow be defined in terms of T .y should be defined in terms of x .

Let y to be x. TM T1 is defined such that for every x, T accepts x if and only if T1 halts on x. A reformulation of this statement is: (i) if T accepts x, then T1 halts on x, and (ii) if T doesn't accept x, then T1 doesn't halt on x.

The Post Correspondence Problem (PCP)

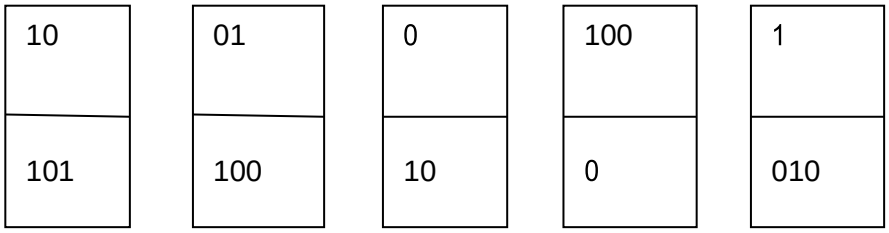
Definition 7 :
An instance of Post's correspondence problem (PCP) is a set $\{(\alpha 1, \beta 1), (\alpha 2, \beta 2), \ldots, (\alpha n, \beta n)\}$ of pairs, where $n \geq 1$ and the αi's and βi's are all non-null strings over an alphabet .

The decision problem:

Given an instance of this type, does there exist a positive integer k and a sequence of integers i1, i2, ...,ik with each ij satisfying $1 \leq ij \leq n$, satisfying

$$\alpha i1 \alpha i2 \ldots \alpha ik = \beta i1 \beta i2 \ldots \beta ik$$

The PCP decision problem is a combinatorial problem involving pairs of strings not related to Turing machines. The figure below shows a sample instance. Each of the five rectangles in the Figure is called a domino, and assume that there is an unlimited supply of each of the five.



The question is whether it is possible to make a horizontal line of one or more dominoes with duplicates allowed, so that the string obtained by reading across the top halves matches the one obtained by reading across the bottom.

Solution:

10	1	01	0	100	100	0	100
101	010	100	10	0	0	10	0

Definition 8:

An instance of the modified Post correspondence problem (MPCP) looks exactly like an instance of PCP, but now the sequence of integers is required to start with 1. The question can be formulated this way:

Does there exist a positive integer k and a sequence i2, i3, ... , ik such that Instances of PCP and MPCP are called correspondence systems and modified correspondence systems, respectively.

For an instance of either type, if it is a yes-instance we will say that there is a match for the instance, or that the sequence of subscripts is a match, or that the string formed by the αij 's represents a match.

$$\alpha 1 \alpha i2 \ldots \alpha ik = \beta 1 \beta i2 \ldots \beta ik$$

More Unsolvable Problems

1. Assuming L is a recursively enumerable language): Accepts L: Given a TM T, is L(T) = L?

2. Accepts something: Given a TM T, is there at least one string in L(T)?\

3. Accepts two or more : Given a TM T , does L(T) have at least two elements?

4. Accepts finite: Given a TM T, is L(T) finite? 5. Accepts Recursive: Given a TM T, is L(T) (which by definition is recursively enumerable) recursive.

5. MPCP $\leq$ PCP.

6. Accepts $\leq$ MPCP.

Halting Problem:
In the theory of computability, the problem of halting is the question of deciding, from an arbitrary computer program description and an input, whether the program will finish running or continue to run indefinitely. In 1936, Alan Turing proved that there could not be a general algorithm for all possible program-input pairs to solve the halting problem.

Given a program and an input to the program, determine if the program will eventually halt when it is given that input.

