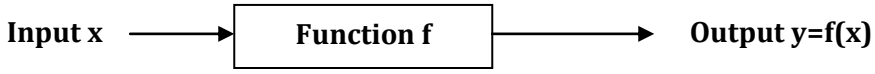
UNIT- I

Finite Automata and Regular Expressions: Formal Languages and Regular expressions, Deterministic and Non-Deterministic Finite Automata, Finite Automata with ε-moves, Equivalence of NFA and DFA, Minimization of finite automata, Two-way finite automata, Moore and Mealy machines, Applications of finite automata.

2 MARKS

1. **What is meant by Computations?**

   Computation is executing an algorithm (i.e.) it involves taking some inputs and performing required operation on it to produce an output.



   - Computation is the sequence of steps that can be performed by computer.
   - The computer which performs computation is not actual computers they are "abstract machines". It can be defined mathematically.

2. **Give example of abstract machines?**
   - Turing Machine (Powerful as real Computers)
   - Finite Automata (Simple)

3. **What is a formal language?**

   Language is a set of valid strings from some alphabet. The set may be empty, finite or infinite. L(M) is the language defined by machine M and L(G) is the language defined by context free grammar. The two notations for specifying formal languages are:
   - Grammar or regular expression (Generative approach)
   - Automaton (Recognition approach)

4. **What is theory of computations?**

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

   The field is divided into three major branches:
   1. automata theory,
   2. computability theory, and
   3. computational complexity theory

5. What is Automata Theory?

- Automata theory is the study of abstract machines (or abstract 'mathematical' machines or systems) and the computational problems that can be solved using these machines. These abstract machines are called automata.
- Automata which means that something is doing something by itself. Automata theory is also closely related to formal language theory, as the automata are often classified by the class of formal languages they are able to recognize. An automaton can be a finite representation of a formal language that may be an infinite set.

6. Why are switching circuits called as finite state systems?

A switching circuit consists of a finite number of gates, each of which can be in any one of the two conditions 0 or 1. Although the voltages assume infinite set of values, the electronic circuitry is designed so that the voltages corresponding to 0 or 1 are stable and all others adjust to these values. Thus control unit of a computer is a finite state system.

7. What are the Operations on strings?

- Length of string
- Empty or null string
- Concatenation of string
- Reverse of a string
- Powers of an alphabet
- Substring
- Reversal
- Kleene Closure

8. Define alphabets in automata?

An alphabet is a finite, non-empty set no of symbols. It is denoted by Σ.

Example:

- ✓ Σ = {*a, b*}, an alphabet of 2 symbols *a* and *b.*
- ✓ Σ = {*0, 1, 2*} an alphabet of 3 symbols *0, 1*, and *2.*

9. Define strings with examples?

A String or Word is a finite sequence of symbols from Σ. The group of characters also referred as a String.

Example:

- ✓ *a, b,* and *c* are symbols and *abcb* is a string.
- ✓ If Σ = {*a,b*} then *abab,aabbb,aaabb,……* are all strings over the alphabet Σ ={*a,b*}.
- ✓ *0, 1, 11, 00,* and *01101* are strings over {0, 1}.

10. What is substring with example?

Let *x* and *y* be strings over an alphabet Σ.

The string *x* is a substring of *y* if there exist strings *w* and *z* over Σ such that *y = w x z.*

- ε is a substring of every string.

- For every string *x*, *x* is a substring of *x* itself.

Example:

- ε, comput and computation are substrings of computation.

11. What is Reversal with example?

Let *x* be a string over an alphabet N.The reversal of the string *x*, denoted by *x r*, is a string such that

- if *x* is ε, then *xr* is ε.
- If *a* is in Σ, *y* is in Σ* and *x* = *a y*, then *xr* = *yr a*.

(*automata*)*r* =

= (*utomata*)*r a*

= (*tomata*)*r ua*

= (*omata*)*r tua*

= (*mata*)*r otua*

= (*ata*)*r motua*

= (*ta*)*r amotua*

= (*a*)*r tamotua*

= (ε)*r atamotua*

- = atamotua The set of strings created from any number (0 or 1 or …) of symbols in an alphabet Σ is denoted by Σ*.

12. Explain the following a. Length of string b. Empty or null string?

a. Length of string

Let w is a string. Length of the string is |w|=> number of symbols composing the string.

Example:

1) If w=abcd then |w|=4

2) If x=01010110 the |x|=8

3) If y= 0101 the |y|=4

4) If |Є| = 0

b. Empty or null string

The string consisting of zero symbols. It is denoted by Є.

Example:

{Є}=0.

13. Define Concatenation of string and Reverse of a string?

Concatenation of string

Appending the strings (2) referred as concatenation of strings.

i.e w=a1,a2,….am

v=b1,b2,b3,……bn

then wv=a1,a2,….am b1,b2,…..bn

Example:

1. x=00 y=1

   xy=001, yx=100

   x=AL y=GOL

   xy=ALGOL

2. Empty string is the identity element for concatenation operator. i.e

   w Є= Єw=w

Reverse of a string

The reverse of a string is obtained by writing the symbols in reverse order.

Let w is the string

Reverse is w^R

i.e w=a1,a2,……am

w^R=am,……a2,a1

Example:

Let u=010111

u^R=1101010

14. What is meant by Powers of an alphabet?

Let Σ be the alphabet

Σ * is the set of all strings over alphabet Σ.

Σ^m denotes the set of all strings over alphabet Σ of length m.

Example:

Σ = {0, 1}

Σ^0= {Є} empty string of length 0.

Σ^1= {0, 1} is the set of all strings of length one over Σ = {0, 1}

Σ^2= {00, 01, 10, 11} is the set of all strings of length two over Σ = {0, 1}.

15. Define Kleen Closure?

Let Σ be the alphabet

" Kleen Closure"-> Σ* denotes the set of all strings (Σ) over the alphabet

Example:

If Σ ={a} the Σ*={ Є,a,aa,aaa,…….}

If Σ ={0,1} the Σ*={ Є,0,1,00,10,10,……..}

Σ^0={ Є}

Σ^1={a}

Σ^2={aa}

Therefore Σ*= $\Sigma^0 U \Sigma^1 U \Sigma^2$,……..

16. Define a language.

An alphabet is a finite set of symbols. A language is a set of strings of symbols from someone alphabet.

e.g. If Σ = {0,1},then Σ*={ Є,0,1,01,11,10,111,…….}

17. List the applications of Computations?

- Analysis of algorithms
- Complexity Theory
- Cryptography
- Compiler Design
- Circuit design
- Robotics
- Artificial Intelligence
- Knowledge Engineering

18. Define Finite Automata. (NOV'15)

- A finite automaton (FA) consists of a finite set of states and a set of transitions from state to state that occur on input symbols chosen from an alphabet Σ.
- Each input symbol there is exactly one transition out of each state (possibly back to the state itself).
- One State, usually q_0, is the initial state, in which the automaton starts.
- Some states are designated as final or accepting states.

19. How does an FA work?

- At the beginning,
  - ✓ an FA is in the start state (initial state)
  - ✓ its tape head points at the first cell
- For each move, FA
  - ✓ reads the symbol under its tape head
  - ✓ changes its state (according to the transition function) to the next state determined by the symbol read from the tape and its current state
  - ✓ move its tape head to the right one cell

20. Give the purpose of transition diagram? (APR 2014)

A directed graph, called a transition diagram, is associated with an FA as follows:

- The vertices of the graph correspond to the states of the FA.
- If there is a transition from state *q* to state *p* on input *a,* then there is an arc labeled *a* from state *q* to state *p* in the transition diagram.
- The FA accepts a string *x* if the sequence of transitions corresponding to the symbols of *x* leads from the start state to an accepting state.

21. What is Transition Table?

The transition table is the tabular representation of the transition function "delta (∂)" with the rows denoting states and columns denoting input symbol.

22. Write the DFA Specifications? (APR 2014, NOV 2012)

Deterministic Finite Automata (DFA) is generally specified by a 5 tuples **$(Q, \Sigma, \delta, q_0, F)$**

where

1. **Q** is a finite set of states
2. **Σ** is a finite input alphabet
3. **q_0** in Q is the start state or initial state
4. **F⊆Q** is the set of final states
5. **δ** is the transition function mapping **δ: QX Σ→Q**.(i.e.) δ(q,a) is a state for each state *q* and input symbol *a.*

23. Explain Working of DFA?

1. **Initialisation**
   - Reader (read head) should be over the leftmost symbol.
   - Finite Control is in start state.
2. **Single Step**
   - Reader moves to the next symbol to the right.
   - Control enters a new state that (deterministically) depends on current state and current symbol. There may be no desired next state, Machine will stop.
   - The machine repeats this action.
3. **No current symbol**
   - All Symbols have been read.
   - If control is in final state, the input string is accepted.
   - Otherwise, the input string is not accepted.

24. What are the Applications of DFA?

- Compiler Design
- Design of Lexical Analyser
- Text Editor
- Pattern Matching
- File searching Program
- Text Processing

25. Define NFA. **(APR 2012, 2013)**

Non-Deterministic Finite Automata (NFA) is generally specified by a 5 tuples **(Q, Σ, δ, q_0, F)** where

1. **Q** is a finite set of states
2. **Σ** is a finite input alphabet
3. **q_0** in Q is the start state or initial state
4. **F⊆Q** is the set of final states
5. **δ** is the transition function mapping **δ → Q x Σ* to 2^Q.**

On receiving same inputs it goes to many states

- the machine can move without consuming any symbols and sometimes there are possible moves and sometimes there are move then one possible moves.
- the state is only partially determined by the current state and i/p symbol.

26. What are the difference between NFA and DFA? (MAY'15, NOV'15 , NOV'17)

DFA	NFA
i. For each and every state and for each and every input symbol there exists at most one transition. **ii.** The transition mapping for DFA is **Q×Σ→Q.** **iii.** The language accepted by DFA is denoted as L(M). **iv.** Epsilon transition is not possible.	**i.** For each and every state and for each and every input symbol there exists more than one transition. **ii.** The transition mapping for NFA is **$Q \times (\Sigma \cup \{\varepsilon\}) \rightarrow 2^Q$.** **iii.** The language accepted by NFA is denoted by L(M'). **iv.** Epsilon transition is possible.

27. What are Transition Function and give their properties?

Changing from one state to another state on applying the inputs

Properties of Transition function:

1. $\delta(q, \varepsilon)=q$

   This means the state of the system can be changed only by an input symbol else remains in original state.

2. For all strings w and input symbol a $\delta(q,aw)= \delta(\delta(q,a),w)$

   Similarly $\delta(q,wa)= \delta(\delta(q,w),a)$

3. The transition function δ can be extended to δ or δ that operates on states and strings.

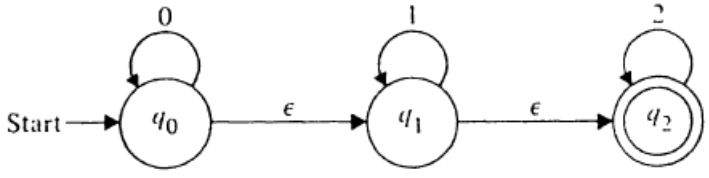
   Basis: $\delta(q, \varepsilon)=q$

   Induction: $\delta(q,xa)= \delta(\delta(q,x),a)$

28. Define finite automata with ε moves?

- The Non deterministic finite automaton with ε moves to be a quintuple $(Q, \Sigma, \delta, q_0, F)$ with all components as before, but δ, the transition function, maps $\delta: Q \times (\Sigma \cup \{\varepsilon\}) \rightarrow 2^Q$.
- The intention is that δ(q,a) will consists of all states *p* such that there is a transition labeled *a* from *q* to *p*, where *a* is either *ε* or a symbol in Σ.

  ε- closure (q) ➔ denotes the set of all vertices P such that there is path from q to P labeled ε.



Finite automaton with ϵ-moves.

29. Define regular expressions? (NOV'14)

The languages accepted by finite automata are easily described by simple expressions called regular expression.

Let Σ be an alphabet. The regular expression over Σ and the sets that they denote are defined recursively as follows:

1. Φ is a regular expression and denotes the empty set.
2. ε is a regular expression and denotes the set { ε }.
3. For each a ϵ Σ, 'a' is a regular expression and denotes the set {a}.
4. If r and s are regular expressions denoting the languages R and S respectively then (r+s), (rs), and (r)* are regular expressions that denotes the sets RUS, RS and R* respectively.

30. What is Kleene and positive closure?

- The Kleene closure of L, denoted L*, is the set

$$L^* = \bigcup_{i=0}^{\infty} L^i$$

- The positive closure of L, denoted L^+, is the set

$$L^+ = \bigcup_{i=1}^{\infty} L^i$$

31. What are the Identities for Regular Expression?

1. Φ + R = R
2. ΦR = R Φ = Φ
3. λR = R λ = R
4. λ* = λ and Φ* = λ
5. R + R = R
6. R* R* = R*
7. RR* = R*R
8. (R*)* = R*
9. λ+ RR* = R* = λ+R*R
10. (PQ)* P = P(QP)*
11. (P + Q)* = (P* Q*)* = (P* + Q*)*
12. (P + Q)R = PR + QR and R(P +Q) = RP + RQ

32. Define two way finite automata (2DFA)? (APR 2012, NOV 2013)

- A finite automata is a control unit that reads a tape, moving one square right at each move.
- A two-way finite automaton is an extension of one- way finite automaton which has the ability to allow the tape head to move left as well as right.
- Two-way finite automata accepts the input string if it moves the tape head, off the right end of the tape at the same time, entering an accepting state.
- A two-way deterministic finite automata (2DFA) is a quintuple M= (Q, Σ, δ, q_0, F)

  Where δ is a map from **Q X Σ →Q X {L, R}**

33. Write the Formal Definition of 2DFA?

A two-way finite deterministic automata (2DFA) is a quintuple **M= (Q, Σ,δ, q_0, F)**

Where

1. **Q** is a finite set (the states)
2. **Σ** is a finite set (the input alphabet)

3. **q_0** in Q is the start state or initial state
4. **F⊆Q** is the set of final states
5. **δ** is a map from **Q X Σ → Q X {L, R}**

34. Define Instantaneous Description (ID) of 2DFA?

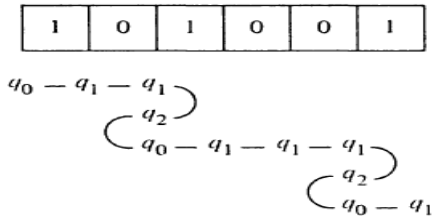
- Instantaneous Description of 2DFA describes the input string, current state, and current position of input head.
- The relation $\vdash_M$ on Id's such that $I_1 \vdash_M I_2$ if and only if M can go from instantaneous description I_1 to I_2 in one move.
- An ID of M is a string in Σ*Q Σ*. The ID *wqx,* where w and x are in Σ* and *q* is a state in Q, represent the facts that
  1. *wx* is the input string
  2. *q* is the current state and
  3. Input head is scanning the first symbol of *x.*

35. What is crossing sequences in 2DFA?

The list of states below each boundary between square is termed a crossing sequence.

36. What is meant by behavior of 2DFA?

In 2DFA consists of the input, the path followed by the head and the state each time the boundary between two tape squares is crossed, with the assumption that the controls enters its new state prior to moving the head.



37. Define Moore and mealy machine?

- In Moore machine the output depends upon the states.
- In Mealy machine the output depends on transition and current state.

38. Write the specifications of Moore machine?

A Moore machine is a 6 tuple **(Q, Σ, Δ, δ , λ ,q_0),**

Where

1. **Q** is a finite set of states
2. **Σ** is a finite input alphabet
3. **Δ** is the output alphabet
4. **λ** is the mapping from Q to Δ giving the output associated with each state
5. **δ** is the transition function
6. **q_0** in Q is the start state or initial state

In Moore machine where the output alphabet is {0, 1} and state "accepting" if only if λ(q)=1.

39. Write the specifications of Mealy machine?

A Mealy machine is a 6 tuple M= $(Q, \sum, \Delta, \delta, \lambda, q_0)$,

Where

1. **Q** is a finite set of states
2. **Σ** is a finite input alphabet
3. **Δ** is the output alphabet
4. **λ** is the mapping from Q X ∑ → Δ (i.e.) λ(q,a) gives the output associated with the transition from state *q* on input *a.*
5. **δ** is the transition function
6. $\mathbf{q_0}$ in Q is the start state or initial state

40. State the difference between Mealy and Moore machine?

MEALY MACHINE	MOORE MACHINE
i. For each and every transition there will be an output. **ii.** An automaton in which the output depends on the state as well as on the input at any instant of time is called a Mealy machine. **iii.** The transition mapping is given by, **λ:Q×Σ→Δ**	**i.** For each and every state there will be an output. **ii.** An automaton in which the output depends on the states of the states of the machine is called Moore machine. **iii.** The transition mapping is given by, **λ:Q→Δ**

41. Give the applications of finite automata? (NOV 2012,NOV'14, NOV'17)

Lexical Analyzers

- Tokens of a programming language are regular sets.
- **Example:** ALGOL identifiers, which are upper-or lower-case letters followed by any sequence of letters and digits with no limit on length, expressed as,

  (letter)(letter + digit)*

  where
  - ✓ letter stands for A + B + …… + Z + a + b + …… + z and
  - ✓ digit stands for 0 + 1 + ….. +9

Text Editors

- Certain text editor and similar programs permits substitution of string for any string matching given regular expression.
- For example: the UNIX text editor allows a command such as

  s /bbb*/b*

  that substitutes a single blank for the first string of two or more blanks found in a given line.

  Let "any" represents expression $a_1+a_2+\ldots..+a_n$.

  where a_i are computer characters except "newline "character.

42. Define FORTRAN identifiers?

- FORTRAN identifiers, with length should limited of six and letters restricted to uppercase and symbol $, may be expressed as

(letter)(ε + letter + digit)*

Where, letter stands for ($ + A + B +……. + Z)

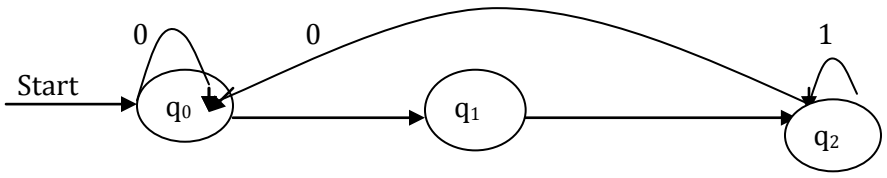
43. Define the SNOBOL?

- SNOBOL arithmetic constants(do not permit the exponential notation) may be expressed as

(ε + -)(digit+ (.digit* + ε) +.digit+)

- Lexical analyzers takes input as sequence of regular expressions describing the tokens and produce a single finite automaton recognizing the any token.
- Usually Regular expression to NFA with ε moves to NFA without ε moves and to DFA.

44. Construct DFA that accepts input stings of 0's and 1's that end with 11.



The language accepted by this automaton is L = {11, 011, 01011, and 000111}

1. Machine will be in start state q_0.
2. It will remain unchanged until it sees the input symbol '1'.
3. On getting '1' goes to q_1.
4. From q_1, on input symbol '0' goes to q_0. From q_1, one i/p symbol '1' goes to q_2 (acceptance state).
5. From q_2, any number of 1 is accepted and if any 0 comes it goes to start state.

45. Check whether given input string 0111 is accepted or not?

Transition table

	States	Input	
		0	**1**
→*	q_0	$\{q_0\}$	$\{q_1\}$
	q_1	$\{q_0\}$	$\{q_2\}$
	q_2	$\{q_2\}$	$\{q_1\}$

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

Solution

Let M be the Machine. It consists of the following attributes or tuples.

M= (Q, Σ, δ, S, F)

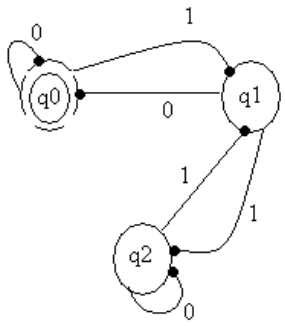
Q= $\{q_0, q_1, q_2\}$

Σ = {0, 1}

$S=q_0$

$F= \{q_0\}$

Transition Diagram



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

$\delta (q_0, 1) = \{q_1\}$

$\delta (q_1, 1) = \{q_2\}$

$\delta (q_2, 1) = \mathbf{\{q_1\}}$

Here **{q1}** is the final state. Hence the given input string 0111 is accepted by DFA.

46. Define Equivalence Relation? (NOV 2013)

A relation R that is reflexive, symmetric, and transitive is said to be an equivalence relation. An important property of an equivalence relation R on a set S is that R partitions S into disjoint non-empty equivalence classes. That is $S=S_1 \cup S_2 \cup \ldots$, where for each i and j, with $i \neq j$:

1. $S_i \cap S_j = \varphi$
2. For each a and b: n Si, aRb is true;
3. For each a in Si and b is Sj,aRb is false.

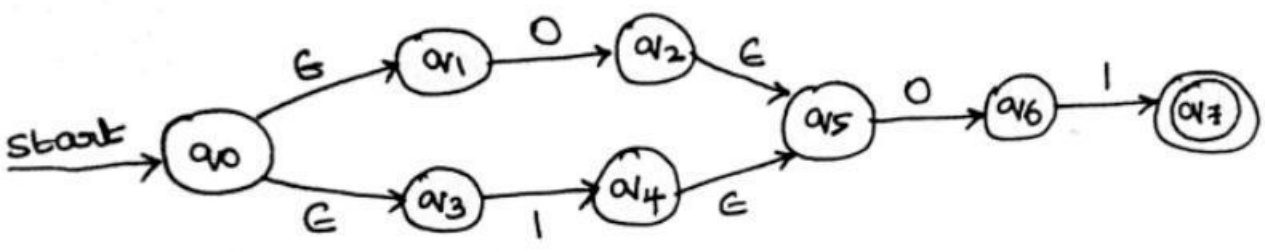
The S_i's are called ***equivalence classes.***

47. **What are the applications of automata theory?**

1. In compiler construction.
2. In switching theory and design of digital circuits.
3. To verify the correctness of a program.
4. Design and analysis of complex software and hardware systems.
5. To design finite state machines such as Moore and mealy machines.

48. Construct NFA equivalent to the regular expression: (0+1)01. (APR 2015)

The NFA equivalent for the given Regular Expression is (0+1) 01



11 MARKS

1. Explain the Operations on strings or Formal Languages or Finite Automata or Automata Theory ? (11 marks)

The operations on Strings are

- **i.** Length of string
- **ii.** Empty or null string
- **iii.** Concatenation of string
- **iv.** Reverse of a string
- **v.** Powers of an alphabet
- **vi.** Substring
- **vii.** Reversal
- **viii.** Kleene Closure

Alphabets:

An alphabet is a finite, non-empty set no of symbols. It is denoted by Σ.

Example:

- ✓ Σ = {*a, b*}, an alphabet of 2 symbols *a* and *b.*
- ✓ Σ = {*0, 1, 2*} an alphabet of 3 symbols *0, 1*, and *2.*

Strings:

A String or Word is a finite sequence of symbols from Σ. The group of characters also referred as a String.

Example:

- ✓ *a, b,* and *c* are symbols and *abcb* is a string.
- ✓ If Σ = {*a,b*} then *abab,aabbb,aaabb,……* are all strings over the alphabet Σ ={*a,b*}.
- ✓ *0, 1, 11, 00,* and *01101* are strings over {0, 1}.

Substring:

Let *x* and *y* be strings over an alphabet Σ.

The string *x* is a substring of *y* if there exist strings *w* and *z* over Σ such that *y = w x z.*

- ε is a substring of every string.
- For every string *x*, *x* is a substring of *x* itself.

Example:

- ε, compute and computation are substrings of computation.

Reversal:

Let *x* be a string over an alphabet N.The reversal of the string *x*, denoted by *x r*, is a string such that

- if *x* is ε, then *xr* is ε.
- If *a* is in Σ, *y* is in Σ* and *x* = *a y*, then *xr* = *yr a*.

(*automata*)*r* =

= (*utomata*)*r a*

= (*tomata*)*r ua*

= (*omata*)*r tua*

= (*mata*)*r otua*

= (*ata*)*r motua*

= (*ta*)*r amotua*

= (*a*)*r tamotua*

= (ε)*r atamotua*

- = atamotua The set of strings created from any number (0 or 1 or …) of symbols in an alphabet Σ is denoted by Σ*.

Length of string:

Let w is a string. Length of the string is |w|=> number of symbols composing the string.

Example:

1) If w=abcd then |w|=4

2) If x=01010110 the |x|=8

3) If y= 0101 the |y|=4

4) If |Є| = 0

Empty or null string:

The string consisting of zero symbols. It is denoted by Є.

Example:

{Є}=0.

Concatenation of string

Appending the strings (2) referred as concatenation of strings.

i.e w=a1,a2,….am

v=b1,b2,b3,…….bn

then wv=a1,a2,….am b1,b2,…..bn

Example:

1. x=00 y=1

xy=001, yx=100

x=AL y=GOL

xy=ALGOL

Reverse of a string

The reverse of a string is obtained by writing the symbols in reverse order.

Let w is the string

Reverse is w^R

i.e w=a1,a2,……am

w^R=am,……a2,a1

Example:

Let u=010111

u^R=1101010

Powers of an alphabet:

Let Σ be the alphabet

Σ * is the set of all strings over alphabet Σ.

Σ^m denotes the set of all strings over alphabet Σ of length m.

Example:

- ✓ Σ = {0, 1}
- ✓ Σ^0= {Є} empty string of length 0.
- ✓ Σ^1= {0, 1} is the set of all strings of length one over Σ = {0, 1}
- ✓ Σ^2= {00, 01, 10, 11} is the set of all strings of length two over Σ = {0, 1}.

Kleen Closure:

Let Σ be the alphabet

" Kleen Closure"-> Σ* denotes the set of all strings (Σ) over the alphabet

Example:

If Σ ={a} the Σ*={ Є,a,aa,aaa,…….}

If Σ ={0,1} the Σ*={ Є,0,1,00,10,10,……..}

Σ^0={ Є}

Σ^1={a}

Σ^2={aa}

Therefore Σ*= Σ^0U Σ^1U Σ^2,……..

2. Explain in detail Regular Expressions? (11 marks) (NOV 2013,MAY'15)

- The languages accepted by finite automata are easily described by simple expressions called regular expressions.
- Let Σ be a finite set of symbols and let L, L_1, and L_2 be sets of strings from Σ*.

The operations of regular expressions are

1. Concatenation
2. Kleene closure
3. Positive closure

- The ***concatenation*** of L_1 and L_2, denoted L_1L_2, is the set $\{xy \mid x \text{ is in } L_1 \text{ and } y \text{ is in } L_2\}$.That is, the strings in L_1L_2 are formed by choosing a string L_1 and following it by a string in L_2.
- The ***Kleene closure*** of L, denoted L*, is the set

$$L^* = \bigcup_{i=0}^{\infty} L^i$$

L^*={ €, a, aa, aaa, ...}

- The ***positive closure*** of L, denoted L^+, is the set

$$L^+ = \bigcup_{i=1}^{\infty} L^i$$

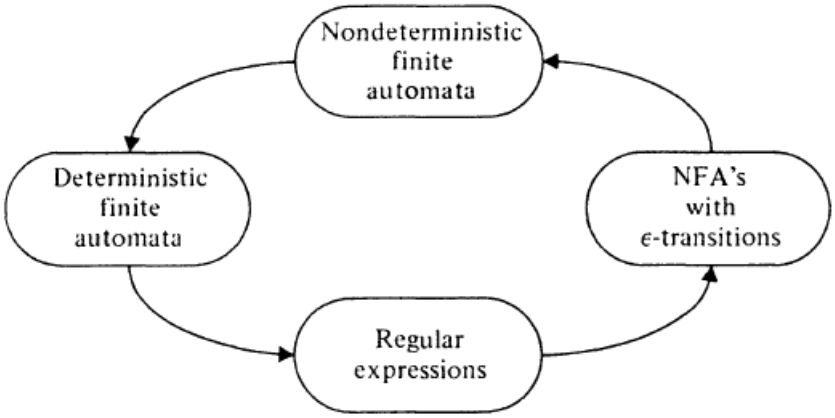
L^+ ={ a, aa, aaa, ...}

Let Σ be an alphabet. The regular expression over Σ and the sets that they denote are defined recursively as follows:

1. Φ is a regular expression and denotes the empty set.
2. ε is a regular expression and denotes the set { ε }.
3. For each a ϵ Σ, 'a' is a regular expression and denotes the set {a}.
4. If r and s are regular expressions denoting the languages R and S respectively then (r+s), (rs), and (r)* are regular expressions that denotes the sets RUS, RS and R* respectively.

Equivalence of finite automata and regular expressions:

The language accepted by finite automata are precisely the languages denoted by regular expressions. This equivalence for calling finite automaton languages regular sets.



Regular Expression to NFA with €: or Prove that for every regular expression there is an equivalent NFA: (NOV 2013)

Regular Expression to NFA with ∈

Theorem

Stmt:

Let R be the regular expression there exists a non-deterministic finite automata with ∈ transition that accepts language to regular exp.

Proof

n = 1 or 0

Basic

$\Sigma = \{\}$ $\quad r = \phi$

$\Sigma = \{a\}$ $\quad r = a$

$\Sigma = \{\varepsilon\}$ $\quad r = \varepsilon$

Induction: No. of operators more than one so languages must be more than 1 ($n \geq 1$)

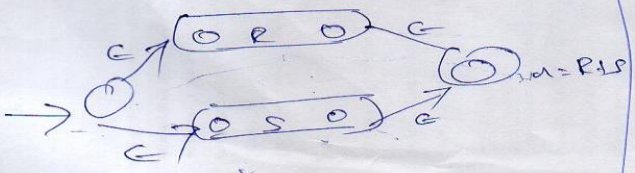
1. Union $(r+s)$ $(L_1 \cup L_2)$
2. Concatenation (rs) $(L_1 L_2)$
3. Closure r^* L_1^*

Case 1 $R+S \Rightarrow$ R and S

Start $\Rightarrow$ R (or) S.