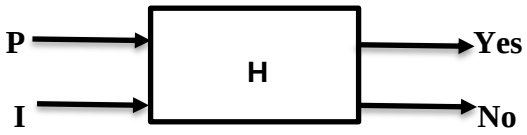
Halting problem is unsolvable

Proof by Contradiction

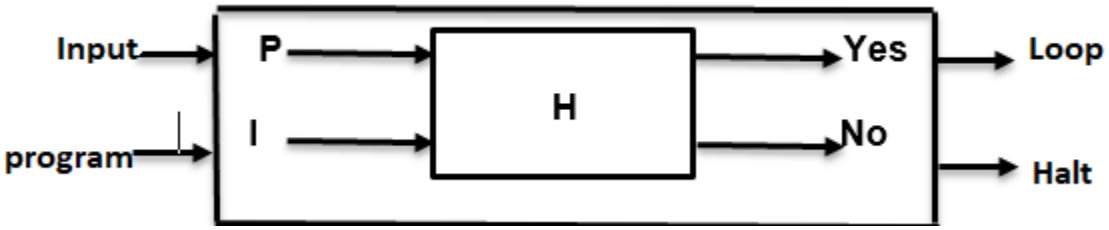
1. Assume the statement is true.
2. Solve the problem.
3. Check if there is a contradiction.

Proof:

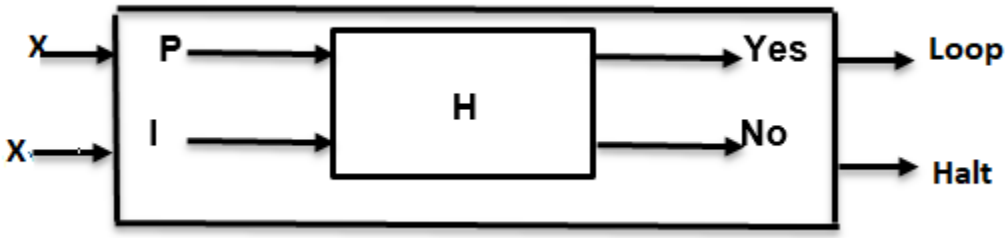
1. Assume it is possible to solve the halting problem.
   Assume it is possible to construct a machine H that solves the halting problem.
   H receives 2 inputs:
   P- program
   I – input to the program P
   H machine gives an answer 'yes' if program P halts on input I.
   If it goes into an infinite loop, H gives answer no.



2. Create a new machine using machine H as basis.
   Let the machine be X.
   Add a condition at the end of machine H such that:
   -If it outputs a 'Yes' answer it will loop.
   - If it outputs a 'No' answer it will halt.



If we give the program of machine X to itself along with the given set of inputs which has already been generated.



If the output of the machine H inside X is 'Yes', it means X will loop which contradicts the result of machine H.

"**Yes means it will halt but it didn't**"

In the same way, if the result of machine H is 'No', it means X will halt which again contradicts the result of H.

"No means it will not halt but it halted"

This machine H does exist and the halting problem is unsolved.

Rice Theorem

Rice theorem states that any non-trivial semantic property of a language which is recognized by a Turing machine is undecidable. A property, P, is the language of all Turing machines that satisfy that property.

Definition 9

If P is a non-trivial property, and the language holding the property, Lp , is recognized by Turing machine M, then Lp = {<M> | L(M) ∈ P} is undecidable.

Properties

- Property of languages, P, is simply a set of languages. If any language belongs to P (L ∈ P), it is said that L satisfies the property P.
- A property is called to be trivial if either it is not satisfied by any recursively enumerable languages, or if it is satisfied by all recursively enumerable languages.
- A non-trivial property is satisfied by some recursively enumerable languages and are not satisfied by others. Formally speaking, in a non-trivial property, where L ∈ P, both the following properties hold:
    - Property 1 − There exists Turing Machines, M1 and M2 that recognize the same language, i.e. either (<M1>, <M2> ∈ L) or (<M1>,<M2> ∉ L)
    - Property 2 − There exists Turing Machines M1 and M2, where M1 recognizes the language while M2 does not, i.e. <M1> ∈ L and <M2> ∉ L

Proof:

Suppose, a property P is non-trivial and φ ∈ P.

Since, P is non-trivial, at least one language satisfies P, i.e., L(M0) ∈ P , ∋ Turing Machine M0.

Let, w be an input in a particular instant and N is a Turing Machine which follows −

On input x

- Run M on w
- If M does not accept (or doesn't halt), then do not accept x (or do not halt)
- If M accepts w then run M0 on x. If M0 accepts x, then accept x.

A function that maps an instance ATM = {<M,w>| M accepts input w} to a N such that

- If M accepts w and N accepts the same language as M0, Then L(M) = L(M0) ∈ p
- If M does not accept w and N accepts φ, Then L(N) = φ ∉ p

Since ATM is undecidable and it can be reduced to Lp, Lp is also undecidable.