So all accept automaton

Language is $L(R) \cup L(S)$



Case 2:

RS

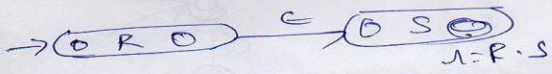
start state

accepting state

only one path from R to S.

2. exp

$L(R) \cdot L(S)$



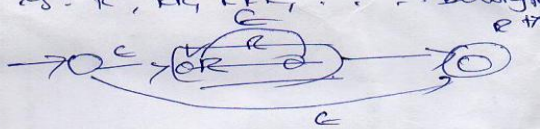
Case 3 R^*

allows either below 2

1. Directly from start state to accepting state along a path labled ∈, the ∈ also belong to R

2. To start state of automaton for R, through that automation one or more times to accepting state

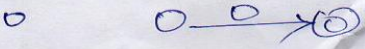
eg: R, RR, RRR, . . . belongs to R^*



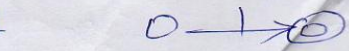
Eg: 1

$(0+1)^* 01$

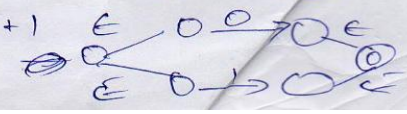
1. 0

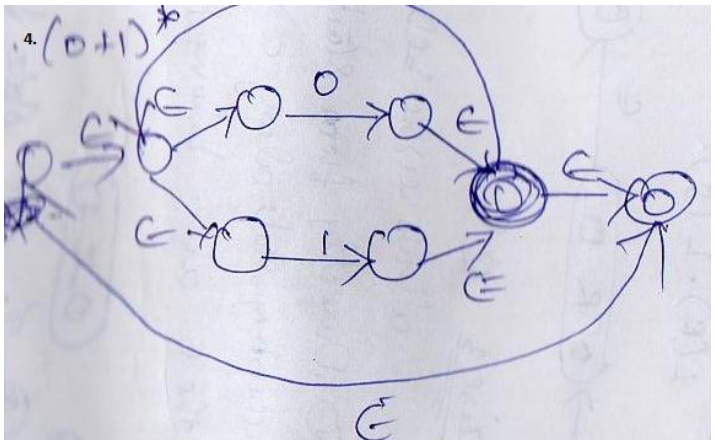


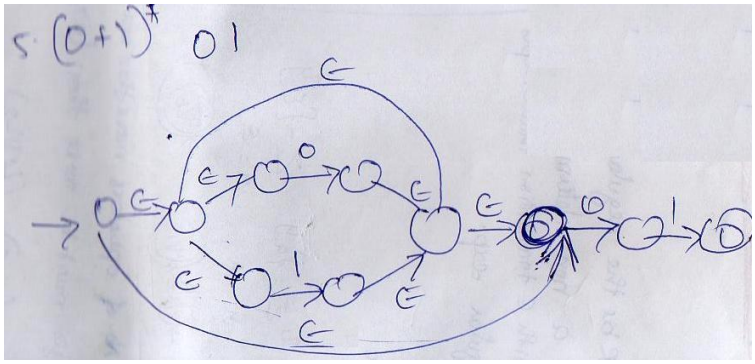
2. 1



3. 0+1







3. Explain Deterministic Finite Automata (DFA) with an example? (11 marks) (NOV 2013)

FINITE STATE SYSTEMS:

- Finite Automata is a mathematical model of a system that represents discrete number of inputs and outputs.
- The system can be in any one of a finite number of internal configurations or states.
- The state of the system summarizes the information concerning past inputs that is needed to determine the behavior of the system on subsequent inputs.
- The control mechanism of an elevator is a good example of a Finite State Machine (FSM).
- The commonly used programs such as text editors and the lexical analyzers found in most compilers are often designed as finite state systems.
- For example, a lexical analyzer scans the symbols of a computer programs to locate the string of characters corresponding to identifiers, numerical constants, reserved words etc.

FINITE AUTOMATA:

- A finite automaton (FA) consists of a finite set of states and a set of transitions from state to state that occur on input symbols chosen from an alphabet Σ.
- **DEFINITION:**

  M=(Q, Σ, δ,q,F)

  Where Q= set of finite state

  Σ= set of input alphabet

  δ= transition function from Q* Σ -> Q

  q_0 = start state

  F = set of final state
- Each input symbol there is exactly one transition out of each state (possibly back to the state itself).
- One State, usually q_0, is the initial state, in which the automaton starts.
- Some states are designated as final or accepting states.

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

Components (Model)of Finite Automata:

1. **Input tape:** divided into number of cells. Each cell is capable of holding one cell.
2. **Read Head:** reads one cell at time and moves ahead
3. **Finite Control:** it acts like CPU. Depending on current state and input symbol read from tape, it changes state.

How does an FA work?

- At the beginning,
  - ✓ an FA is in the start state (initial state)
  - ✓ its tape head points at the first cell
- For each move, FA
  - ✓ reads the symbol under its tape head
  - ✓ changes its state (according to the transition function) to the next state determined by the symbol read from the tape and its current state
  - ✓ move its tape head to the right one cell

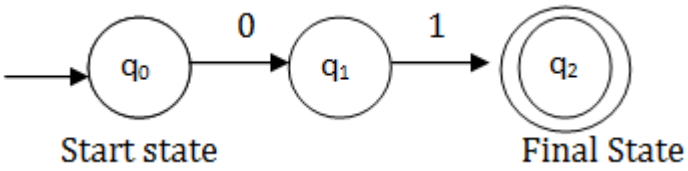
Two way of representation of Finite Automata:

1. Transition diagram
2. Transition table

Transition diagram:

A directed graph, called a transition diagram, is associated with an FA as follows:

- The vertices of the graph correspond to the states of the FA.
- If there is a transition from state *q* to state *p* on input *a,* then there is an arc labeled *a* from state *q* to state *p* in the transition diagram.
- The FA accepts a string *x* if the sequence of transitions corresponding to the symbols of *x* leads from the start state to an accepting state.



Transition table:

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

Input / State	a	b
q_0	q_1	q_0
q_1	-	q_2

The finite automata are of two types;

1. Deterministic Finite Automata (DFA)
2. Non-Deterministic Automata (NFA)

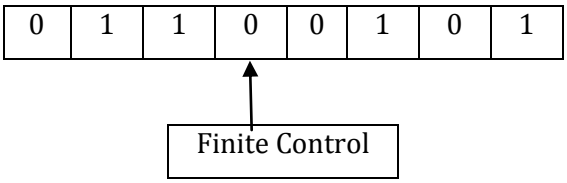
DETERMINISTIC FINITE AUTOMATA (DFA):

Deterministic Finite Automata (DFA) is generally specified by a 5 tuples **(Q, Σ, δ, q_0, F)**

where

1. **Q** is a finite set of states
2. **Σ** is a finite input alphabet
3. **q_0** in Q is the start state or initial state
4. **F**⊆Q is the set of final states
5. **δ** is the transition function mapping **δ: QX Σ→Q.**(i.e.) δ(q,a) is a state for each state *q* and input symbol *a*. (i.e.) δ(q,a) is a state for each state *q* and input symbol *a*.

- An FA as finite control, which is in some state from Q, reading a sequence of symbols from Σ written on tape.



A finite automaton as a machine can also be thought of as the device shown below consisting of a tape and a control circuit which satisfy the following conditions:

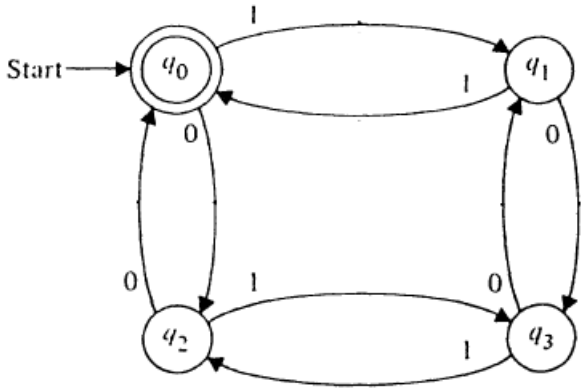
1. The tape has the left end and extends to the right without an end.
2. The tape is divide into squares in each of which a symbol can be written prior to the start of the operation of the automaton.
3. The tape has a read only head.
4. The head is always at the leftmost square at the beginning of the operation.
5. The head moves to the right one square every time it reads a symbol. It never moves to the left. When it sees no symbol, it stops and the automaton terminates its operation.
6. There is a finite control which determines the state of the automaton and also controls the movement of the head.

- A string x is said to be accepted by a finite automaton M= (Q, Σ, δ, q_0, F) if δ (q, x) =p for some p in F.
- The language accepted by M, designated L (M), is the set {x| δ (q_0, x) is in F}.
- A language is a regular set if it's the accepted by some finite automaton.

Example:

Consider the formal notation this FA is denoted M= (Q, Σ, δ, q_0, F), where Q= {q0, q1, q2, q3}, Σ= {0, 1}, F= {q0} and δ is shown as

Transition Diagram:



Transition Table:

	States	Inputs	
		0	**1**
→*	q0	q2	q1
	q1	q3	q0
	q2	q0	q3
	q3	q1	q2

Check whether given input string 110101 are accepted or not.

Solution:

The given input string is 110101 to machine M. We denote that

δ (q0, 1) = {q1}

δ (q1, 1) = {q0}

δ (q0, 0) = {q2}

δ (q2, 1) = {q3}

δ (q3, 0) = {q1}

δ (q1, 1) = **{q0}**

Therefore **q0** is the accepting or final state.

Hence the given input string 110101 is accepted by DFA.

4. Explain Non-Deterministic Finite Automata (NFA) with an example? (11 marks)(APR 2014)

- Consider modifying the finite automaton model to allow zero, one, or more transitions from a state on the same input symbol. This new model is called a Non-Deterministic Finite Automata (NFA).
- Non-deterministic finite automaton is a useful concept in proving theorems.
- NFA plays a central role in both the theory of languages and the theory of computation.
- Any set accepted by a NFA can also accepted by a DFA.

NFA Mathematical Specification:

Non-Deterministic Finite Automata (NFA) is generally specified by a 5 tuples **(Q, Σ, δ, q_0, F)** where

1. **Q** is a finite set of states
2. **Σ** is a finite input alphabet
3. **q_0** in Q is the start state or initial state
4. **F⊆Q** is the set of final states
5. **δ** is the transition function mapping **δ → Q x Σ* to 2^Q.**

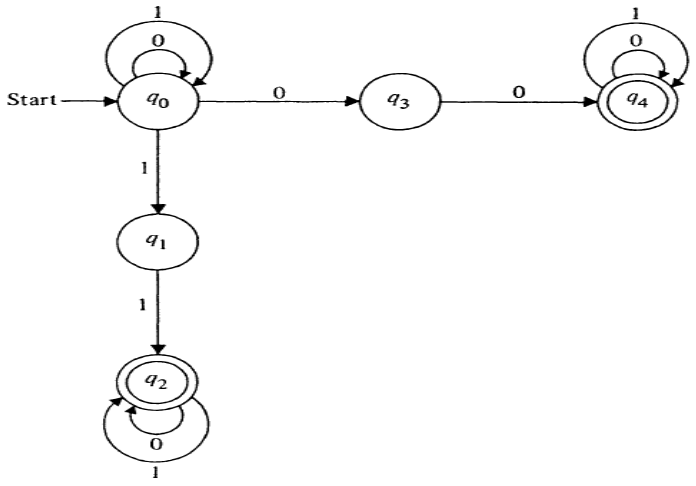
On receiving same inputs it goes to many states

- The machine can move without consuming any symbols and sometimes there are possible moves and sometimes there are move then one possible moves.
- The state is only partially determined by the current state and i/p symbol.

Examples:

Consider the given NFA to check whether the string is accepted or not W=01001.

Transition diagram:



Solution :

In Non-Deterministic Finite Automata (NFA), Consider a machine M= (Q, Σ, δ, q0, F),

where,

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

Σ = {0, 1}

{q0}=initial state

{q2, q4}=Final state

Transition table:

	States	Input	
		0	**1**
→	q0	{q0,q3}	{q0,q1}
	q1	Φ	{q2}
*	q2	{q2}	{q2}
	q3	{q4}	Φ
*	q4	{q4}	{q4}

Method1:

The given input string is 01001

δ (q0, 0) = {q0, q3}

δ (q0, 01) = δ (δ (q0, 0), 1)
= δ ({q0, q3}, 1)
= δ (q0, 1) U δ (q3, 1)
= {q0, q1} U {Φ}
= {q0, q1}

δ (q0, 010) = δ (δ (q0, 01), 0)
= δ ({q0, q1}, 0)
= δ (q0, 0) U δ (q1, 0)
= {q0, q3} U {Φ}
= {q0, q3}

δ (q0, 0100) = δ (δ (q0, 010), 0)
= δ ({q0, q3}, 0)
= δ (q0, 0) U δ (q3, 0)
= {q0, q3} U {q4}
= {q0, q3, q4}

δ (q0, 01001) = δ (δ (q0, 0100), 1)
= δ ({q0, q3, q4}, 1)
= δ (q0, 1) U δ (q3, 1) U δ (q4, 1)
= {q0, q1} U {Φ} U {q4}
= {q0, q1, **q4**}

The given input string 010001 is accepted by machine. Therefore **q4** is the final state or accepting state.

5. Explain about Finite Automata with Epsilon (ε) – moves with an example? (11 marks)

- In our model of the non-deterministic finite automaton to include transitions on the empty input ε.

- The transition diagram for NFA accepting the language consisting of any number (including zero) of 0's followed by any number of 1's followed by any number of 2's.
- NFA with ε move is defined as

  $(Q, \Sigma, \delta, q_0, F)$ where

  - **Q** is a finite set of states
  - **Σ** is a finite input alphabet
  - **q_0** in Q is the start state or initial state
  - **$F \subseteq Q$** is the set of final states
  - **δ** is the transition function mapping **$\delta \rightarrow Q \times \Sigma \cup \Sigma$ to 2^Q.**
- An NFA accepts a string ω if there is some path labeled ω from the initial state to a final state.
- The edges labeled ε may be included in the path, although the ε's do not appear explicitly in ω.

1. EPSILON CLOSURE
2. EXTENDED TRANSITION FUNCTION
3. LANGUAGE ACCEPTED BY NFA WITH ε MOVE
4. ELIMINATING ε TRANSITIONS
5. EQUIVALANCE OF NFA WITH ε MOVES TO DFA

1.11.1. Epsilon Closure:

Informally ε-closure of a state is all transitions out of that state are labeled ε. ε-closure (q) is set of all states which are reachable from q on ε-transitions only.

Example: Find ε-closure of all states.

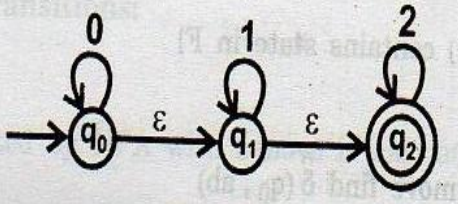


Fig. 1.30

Solution:

ε – closure $(q_0) = \{ q_0, q_1, q_2 \}$

ε – closure $(q_1) = \{ q_1, q_2 \}$

ε – closure $(q_2) = \{ q_2 \}$

Example 2:

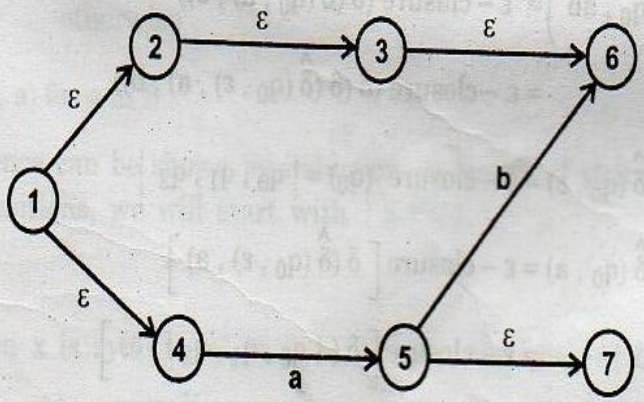


Fig. 1.31

Find ε – closure (1), ε – closure (2) and ε – closure (5).

Solution:

ε – closure (1) = { 1, 2, 3, 4, 6 }

ε – closure (2) = { 2, 3, 6 }

ε – closure (5) = { 5, 7 }.

1.11.2. Extended Transition Function $\hat{\delta}$: for ε-closure

$\hat{\delta}$ is defined as follows:

(i) $\hat{\delta}(q, xa) = \varepsilon$ closure (R), where R is set of states that are reachable while processing xa.

(ii) $\delta(R, a) = \bigcup_{q \text{ in } R} \delta(q, a)$

(iii) $\hat{\delta}(R, \omega) = \bigcup_{q \text{ in } R} \hat{\delta}(q, \omega)$

1.11.3. Language accepted by NFA with ε move:

It is defined as,

$L(M) = \{ w / \hat{\delta}(q_0, w) \text{ contains state in } F \}$

Example:

Consider NFA with ε - move find $\hat{\delta}(q_0, ab)$

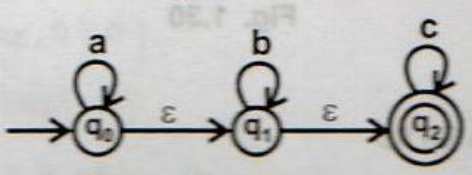


Fig. 1.32

Solution:

$$\hat{\delta}(q_0, ab) = \varepsilon - \text{clasure}(\delta(\hat{\delta}(q_0, a), b))$$

$$= \varepsilon - \text{closure}(\delta(\delta(\hat{\delta}(q_0, \varepsilon), a), b))$$

$$\hat{\delta}(q_0, \varepsilon) = \varepsilon - \text{closure}(q_0) = \{ q_0, q_1, q_2 \}$$

$$\hat{\delta}(q_0, a) = \varepsilon - \text{closure}[\delta(\hat{\delta}(q_0, \varepsilon), a)]$$

$$= \varepsilon - \text{closure}[\delta(\{ q_0, q_1, q_2 \}, a)]$$

$$= \varepsilon - \text{closure}(\delta(q_0, a) \cup \delta(q_1, a) \cup \delta(q_2, a))$$

$$= \varepsilon - \text{closure}(\{ q_0 \} \cup \phi \cup \phi)$$

$$= \varepsilon - \text{closure}[q_0]$$

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

$$\hat{\delta}(q_0, ab) = \varepsilon - \text{closure}\,[\delta(\hat{\delta}(q_0, a), b)]$$

$$= \varepsilon - \text{closure}\,[\delta(\{q_0, q_1, q_2\}, b)]$$

$$= \varepsilon - \text{closure}\,[\delta(q_0, b) \cup \delta(q_1, b) \cup \delta(q_2, b)]$$

$$= \varepsilon - \text{closure}\,[\phi \cup \{q_1\} \cup \phi]$$

$$= \varepsilon - \text{closure}\,(q_1)$$

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

$\therefore \hat{\delta}(q_0, qb) = \{q_1, q_2\}$ which contains a state in set of final states. Hence string is accepted.

1.11.4. Eliminating ε-Transitions:

Theorem:

Any language accepted by NFA with ε-move is also accepted by NFA without ε-move.

Proof:

Let $M = \{Q, \Sigma, \delta, q_0, F\}$ be an NFA with ε-transitions.

Construct $M' = (Q, \Sigma, \delta', q_0, F')$

Where

$$F' = \begin{cases} F \cup \{q_0\} & \text{if } \varepsilon - \text{colsure}\,(q_0) \text{ contains a state in F} \\ F & \text{otherwise} \end{cases}$$

and $\delta'(q, a) = \bar{\delta}(q, a)$ for a in Σ

The equivalence can be shown by induction on length of string x. Since M′ does not contain ε-transitions, we will start with $|x| = 1$.

Basis:

$|x| = 1$, then x is symbol "a" in Σ and $\delta'(q_0, a) = \bar{\delta}(q_0, a)$ by definition of δ'.

Induction:

$|x| > 1$ Let $x = wa$ for symbol a in Σ and w in Σ^*.

$$\delta'(q_0, wa) = \delta'(\delta'(q_0, w), a)$$

By inductive hypothesis

$$\delta'(q_0, w) = \hat{\delta}(q_0, w)$$

Let $\hat{\delta}(q_0, w) = R$ we must show that $\delta'(R, a) = \hat{\delta}(q_0, wa)$.

But $\delta'(R, a) = \bigcup_{q \text{ in } R} \hat{\delta}(q, a) = \bigcup_{q \text{ in } R} \hat{\delta}(q, a)$

Then, $R = \bar{\delta}(q_0, wa)$ we have

$\bigcup_{q \text{ in } R} \bar{\delta}(q, a) = \hat{\delta}(q_0, wa)$ by second rule of definition of $\bar{\delta}$

Thus $\delta'(q_0, w) = \hat{\delta}(q_0, wa)$

If $\delta'(q_0, x)$ contains state in F then string is accepted.

If $\hat{\delta}(q_0, x)$ contains state in F, then

$\delta'(q_0, x)$ also contains

Example:

Construct NFA without ε-move from given NFA with ε?

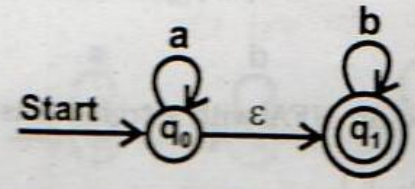


Fig. 1.33

Find ε-closure $(q_0) = \{q_0, q_1\}$

Since ε-closure (q_0) contains final state q_1, add q_0 also in set of final states for NFA without ε-moves.

$$F' = \{q_0, q_1\}$$

$$\hat{\delta}(q_0, \varepsilon) = \varepsilon - \text{closure}(q_0) = \{q_0, q_1\}$$

$$\delta'(q_0, a) = \hat{\delta}(q_0, a)$$

$$\therefore \hat{\delta}(q_0, a) = \varepsilon - \text{closure}(\delta(\hat{\delta}(q_0, \varepsilon), a))$$

$$= \varepsilon - \text{closure}(\delta(\{q_0, q_1\}, a))$$

$$= \varepsilon - \text{closure}(\delta(q_0, a) \cup \delta(q_1, a))$$

$$= \varepsilon - \text{closure}(q_0 \cup \phi)$$

$$= \varepsilon - \text{closure}(q_0)$$

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

$$\delta'(q_0, b) = \hat{\delta}(q_0, b)$$

$$\therefore \hat{\delta}(q_0, b) = \varepsilon - \text{closure}\left(\delta\left(\hat{\delta}(q_0, \varepsilon), b\right)\right)$$

$$= \varepsilon - \text{closure}\left(\delta\left(\{q_0, q_1\}, b\right)\right)$$

$$= \varepsilon - \text{closure}\left(\delta(q_0, b) \cup \delta(q_1, b)\right)$$

$$= \varepsilon - \text{closure}\left(\phi \cup q_1\right)$$

$$= \varepsilon - \text{closure}(q_1)$$

$$= \{q_1\}$$

$$\delta'(q_1, a) = \hat{\delta}(q_1, a)$$

$$\therefore \hat{\delta}(q_1, a) = \varepsilon - \text{closure}\left(\delta\left(\hat{\delta}(q_1, \varepsilon), a\right)\right) \left[\because \hat{\delta}(q_1, \varepsilon) = \varepsilon - \text{closure}(q_1) = \{q_1\}\right]$$

$$= \varepsilon - \text{closure}\left(\delta\left(\{q_1\}, a\right)\right)$$

$$= \varepsilon - \text{closure}\left(\delta(q_1, a)\right)$$

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

$$= \phi$$

$$\delta'(q_1, b) = \hat{\delta}(q_1, b)$$

$$\therefore \hat{\delta}(q_1, b) = \varepsilon - \text{closure}\left(\delta\left(\hat{\delta}(q_1, \varepsilon), b\right)\right)$$

$$= \varepsilon - \text{closure}\left(\delta\left(\{q_1\}, b\right)\right)$$

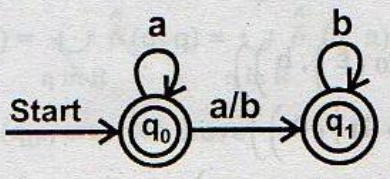
$$= \varepsilon - \text{closure}\left(\delta(q_1, b)\right)$$

$$= \varepsilon - \text{closure}(q_1)$$

$$= \{q_1\}$$

δ can be represented by

State	I/P	
	a	b
q_0	$\{q_0, q_1\}$	$\{q_1\}$
q_1	ϕ	$\{q_1\}$



$\left[\because F' = \{ q_0 , q_1 \} \right]$

Fig. 1.34

1.11.5. Equivalence of NFA with ε-Moves to DFA:

Let $M' = \left(Q' , \Sigma , \delta' , q_0^1 , F' \right)$ be NFA with ε-move.

Where,

Q' = Set of states

Σ = Set of input alphabets

δ' = Transition Function $Q' \times \Sigma \cup \{ \varepsilon \} \rightarrow 2^Q$

q_0 = Start State

F' = Set of final state

The equivalent DFA M can be

$M = (Q, \Sigma, \delta, q_0, F)$ obtained as

Step 1: $q_0 = \varepsilon\text{-closure } (q_0')$

The start state of DFA can be obtained as taking ε-closure of start state of NFA with ε.

Step 2: A state of DFA is represented by $\left[t_1, t_2, \ldots t_k \right] \in Q$ by applying following rule.

$$\delta \left(\left[t_1 , t_2 , \ldots . t_k \right] , a \right) = \varepsilon - \text{closure} \left(\delta' (t_1 , a) \cup \delta' \left(t_2 , a \right) \right) \cup \ldots . \delta' \left(t_k , a \right)$$

$$= \bigcup_{i=1}^{k} \varepsilon - \text{closure} \left(\delta' \left(t_i , a \right) \text{for each } a \in \Sigma \right)$$

Step 3: If $\left[t_1 , t_2 , \ldots . t_k \right] \in Q$ and if any of state $t_i \in F'$. Then $\left[t_1 , t_2 , \ldots t_k \right]$ is final slate DFA.

Example: Convert following NFA to DFA.

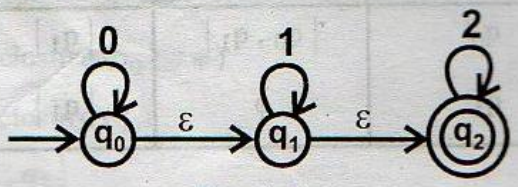


Fig. 1.35

Solution:

The state q_0 is start state of NFA with ε transition.

The start state of DFA can be obtained by applying first rule.

ε-closure $(q_0) = \{ q_0, q_1, q_2 \}$

By using state $[q_0\, q_1\, q_2]$, we can find remaining state on inputs 0, 1, 2.

$[q_0\, q_1\, q_2]$ – Start State

On Input 0:

$$\delta(\{q_0, q_1, q_2\}, 0) = \varepsilon - \text{closure}\left(\delta(q_0, 0) \cup \delta(q_1, 0) \cup \delta(q_2, 0)\right]$$

$$= \varepsilon - \text{closure}[\, q_0 \cup \phi \cup \phi \,]$$

$$= \varepsilon - \text{closure}[\, q_0 \,]$$

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

On input 1:

$$\delta(\{q_0, q_1, q_2\}, 1) = \varepsilon - \text{closure}\left(\delta(q_0, 1) \cup \delta(q_1, 1) \cup \delta(q_2, 1)\right)$$

$$= \varepsilon - \text{closure}\left(\phi \cup q_1 \cup \phi\right)$$

$$= \varepsilon - \text{closure}\left(q_1\right)$$

$= \{ q_1, q_2 \}$ since it is New state

On Input 2:

$$\delta(\{q_0, q_1, q_2\}, 2) = \varepsilon - \text{closure}\left(\delta(q_0, 2) \cup \delta(q_1, 2) \cup \delta(q_2, 2)\right)$$

$$= \varepsilon - \text{closure}\left(\phi \cup \phi \cup q_2\right)$$

$$= q_2$$

New states reachable from $[q_0\, q_1\, q_2]$ is $[q_1\, q_2]$

∴ The transition function for these states are,

$[q_1\, q_2]$ = new state

On Input 0:

$$\delta\left(\{q_1, q_2\}, 0\right) = \varepsilon - \text{closure}\left(\delta\left(q_1, 0\right) \cup \delta\left(q_2, 0\right)\right)$$

$$= \varepsilon - \text{closure}\left(\phi \cup \phi\right) = \phi$$

On Input 1:

$$\delta\left(\{q_1, q_2\}, 1\right) = \varepsilon - \text{closure}\left(\delta\left(q_1, 1\right) \cup \delta\left(q_2, 1\right)\right)$$

$$= \varepsilon - \text{closure}\left(q_1 \cup \phi\right)$$

$$= \varepsilon - \text{closure}\left(q_1\right)$$

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

On Input 2:

$$\delta\left(\{q_1, q_2\}, 2\right) = \varepsilon - \text{closure}\left(\delta\left(q_1, 2\right) \cup \delta\left(q_2, 2\right)\right)$$

$$= \varepsilon - \text{closure}\left(\phi \cup q_2\right)$$

$$= \varepsilon - \text{closure}\left(q_2\right)$$

$$= q_2$$

As here are no more new states.

So the states are,

$[q_0\ q_1\ q_2]$ – Start State

$[q_1\ q_2]$

$[q_2]$

The final state is the state which consists of final state of NFA with ε.

$F = \{[q_0\ q_1\ q_2], [q_1\ q_2], [q_2]\}$

DFA:

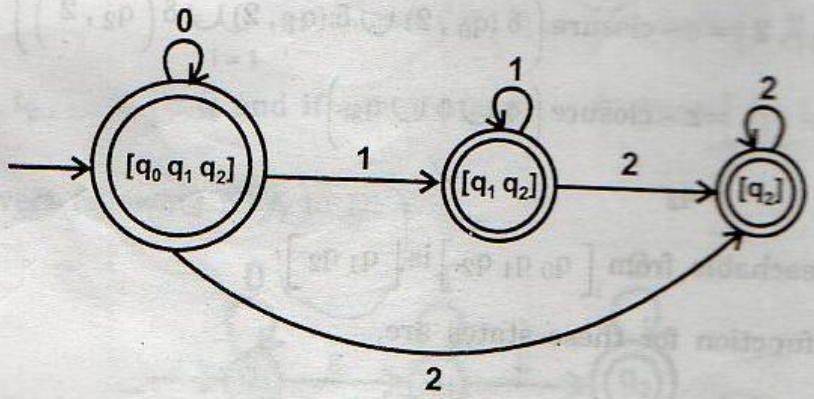


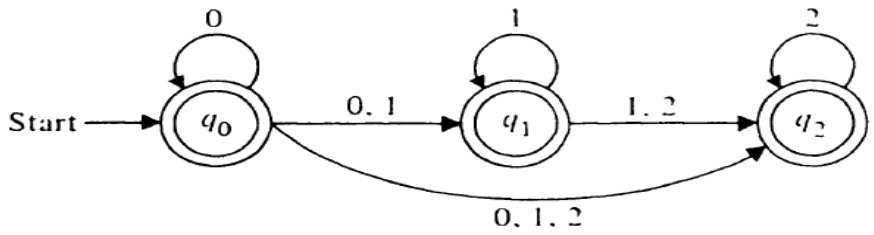
Fig. 1.36

6. Construct a DFA equivalent to NFA, where M = ({q0, q1, q2}, {0, 1, 2}, δ, q0, {q0, q1, q2}).
The Transition is given by the transition diagram and transition table. (11 marks)

Transition Table:

	States	Inputs		
		0	**1**	**2**
→*	q0	{q0,q1,q2}	{q1,q2}	{q2}
*	q1	{φ}	{q1,q2}	{q2}
*	q2	{φ}	{φ}	{q2}

Transition Diagram:



CONSTRUCTING DFA:

Solution:

The equivalent DFA is M' = (Q, {0, 1, 2}, δ', S', F')

Q = {q0, q1, q2}

∑ = {0, 1, 2}

Q' = 2^Q

S'= [q0] Є Q'

All subsets = { [φ],[q0],[q1],[q2],[q0,q1,q2],[q0q1],[q0q2],[q1q2][q0q1q2]}

Transition Function:

δ'(q0,0) = [q0q1q2]

δ'(q0,1) = [q1q2]

δ'(q0,2) = [q2]

δ'(q1,0) = [φ]

δ'(q1,1) = [q1q2]

δ'(q1,2) = [q2]

δ'(q2,0) = [φ]

δ'(q2,1) = [φ]

δ'(q2,2) = [q2]

δ'(q0q1,0) = δ'(q0,0) U δ'(q1,0)
= [q0, q1, q2] U {φ}
= [q0, q1, q2]

δ'(q0q1,1) = δ'(q0,1) U δ'(q1,1)
= [1q2] U [q1q2]
= [q1, q2]

δ'(q0q1,2) = δ'(q0,2) U δ'(q1,2)
= [q2] U [q2]
= [q2]

δ'(q0q2,0) = δ'(q0,0) U δ'(q2,0)
= [q0q1q2] U {φ}
= [q0q1q2]

δ'(q0q2,1) = δ'(q0,1) U δ'(q2,1)
= [q1q2] U {φ}
= [q1q2]

δ'(q0q2,2) = δ'(q0,2) U δ'(q2,2)
= [q2] U [q2]
= [q2]

δ'(q1q2,0) = δ'(q1,0) U δ'(q2,0)
= {φ} U {φ}
= [φ]

δ'(q1q2,1) = δ'(q1,1) U δ'(q2,1)
= [q1q2] U {φ}
= [q1q2]

δ'(q1q2,2) = δ'(q1,2) U δ'(q2,2)
= [q2] U [q2]
= [q2]

δ'(q0q1q2,0) = δ'(q0,0) U δ'(q1,0) U δ' (q2,0)
= {q0q1q2} U {φ} U {φ}
= [q0q1q2]

δ (q0q1q2,1) = δ'(q0,1) U δ'(q1,1) U δ' (q2,1)
= [q1q2] U [q1q2] U {φ}
= [q1q2]

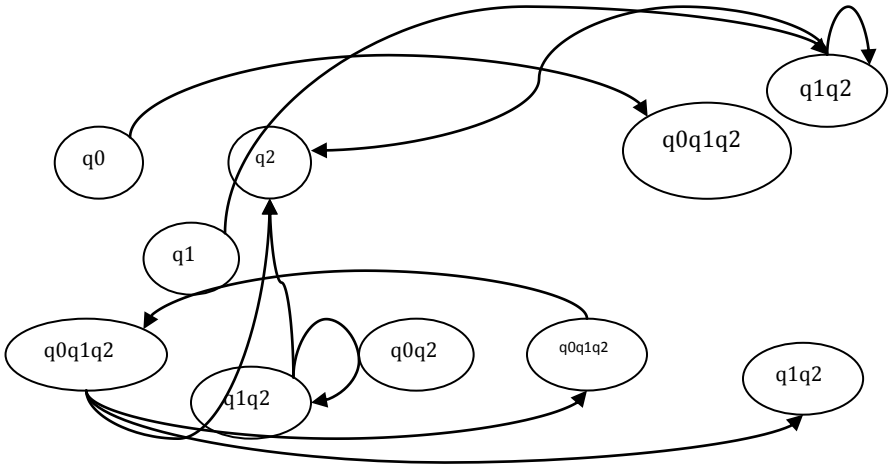
δ'(q0q1q2,2) = δ'(q0,2) U δ'(q1,2) U δ' (q2,2)
= [q2] U [q2] U [q2]
= [q2]

Transition Table for DFA:

States	Inputs		
	0	**1**	**2**
[q0]	[q0q1q2]	[q1q2]	[q2]
[q1]	φ	[q1q2]	[q2]
[q2]	φ	φ	[q2]
[q0q1]	[q0q1q2]	[q1q2]	[q2]
[q0q2]	[q0q1q2]	[q1q2]	[q2]
[q1q2]	φ	[q1q2]	[q2]
[q0q1q2]	[q0q1q2]	[q1q2]	[q2]

Final States are {[q2], [q1q2], and [q0q1q2]}

Transition Diagram:



7. Explain the Equivalence of NFA and DFA

1.10. EQUIVALENCE OF NFA TO DFA:

1.10.1. Language of NFA:

A language L is accepted by NFA is defined as L(M).

$L(M) = \{ w / w \in \Sigma^* \text{ and } \bar{\delta}(q_0, w) = Q \text{ with at least one element of } Q \text{ is in } F \}$

$L(M) = \{ w / \hat{\delta}(q_0, w) \cap F \neq \phi \}$

1.10.2. Theorem:

Statement: Let L be set accepted by NFA then there exists a DFA that accepts same language.

Proof:

Let $M = (Q, \Sigma, \delta, q_0, F)$ be NFA for language L.

Then define DFA as

$$M' = (Q', \Sigma', \delta', q'_0, F')$$

The states of DFA M′ are all subsets of M.

$$Q' = 2^Q$$

F′ be set of all finite states in M. The elements in Q′ will be denoted by $[q_1, q2, ... q_i]$ and elements in Q are denoted by $\{ q_1, q_2, ... q_i \}$.

$\therefore [q_1 q_2, ... q_i]$ is considered as single state for DFA.

To Prove:

$$\delta'(q'_0, x) = \left[q_1, q_2, ... q_i \right] \text{ for DFA}$$

$$\text{iff } \delta(q_0, x) = \{ q_1, q_2, ... q_i \} \text{ for NFA}$$

If we assume that hypothesis proving of input string of length m or less than m then if xa is string of length m + 1.

The function δ′ is

$$\delta'\left(q'_0, xa \right) = \delta'\left(\delta'(q'_0, x), a \right) \quad .... (1)$$

By induction hypothesis,

$$\delta'\left(q'_0, x\right) = \left[p_1, p_2 \ldots p_j\right] \quad \ldots. (2)$$

iff $\delta(q_0, x) = \{p_1, p_2, \ldots. p_j\}$ .... (2)

By definition of δ' substitute equation (2) in (1),

$$\delta'\left([p_1 p_2 \ldots p_j], a\right) = \left[r_1, r_2, \ldots r_k\right] \text{ in DFA}$$

iff $\delta(\{p_1, p_2, \ldots p_j\}, a) = \{r_1, r_2, \ldots r_k\}$ in NFA

Thus

$$\delta'\left(q'_0, xa\right) = \left[r_1, r_2 \ldots r_k\right]$$

iff $$\delta\left(q_0, xa\right) = \{r_1, r_2 \ldots. r_k\}$$

Example 1:

Let NFA $M = \left(\{q_0, q_1\}, \{0, 1\}, \delta, q_0, \{q_1\}\right)$ where δ is

Input / State	0	1
q_0	$\{q_0, q_1\}$	$\{q_1\}$
q_1	ϕ	$\{q_0, q_1\}$

$Q = \{q_0, q_1\}$

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

$q_0 = \{q_0\}$

$F = \{q_1\}$

$Q' = 2^Q = \{\phi, [q_0], [q_1], [q_0, q_1]\}$

$\{\because Q = 2, 2^2 = 4\}$

$Q = 2$

$Q^1 = 2^2 = 4$

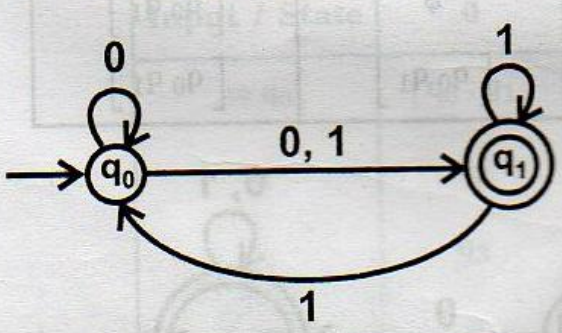


Fig. 1.26

To convert NFA to DFA

NFA	DFA
$\delta(q_0, 0) = \{q_0, q_1\}$	$\delta'([q_0], 0) = [q_0\ q_1]$ single state
$\delta(q_0, 1) = \{q_1\}$	$\delta'([q_0], 1) = [q_1]$
$\delta(q_1, 0) = \phi$	$\delta'([q_1], 0) = \phi$
$\delta(q_1, 1) = \{q_0, q_1\}$	$\delta'([q_1], 1) = [q_0\ q_1]$

$[q_0\ q_1]$ is a new state and hence find transitions on 0 and 1 for this new state.

$$\delta'([q_0\ q_1], 0) = [q_0\ q_1]$$

iff $\quad \delta(\{q_0, q_1\}, 0) = \delta(q_0, 0) \cup \delta(q_1, 0)$

$$= \{q_0\ q_1\} \cup \{\phi\}$$

$$= \{q_0\ q_1\}$$

$$\delta'([q_0\ q_1], 1) = [q_0\ q_1]$$

iff $\quad \delta(\{q_0, q_1\}) = \delta(q_0, 1) \cup \delta(q_1 = 1)$

$$= q_1 \cup \{q_0, q_1\}$$

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

No new states are formed and hence transition table is given by (for DFA):

δ is given by

Input / State	0	1
$[q_0]$	$[q_0, q_1]$	$[q_1]$
$[q_1]$	ϕ	$[q_0\ q_1]$
$[q_0\ q_1]$	$[q_0\ q_1]$	$[q_0\ q_1]$

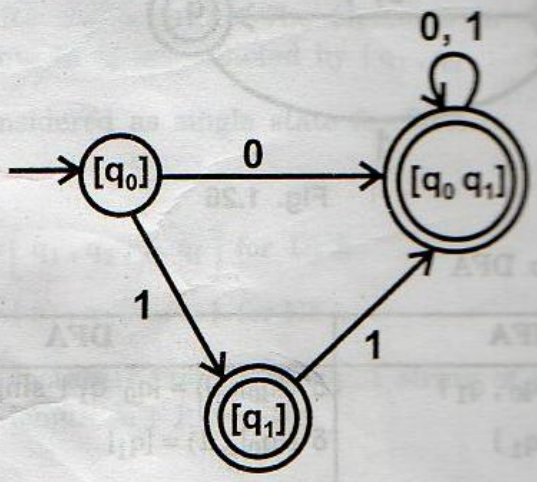


Fig. 1.27

The states which has NFA's final state are considered as final state in DFA.

States are

$$Q = \{[q_0], [q_1], [q_0\ q_1], \phi\}$$

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

$$q_0 = [q_0]$$

$$F = \{[q_1], [q_0\ q_1]\}$$

Take $[q_0]$ as A, $[q_1]$ as B, $[q_0\ q_1]$ as C.

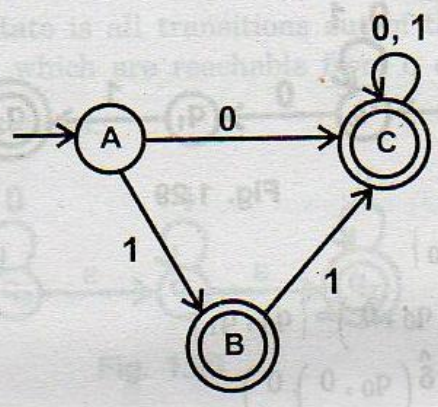


Fig. 1.28

Example 2:

Input / State	0	1
$\Rightarrow q_0$	$\{q_0, q_1\}$	q_0
q_1	q_2	q_1
q_2	q_3	q_3
(q_3)	ϕ	q_2

1.10.3. Extended Transition Function:

This is used represent transition functions with a string of input symbols 'ω' and returns a set of states. It is represented by $\hat{\delta}$.

$$\omega = xa.$$

where a is last symbol of ω and x may be rest of ω, also we have $\delta(q, x) = \{p_1, p_2, \ldots p_k\}$

Then,

$$\bigcup_{i=1}^{k} \hat{\delta}\left(p_i, a\right) = \left\{r_1, r_2, \ldots r_m\right\}$$

$$\hat{\delta}(q, \omega) = \left\{r_1, r_2, \ldots r_m\right\}$$

(or) $$\hat{\delta}(q, xa) = \hat{\delta}(\hat{\delta}(q, x), a)$$

Example:

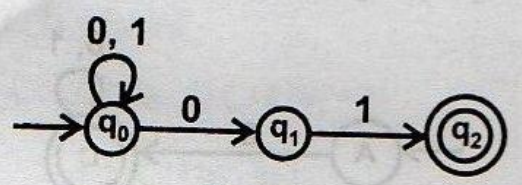


Fig. 1.29

$$\hat{\delta}(q_0, \varepsilon) = \left\{q_0\right\}$$

$$\hat{\delta}\left(q_0, 0\right) = \delta\left(q_0, 0\right) = \left\{q_0, q_1\right\}$$

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

$$= \delta\left(\left\{q_0, q_1\right\}, 0\right)$$

$$= \delta\left(q_0, 0\right) \cup \delta(q_1, 0)$$

$$= \left\{q_0, q_1\right\} \cup \phi$$

$$= \left\{q_0, q_1\right\}$$

$$\hat{\delta}\left(q_0, 001\right) = \delta\left(\hat{\delta}(q_0, 00), 1\right)$$

$$= \delta(\delta\left(\hat{\delta}\left(q_0, 0\right), 0\right), 1)$$

$$= \delta\left(\left\{q_0, q_1\right\}, 1\right\}$$

$$= \delta\left(q_0, 1\right) \cup \delta(q_1, 1)$$

$$= \left\{q_0\right\} \cup \left\{q_2\right\}$$

$$= \left\{q_0\ q_2\right\}$$

8. Explain Minimization of Finite Automata with an example? (11 marks)

DFA minimization is the task of transforming a given Deterministic Finite Automaton (DFA) into an equivalent DFA that has a minimum number of states. Here, two DFAs are called equivalent if they recognize the same regular language. Several different algorithms accomplishing this task are known and described in standard textbooks on automata theory.

A DFA can recognize a unit language. It is possible to construct a new DFA where it contains only minimal number of states. The central idea in minimizing the DFA, partition the states into blocks such

that all the states in the block are equivalent. No two states chosen from two different blocks are equivalent.

Property of Equivalent States:

A= (Q, Σ, δ, q0, F)

Equivalent states are transitive

P&Q⇒ equivalent A&B⇒equivalent

Q&R⇒equivalent B&C⇒equivalent

(P&R⇒ equivalent i.e. PR) (A&B⇒ equivalent i.e. AB)

Partitioning:

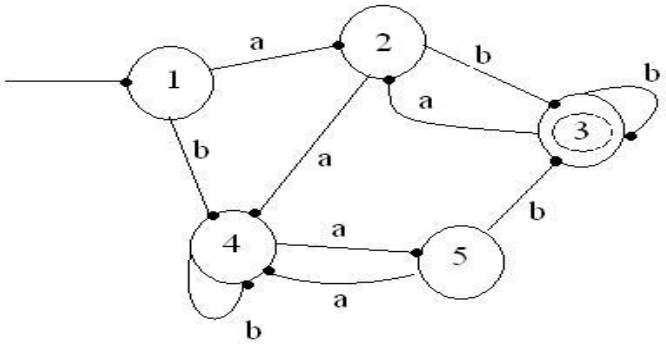
- Split Q into two parts or sets, the first set containing the final state and the second set containing the non-final state.

  If we create for each state

1. q [DFA], a block consisting of q, then the different blocks of stated form a partition of set of states.
2. Each state is in exactly one block.

Example:

Construct a minimized DFA from the equivalent the given DFA.



Given

Let Q = {1, 2, 3, 4, 5}

Σ = {a, b}

S = {1}

F = {3}

Split Q into two parts or sets, the first set containing the final state and the second set containing the non-final state.

Let A = {3}

B = {1, 2, 4, 5}

Then the list containing non-final state and according to the output of each transition, gives the set representation.

B	1	2	4	5
a	B	B	B	B

Then consider the set with a larger number of states

A = {3}

B = {1,4} equivalent

C = {2,5} non-equivalent

B	1	4
a	C	C
b	B	B

Transition for A

A	3
a	C
b	A

Transition for B

B	1	4
a	C	C
b	B	B

Transition for C

B	2	5
a	B	B
b	A	A

Transition Table:

STATES	**INPUTS**	
	a	b
A	C	A
B	C	B
C	B	A

Transition Diagram:

9. Explain two – way finite automata with an example? (11 marks) (APR 2012)

- A finite automaton is a control unit that reads a tape, moving one square right at each move. Two way finite automatons is an extension of one- way finite automaton which has the ability to allow the tape head to move left as well as right. Such a finite automaton is called a two-way finite automaton.
- Two ways finite automaton accepts the input state if it moves the tape head, off the right end of the tape, at the same time, entering an accepting state.
- A **two-way finite deterministic automaton (2DFA)** is a quintuple M= **$(Q, \Sigma, \delta, q_0, F)$**
  Where
  1. **Q** is a finite set (the states)
  2. **Σ** is a finite set (the input alphabet)
  3. **q_0** in Q is the start state or initial state
  4. **F⊆Q** is the set of final states
  5. **δ** is a map from **Q X Σ → Q X {L, R}.**

- If δ(q,a)=(p,L), then in state q, scanning input symbol a, the 2DFA enters state p and moves its head left one square.
- If δ(q,a)=(p,R), then in state q, scanning input symbol a, the 2DFA enters state p and moves its head right one square.

<u>Instantaneous Description (ID):</u>

- Instantaneous Description of 2DFA, which describes
  - ✓ the input string,
  - ✓ current state, and
  - ✓ current position of input head
- The relation $\vdash_M$ on ID's such that $I_1 \vdash_M I_2$ if and only if M can go from instantaneous description I_1 to I_2 in one move.
- An ID of M is a string in Σ*Q Σ*. The ID *wqx*, where *w* and x are in Σ* and *q* is a state in Q, represent the facts that
  1. *wx* is the input string
  2. q is the current state and

3. the input head is scanning the first symbol of *x.*

If x= ε, the input head has moved off the right end of the input.

Example:

Consider 2DFA as follows M=({q0,q1,q2},{0,1},δ,q0,{q0,q1,q2}).Show that the string w=101001 is accepted or not. The transition table is given by

States	Inputs	
	0	**1**
q_0	$\{q_0,R\}$	$\{q_1,R\}$
q_1	$\{q_1,R\}$	$\{q_2,L\}$
q_2	$\{q_0,R\}$	$\{q_2,L\}$

Solution:

$q_0$101001 ⊢ 1 **q_1** 01001

⊢10 **q_1** 1001

⊢1 **q_2** 01001

⊢1 0 **q_0** 1001

⊢101 **q_1** 001

⊢1010 **q_1** 01

⊢1010 0 **q_1** 1

⊢1010 **q_2** 01

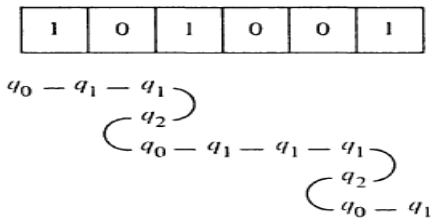
⊢1010 0 **q_0** 1

⊢1010 01 **q_1**

Here **q_1** is final state. So the given input string 01001 is accepted.

Crossing Sequences:

In 2DFA consists of the input, the path followed by the head and the state each time the boundary between two tape squares is crossed, with the assumption that the controls enters its new state prior to moving the head.



The list of states below each boundary between square is termed a ***crossing sequence.*** If a 2DFA accepts its input, no crossing sequence may have a repeated state with the head moving in the same direction, otherwise the 2DFA being deterministic, would be in a loop and thus could never fall off the right end.

In crossing sequences is that the first time a boundary is crossed, the head must be moving right. Subsequent crossings must be in alternate directions. Thus odd-numbered elements of a crossing sequence represent right moves and even-numbered elements represent left moves. If the input is accepted, it follows that all crossing sequences are of odd length.

A crossing sequence q_1, q_2... q_k is said to be valid if it's of odd length, and no two odd and no two even-numbered elements are identical. A 2DFA with s states can have valid crossing sequence of length at most 2s, so the number of valid crossing sequence is finite.

10. Explain Moore and Mealy machines with an example? (11 marks) (NOV 2012,MAY'15)

The finite automaton defined that its output is limited to a binary signal: "accept/don't accept". There are two different approaches are

1. Moore Machine
2. Mealy Machine

MOORE MACHINE:

- Moore Machines was developed by E.F.Moore in the year 1956.
- In Moore machine the output depends upon the states.
- It can be represented by transition tables and transition diagram.

A Moore machine is a 6 tuple **(Q, Σ, Δ, δ , λ ,q_0),**

Where

1. **Q** is a finite set of states
2. **Σ** is a finite input alphabet
3. **Δ** is the output alphabet
4. **λ** is the mapping from **Q** to **Δ** giving the output associated with each state
5. **δ** is the transition function
6. **q_0** in Q is the start state or initial state

- ✓ The output of M in response to input a_1 a_2... a_n, n≥0, is $\lambda(q_0)$ $\lambda(q_1)$.. $\lambda(q_n)$ where q_0,q_1, ... q_n is the sequence of states such that $\delta(q_{i-1},a_i)=q_i$ for $1 \le i \le n$.
- ✓ Any Moore machine gives output $\lambda(q_0)$ in response to input n.

- ✓ The DFA as a special case of a Moore machine where the output alphabet is {0, 1} and state "accepting " if only if λ(q)=1.

- In a Mealy machine, the outputs are a function of the present state and the value of the inputs as shown in Figure 1.
- Accordingly, the outputs may change asynchronously in response to any change in the inputs.

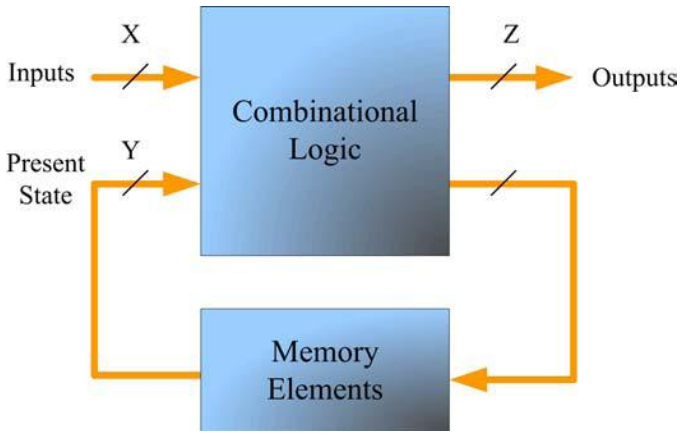


Figure 1: Mealy Type Machine

- In a Moore machine the outputs depend only on the present state as shown in Figure 2.
- A combinational logic block maps the inputs and the current state into the necessary flip-flop inputs to store the appropriate next state just like Mealy machine.
- However, the outputs are computed by a combinational logic block whose inputs are only the flip-flops state outputs.
- The outputs change synchronously with the state transition triggered by the active clock edge.

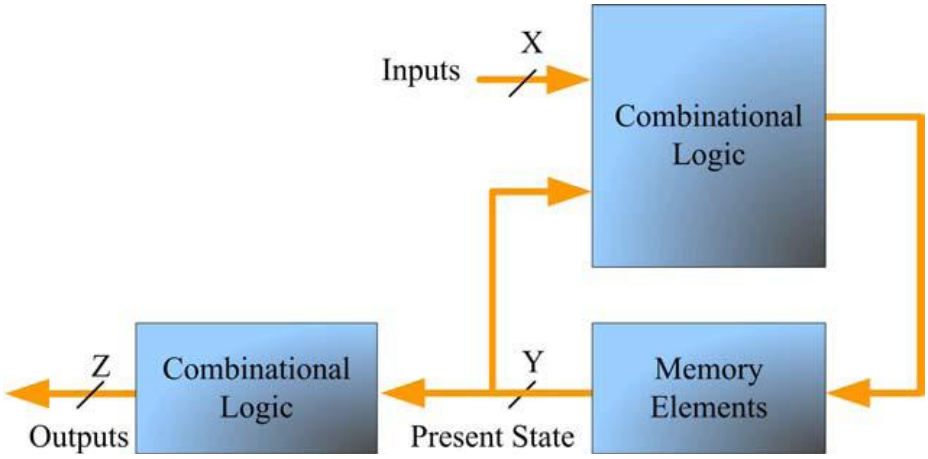


Figure 2: Moore Type Machine

MEALY MACHINE:

- Mealy Machine was developed by G.H.Mealy.
- In Mealy machine the output depends on transition and current state.
- The output is fixed for particular input symbols.

- It can be represented by transition tables and transition diagram.

A Mealy machine is a 6 tuple **$M= (Q, \sum, \Delta, \delta, \lambda, q_0)$,**

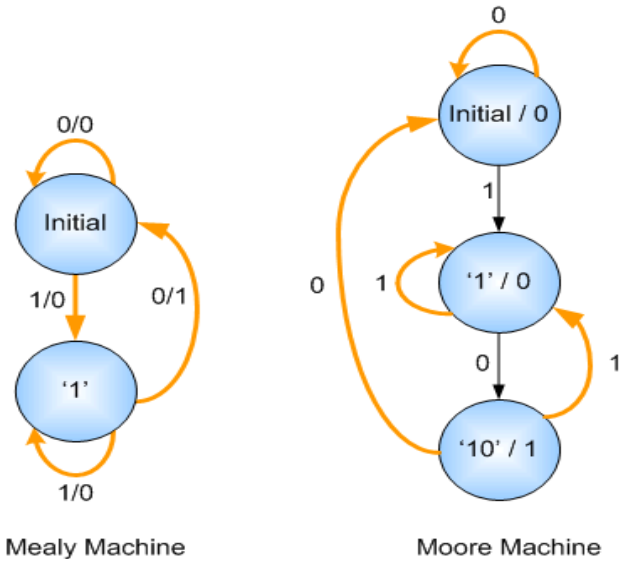
where

1. **Q** is a finite set of states
2. **Σ** is a finite input alphabet
3. **Δ** is the output alphabet
4. **λ** is the mapping from **Q X ∑ to Δ** (i.e.) λ(q,a) gives the output associated with the transition from state *q* on input *a.*
5. **δ** is the transition function
6. **q_0** in Q is the start state or initial state

- The output of M in response to input $a_1\ a_2 \ldots\ a_n$ is $\lambda(q_0,a_1)\ \lambda(q_1,a_2) \ldots \lambda(q_{n-1},a_i)$ where $q_0,q_1, \ldots ,q_n$ is the sequence of states such that $\delta(q_{i-1},a_i)=q_i$ for $1 \leq i \leq n$.
- In this sequence has length n rather than length n+1 as for the Moore machine, and on input ε a Mealy machine gives output ε.

<u>Comparison of the Moore and Mealy Machine:</u>

- Consider a finite state machine that checks for a pattern of '10' and asserts logic high when it is detected.
- The state diagram representations for the Mealy and Moore machines are shown in figure below.
- The state diagram of the Moore machine lists the inputs with their associated outputs on state transitions arcs.
- The value stated on the arrows for Mealy machine is of the form Z_i/X_i where Z_i represents input value and X_i represents output value.
- A Mealy machine produces a unique output for every state irrespective of inputs.
- Accordingly the state diagram of the Moore machine associates the output with the state in the form state-notation/output-value.
- The state transition arrows of Moore machine are labeled with the input value that triggers such transition.
- Since a Mealy machine associates outputs with transitions, an output sequence can be generated in fewer states using Mealy machine as compared to Moore machine. This was illustrated in the previous example.



11. Give applications of finite automata? (6 marks)

There are a variety of software design problems that are simplified by automatic conversion of regular expression notation to an efficient computer implementation of the corresponding finite automaton.

The two applications are

1. Lexical analyzers
2. Text editors

LEXICAL ANALYZERS:

- Tokens of a programming language are regular sets.
- For example: **ALGOL identifiers**, which are upper-or lower-case letters followed by any sequence of letters and digits with no limit on length, expressed as,

  (letter)(letter + digit)*

  where
  - ✓ "letter" stands for A + B + …… + Z + a + b + …… + z and
  - ✓ "digit" stands for 0 + 1 + ….. + 9
- **FORTRAN identifiers**, with length should limited of six and letters restricted to uppercase and symbol $, may be expressed as

  (letter)(ε + letter + digit)*

  where, "letter" stands for ($ + A + B + ……. + Z).
- A number of ***lexical analyzers*** takes input as sequence of regular expressions describing the tokens and produce a single finite automaton recognizing the any token.
- Usually, they convert the regular expression to an NFA with ε-transitions and then construct subset of states to produce a DFA directly rather than first eliminating ε-transitions.
- Each final state indicates the particular token found, so the automaton is really a Moore Machine.

- The transition function of the FA is encoded in one of several ways to take less space than the transition table represented as a two-dimensional array.
- The lexical analyzer produced by the generator is a fixed program that interprets coded tables, together with the particular table that represents the FA recognizing the tokens.
- This lexical analyzer can be used as a module in a compiler.

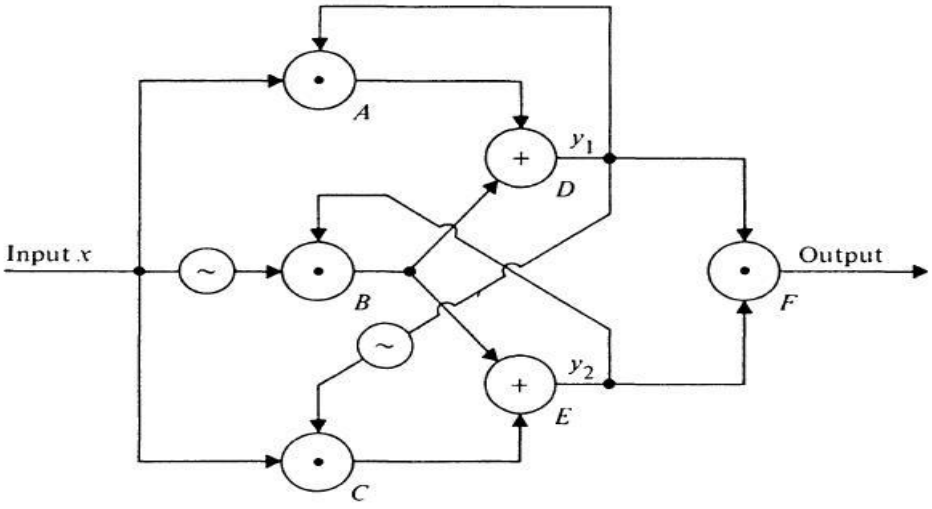
TEXT EDITORS:

- Certain text editor and similar programs permits substitution of string for any string matching given regular expression.
- For example: the UNIX text editor allows a command such as

  s/bbb*/b/

  that substitutes a single blank for the first string of two or more blanks found in a given line.
- Let "any" represents expression $a_1+a_2+.....+a_n$, where a_i's are computer characters except the "newline "character.
- We could convert a regular expression r to a DFA that accepts any *r.
- The presence of any* allows to recognize a member of L(r) beginning anywhere in the line.
- The conversion of a regular expression to a DFA takes more time than it takes to scan a single sort line using the DFA and the DFA have a number of states that is an exponential function of the length of the regular expression.
- In UNIX text editor is that Regular expression of any*r converted to an NFA with ε-transitions and NFA is then simulated directly.

However once a column has been constructed listing all the states the NFA can enter on a particular prefix of the input, the previous column is no longer needed and is thrown away to save space.



Deterministic finite automata have many practical applications:

1. Almost all compilers and other language-processing systems use DFA-like code to divide an input program into tokens like identifiers, constants, and keywords and to remove comments and white space.
2. For many applications that accept typed commands, the command language is quite complex, almost like a little programming language. Such applications use DFA-like code to process the input command.
3. Text processors often use DFA-like code to search a text fi le for strings that match a given pattern. This includes most versions of UNIX tools like awk, egrep, and Procmail, along with a number of platform-independent systems such as MySQL.
4. Speech-processing and other signal-processing systems often use a DFA-like technique to transform an incoming signal.
5. Controllers use DFA-like techniques to track the current state of a wide variety of finite-state systems, from industrial processes to video games. They can be implemented in hardware or in software.
6. Sometimes, the DFA-like code in such applications is generated automatically by special tools such as lex.

12.With a suitable transition diagram differentiate DFA and NFA (MAY'15 APR'16)

S.NO	DFA	NFA
1	"DFA" stands for "Deterministic Finite Automata"	"NFA" stands for "Nondeterministic Finite Automata."
2	Both are transition functions of automata. In DFA the next possible state is distinctly set	In NFA each pair of state and input symbol can have many possible next states.
3	DFA cannot use empty string transition.	NFA can use empty string transition
4	it is more difficult to construct DFA.	NFA is easier to construct
5	Backtracking is allowed in DFA	In NFA it may or may not be allowed.
6	DFA requires more space	NFA requires less space.
7	DFA can be understood as one machine and a DFA machine can be constructed for every input and output,	NFA can be understood as several little machines that compute together, and there is no possibility of constructing an NFA machine for every input and output.

Steps: The method of converting NFA to its equivalent DFA. Let M = (Q, Σ, δ, q0, F) is a NFA which accepts the language L(M). There should be equivalent DFA denoted by M'= (Q', Σ', δ', q0', F') such that L (M) =L (M').

The conversion method will follow following steps:

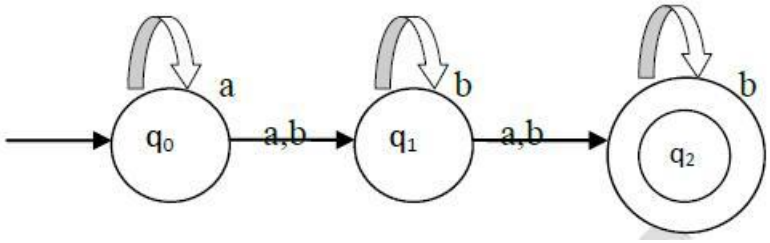
1. The start state of NFA M will be the start for DFA M'. Hence add q0 of NFA(start state) to Q'. Then find the transitions from this start state.
2. For each state [q1,q2..qi] in Q the transition for each input symbol Σ can be obtained as,

- ✓ δ'([q1,q2...qi],a) = δ(q1,a)U δ(q1,a)U .....δ(qi,a) =[q1,q2....qk] may be some state.
- ✓ Add the state [q1,q2....qk] to DFA if it is not already added in Q'.
- ✓ Then find out the transition for every input symbol from Σ for state[q1,q2....qk]. if we get some state [q1,q2....qn] which is not in Q' of DFA then add this state to Q'.
- ✓ If there is no new state generating then stop the process after finding all the transitions.

3. For the state [q1,q2.....qn] ϵ Q' of DFA if any one state qi is a final state of NFA then [q1,q2,,,,,qn] becomes a final state. Thus the set of all the final states ϵ F' of DFA.

E**xample**

Consider a NFA with the transition diagram



The NFA is given by

Q= {q0, q1, q2}

Σ= {a, b}

q0= {q0}

F= {q2}

Transition Table

δ	a	b
q0	{q0,q1}	{q1}
q1	{q2}	{q1,q2}
q2	φ	{q2}

Converting to DFA

The DFA can be expressed during conversion M´ = (Q, Σ, δ ´, q0, F´)

where

Q´=2_Q

Σ= {a, b}

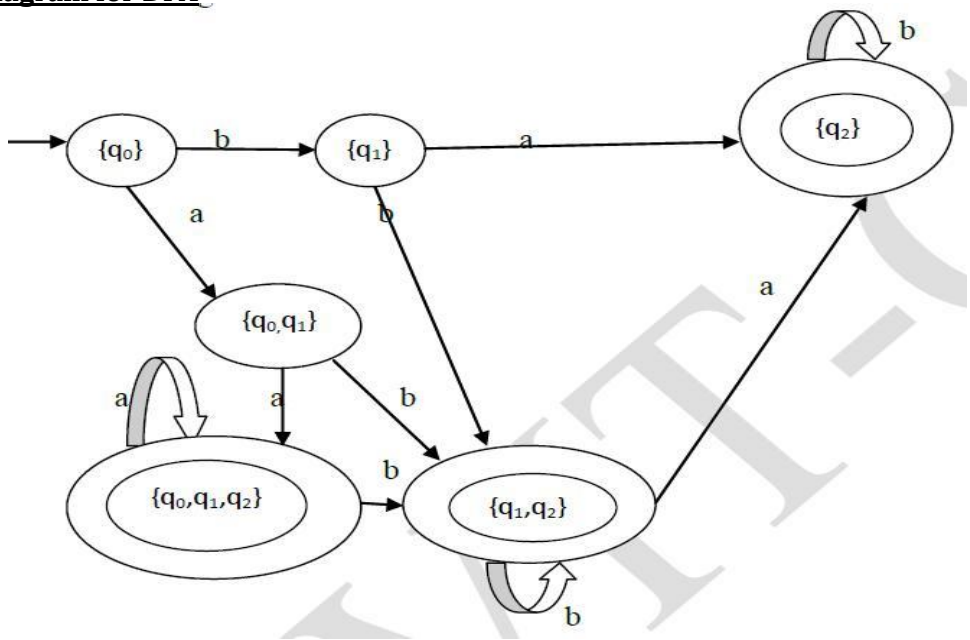
q0=q0

F´= {{q2}, {q1, q2}, {q0, q2}, {q0, q1, q2}}

Transition table for DFA

δ	a	b
{q0}	{q0,q1}	{q1}
{q1}	{q2}	{q1,q2}
{q2}	φ	{q2}
{q0,q1}	{q0,q1,q2}	{q1,q2}
{q1,q2}	{q2}	{q1,q2}
{q0,q1,q2}	{q0,q1,q2}	{q1,q2}

Transition diagram for DFA



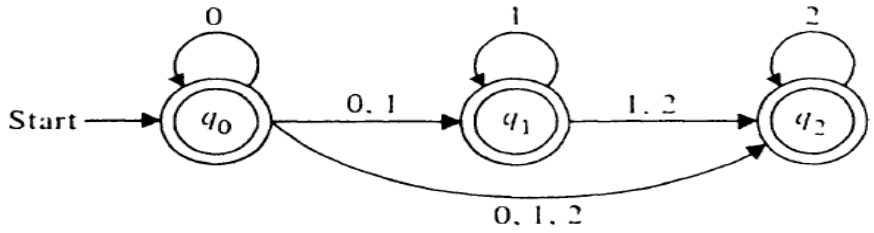
13. Construct a DFA equivalent to NFA, where M = ({q0, q1, q2}, {0, 1, 2}, δ, q0, {q0, q1, q2}).

The Transition is given by the transition diagram and transition table.

Transition Table:

	States	Inputs		
		0	1	2
→*	q0	{q0,q1,q2}	{q1,q2}	{q2}
*	q1	{φ}	{q1,q2}	{q2}
*	q2	{φ}	{φ}	{q2}

Transition Diagram:



CONSTRUCTING DFA:

Solution:

The equivalent DFA is M' = (Q, {0, 1, 2}, δ', S', F')

Q = {q0, q1, q2}

∑ = {0, 1, 2}

Q' = 2^Q

S'= [q0] Є Q'

All subsets = { [φ],[q0],[q1],[q2],[q0,q1,q2],[q0q1],[q0q2],[q1q2][q0q1q2]}

Transition Function:

δ'(q0,0) = [q0q1q2]

δ'(q0,1) = [q1q2]

δ'(q0,2) = [q2]

δ'(q1,0) = [φ]

δ'(q1,1) = [q1q2]

δ'(q1,2) = [q2]

δ'(q2,0) = [φ]

δ'(q2,1) = [φ]

δ'(q2,2) = [q2]

δ'(q0q1,0) = δ'(q0,0) U δ'(q1,0)

= [q0, q1, q2] U {φ}

= [q0, q1, q2]

δ'(q0q1,1) = δ'(q0,1) U δ'(q1,1)

= [1q2] U [q1q2]

= [q1, q2]

δ'(q0q1,2) = δ'(q0,2) U δ'(q1,2)

= [q2] U [q2]

= [q2]

δ'(q0q2,0) = δ'(q0,0) U δ'(q2,0)

= [q0q1q2] U {φ}

= [q0q1q2]

δ'(q0q2,1) = δ'(q0,1) U δ'(q2,1)

= [q1q2] U {φ}

= [q1q2]

δ'(q0q2,2) = δ'(q0,2) U δ'(q2,2)

= [q2] U [q2]

= [q2]

δ'(q1q2,0) = δ'(q1,0) U δ'(q2,0)

= {φ} U {φ}

= [φ]

δ'(q1q2,1) = δ'(q1,1) U δ'(q2,1)

= [q1q2] U {φ}

= [q1q2]

δ'(q1q2,2) = δ'(q1,2) U δ'(q2,2)

= [q2] U [q2]

= [q2]

δ'(q0q1q2,0) = δ'(q0,0) U δ'(q1,0) U δ' (q2,0)

= {q0q1q2} U {φ} U {φ}

= [q0q1q2]

δ (q0q1q2,1) = δ'(q0,1) U δ'(q1,1) U δ' (q2,1)

= [q1q2] U [q1q2] U {φ}

= [q1q2]

δ'(q0q1q2,2) = δ'(q0,2) U δ'(q1,2) U δ' (q2,2)

= [q2] U [q2] U [q2]

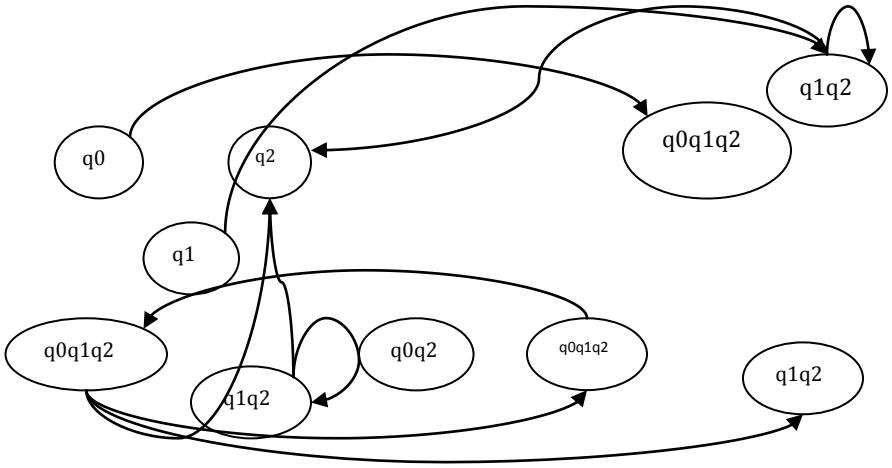
= [q2]

Transition Table for DFA:

States	Inputs		
	0	**1**	**2**
[q0]	[q0q1q2]	[q1q2]	[q2]
[q1]	φ	[q1q2]	[q2]
[q2]	φ	φ	[q2]
[q0q1]	[q0q1q2]	[q1q2]	[q2]
[q0q2]	[q0q1q2]	[q1q2]	[q2]
[q1q2]	φ	[q1q2]	[q2]
[q0q1q2]	[q0q1q2]	[q1q2]	[q2]

Final States are {[q2], [q1q2], and [q0q1q2]}

Transition Diagram:



14. Prove that for every regular expression there is an equivalent NFA? (11 MARKS)

Statement:

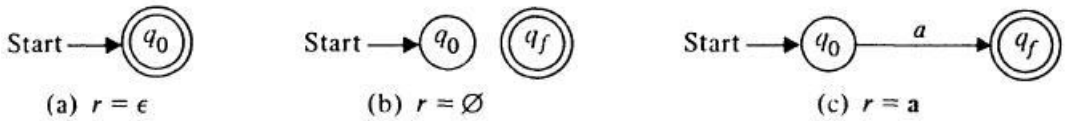
Let r be a regular expression. Then there exists an NFA with ε-transitions that accepts L(r).

Proof:

We prove by induction on the number of operators in the regular expression *r* that there is an NFA M that there is an NFA M with ε-transition, having one final state and no transitions out of this final state, such that *L(M)= L(r).*

Basis Step (Zero operators):

The expression 'r' must be ε (epsilon), Φ or 'a' for some a ϵ Σ. Then the equivalent NFA's are (a), (b) and (c) clearly satisfy the conditions.



Finite automata for basis step

Induction (One or more operators):

Assume that the theorem is true for regular expressions with fewer than i operators, i≥1. Let *r* have *i* operators. There are three cases depending on the form of *'r'.*

Case (1):

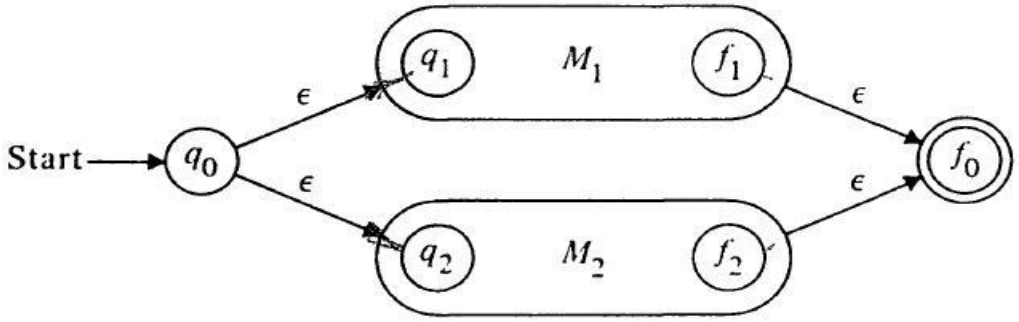
Let r= r1+ r2. Both r1 and r2 must have fewer than *i* operators.

Thus there are NFA's,

M1 = (Q1, Σ1, δ1, q1, {f1}) and M2 = (Q2, Σ2, δ2, q2, {f2}) with L(M1)= L(r1) and L(M2)= L(r2).

Assume Q1 and Q2 are disjoint. Let q0 be a new initial and f0 a new final state.
Construct M = (Q1 ∪ Q2 ∪ {q0, f0}, Σ1 ∪ Σ2, δ, {q0}, {f0}), where δ is defined by

1. δ(q0, ε) = {q1, q2}
2. δ(q, a) = δ1(q, a) if q ϵ Q1 →{f1}, a ϵ Σ1 ∪ { ε }
3. δ(q1, a) = δ2(q, a) if q ϵ Q2 → {f1}, a ϵ Σ2 U { ε }
4. δ1(f1, ε)= {f0}.

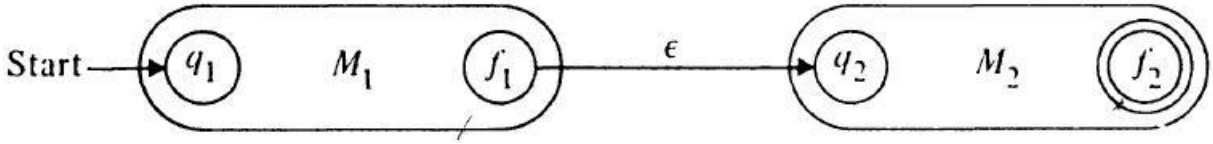


All the moves of M1 and M2 are present in M any path in the transition diagram of M from q0 to f0 must begin by going to either Q1 or Q2 on ε. If the path goes to Q1, it may follow any path in M1 to f1 and go to f0 on ε. Similarly paths begins by going to Q2 may follow any path in M2 to f2 and then go to f0 on ε. These are the only paths from g0 to f0. If there is a path labeled 'x' in M1, from Q1 to f1 or a path in M2 from q2 to f2. Hence **L(M)= L(M1) ∪ L(M2).**

Case (2):

Let r= r1r2.

Let M1 = (Q1, Σ1, δ1, q1, {f1}) and M2 = (Q2, Σ2, δ2, q2, {f2}) with L(M1) = L(r1) and L(M2) = L(r2).



Construct M = (Q1 ∪Q2, Σ1 ∪ Σ2, δ, {q1}, {f2}),

where δ is defined as,

1. δ(q, a) = δ1(q, a) for q in Q1→ {f1}, a in Σ ∪{ ε}

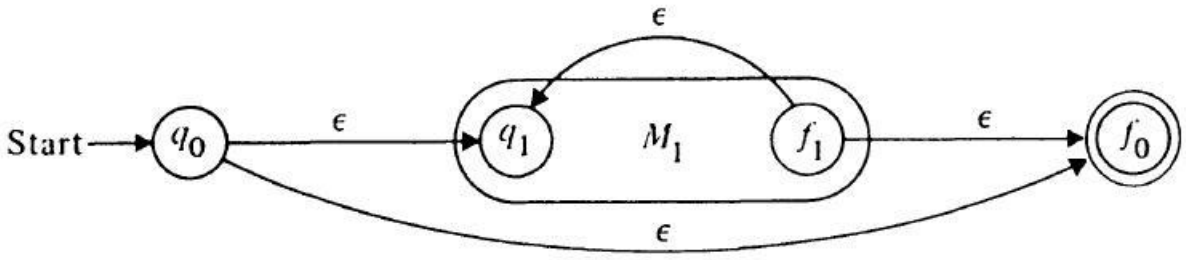
2. δ(f1, ε) = {q2}
3. δ(q, a) = δ(q, a) for q in Q2 and a in Σ2 ∪ { ε}

Every path in M< from q1 to f2 is a path labeled by some string 'x' from q1 to f1, followed by some string 'y' from q2 to f2. Thus L(M) = {xy | x is in L(M2) and y is in L(M2)}. And L(M)= L(M1)L(M2).

Case (3):

Let r= r*. Let M1= (Q1, Σ1, δ1, q1, {f1}) and L(M1) = r1.

Construct M= (Q1 ∪{f0}, Σ1, δ1, q0, {f0})



Where δ is given by

1. δ(q0, ε) = δ(f1, ε) = {q1, f0}
2. δ(q, a) = δ1(q, a) for q in Q1 → {f1} and a in Σ1 ∪{ ε}.

Any path from q0 to f0 consists either of a path from q0 to f0 or a path from q0 to q1 on ε followed by same no. of paths from q1 to f1, then back to q1 on ε. There is a path in M from q0 to f0 labeled 'x' if we write

x= x1, x2, ……xj for some j≥ 0 such that each xi belongs to L(M1). Hence **L(M)= L(M1*).**

15. Prove: A language L is accepted by some ε-NFA if and only if L is accepted by some DFA. (APR ,15)

To Prove:

The Language accepted by NFA is equal to the language set
accepted by DFA. i.e. L(M)=L(M´) ,
Where M is the NFA and M´ is the DFA.

Proof:

Let M= (Q, Σ, δ, q0, F) be an NFA, accepting L and we can
define a DFA M´ = (Q, Σ, δ´, q0, F´) as follows.
The states of M´ are all the subsets of the set
of states of M. That is, Q´ = 2Q
M´ will keep track in its states of all the states M could be in at any given time. F´ is the set all the states in Q´ containing the final state of M. An element of Q´ will be denoted by [q1, q2, q3....qi] are in Q.
Observe that, [q1, q2, q3....qi] is a single state of DFA corresponding to the
states of the NFA. Note q0´= [q0]

We define

$$\delta'([q1, q2, q3 .... qi], a) = [P1, P2, P3 .... Pj]$$

if and only if

$$\delta([q1, q2, q3 ....qi], a) = [P1, P2, P3 .... Pj]$$

that is

δ' applied to an element [q1, q2, q3 ....qi] of Q′ is computed by applying δ to each state of Q represented by [q1, q2, q3 .... qi] in applying δ to each of [q1, q2, q3 .... qi] and taking "union" we get some new set of states P1, P2, P3 .... Pj. This new set of states has a representative P1, P2, P3.... Pj in Q′ and that element is the value of δ'([q1, q2, q3 .... qi], a).

It is easy to show by induction for the length of the input string x, that

$$\delta'(q0', x) = [q1, q2, q3 .... qi]$$

if and only if,

$$\delta(q0, x) = \{q1, q2, q3 .... .qi\}$$

Basis

Consider |x|=0 then x must be since q0′= [q0]

Then

$$\delta'(q0, \varepsilon) = (q0)$$

if and only if

$$\delta(q0, \varepsilon) = \{q0\}$$

Hence the result is trivial.

Induction

Let us assume that the hypothesis is true for inputs of length m, Let xa be a string of length m+1 with a in Σ, then

$$\delta'(q0', xa) = \delta'(\delta'(q0', x), a)$$

By the inductive hypothesis,

$$\delta'(q0', x) = [P1, P2, P3. ... Pj]$$

If and only if ,

$$\delta(q0, x) = \{P1, P2, P3 .... Pj\}.$$

But by the definition of δ',

$$\delta'[(P1, P2, P3 .... Pi), a] = [r1, r2, r3 .... rk]$$

If and only if,

$$\delta(\{P1, P2, P3 .... Pi\}, a) = [r1 ,r2, r3 .... rk]$$

Thus,

$$\delta'(q0, xa) = [r1, r2, r3 .... rk]$$

If and only if,

$\delta(q0, xa) = [r1, r2, r3 .... rk]$

Which establishes the inductive hypothesis,

To complete the proof, we have only to add that $\delta'(q0', x)$ is in F' exactly when $\delta(q0)$ contains a state of Q that is F.

Thus, $L(M)=L(M')$

PONDICHERRY UNIVERSITY QUESTIONS

2 MARKS

1. Give the purpose of transition diagram? **(APR 2014)(Ref.Qn.No.20, Pg.no.8)**
2. Write the DFA Specifications? **(APR 2014, NOV 2012) (Ref.Qn.No.22, Pg.no.9)**
3. Define NFA. **(APR 2012, 2013)(Ref.Qn.No.25, Pg.no.10)**
4. Define two way finite automata (2DFA)? **(APR 2012, NOV 2013) (Ref.Qn.No.32, Pg.no.12)**
5. Give the applications of finite automata? **(NOV 2012,NOV'14) (Ref.Qn.No.41, Pg.no.14)**
6. Define Equivalence Relation? **(NOV 2013) (Ref.Qn.No.46, Pg.no.16)**
7. What are the difference between NFA and DFA? **(MAY '15,NOV'15) (Ref.Qn.No.26, Pg.no.10)**
8. Define regular expressions? **(NOV'14) (Ref.Qn.No.29, Pg.no.11)**
9. Define Finite Automata. **(NOV'15) (Ref.Qn.No.18, Pg.no.8)**

11 MARKS

1. Explain Non-Deterministic Finite Automata (NFA) with an example? **(APR 2014) (Ref.Qn.No.4, Pg.no.24)**
2. Explain about converting DFA's to regular expression by eliminating states. **(APR 2014)**
3. Let r be a regular expression. Then there exists an NFA with ε-transitions that accepts L(r). **(NOV 2013) (Ref.Qn.No.11, Pg.no.43)**
4. If L is accepted by a 2DFA, then L is regular set. **(NOV 2013)**
5. Explain the DFA with example? **(APR 2013) (Ref.Qn.No.3, Pg.no.21)**
6. Explain about Regular Expressions with an example? **(APR 2013, APR 2012,MAY'15) (Ref.Qn.No.2, Pg.no.19)**
7. Describe in detail about two-way automation with an example. **(APR 2012) (Ref.Qn.No.8, Pg.no.36)**
8. Define the Moore and Mealy Machine. State the configuration and illustrate the same with an example. **(NOV 2012,MAY'15) (Ref.Qn.No.9 Pg.no.38)**
9. Define Regular expression. Write a r.e. denote a language L which accepts all the strings which begin or end with either 00 or11. **(NOV 2012)**
10. With a suitable transition diagram differentiate DFA and NFA **(MAY'15)) (Ref.Qn.No.12 Pg.no.45)**

UNIT II

Regular Sets and Context Free Grammars: Properties of regular sets, Context-Free Grammars – Derivation trees, Chomsky Normal Forms and Greibach Normal Forms, Ambiguous and unambiguous grammars.

---

2 Marks

1. What is pumping Lemma for regular sets?

- A basic result called the ***pumping lemma***, which is a powerful tool for proving certain languages, is regular or not.
- It is also useful in the development of algorithms to answer certain questions concerning finite automata, such as whether the language accepted by a given FA is finite or infinite.
- If a language is regular, it is accepted by a DFA M= (Q, Σ, δ, q0, F) with particular number of states.
- Let L be a regular set. Then there is a constant n such that if Z is any word in L, and |z|≥n. we may write Z=uvw in such a way that |uv| ≤ n, |v| ≥ 1and for all L ≥0 then uv^iw is in L.

2. List the applications of the pumping lemma? (APR 2014)

The applications of pumping lemma for regular set are

- The pumping lemma is a powerful tool providing and proving certain language is regular or not.
- It is also useful in the development of algorithms to answer certain question concerning finite automata, such as whether the language accepted by a given FA is finite or infinite.

(Or)

1) The pumping lemma is useful in proving that certain sets are not regular. The application is an "adversary arguments" of the following form.

2) Select the language L to prove non-regular.

3) The "adversary" picks n, the constants mentioned in the pumping lemma.

4) Select a string *z* in L.

5) The adversary breaks *z* into *u, v* and *w,* subject to the constraints that |uv|≤n and |v|≥ 1.

6) A contradiction to the pumping lemma by showing, for any *u, v,* and *w,* determined by the adversary, that there exists an *i* for which $\boldsymbol{uv^iw}$ is not in L. It may then be concluded that L is not regular.

3. What are the properties of Regular Sets?

If a class of languages is closed under a particular operation we call that fact a closure property of the class of languages. The properties of regular sets are

i. Union

ii. Concatenation

iii. Kleene closure

iv. Complementation

v. Intersection

vi. Substitution

vii. Homomorphism

4. What is homomorphism with example? (APR 2014)

A string homomorphism is a function on strings that works by substituting a particular string for each symbol.

Example:

The function h given by h(0) = abb and h(1) = ba is a homomorphism which replaces each 0 by abb and each 1 by ba. Thus h(1011) = baabbbaba.

5. What is inverse homomorphism?

Homomorphism can also applied in reverse and in this mode they also preserve regular language. That is, suppose h is a homomorphism from some alphabet Σ to strings in another (possibly the same) alphabet T^{-1}. Let L be a language over alphabet T. Then h^{-1} is the set of strings w in Σ* such that h(w) is in L.

6. Write the principal closure properties for regular language?

The principal closure properties for regular language are

i. The union of two regular languages is regular.

ii. The intersection of two regular languages is regular.

iii. The complement of a regular language is regular.

iv. The difference of two regular languages is regular.

7. Write the closure properties of regular set under Boolean operations?

The first closure properties are the three Boolean operations: Union, Intersection, and Complementation.

i. Let L and M be language over alphabet Σ. Then L∪M is the language that contains all strings that are in either or both of L and M.

ii. Let L and M be language over alphabet Σ. Then L∩M is the language that contains all strings that are in both of L and M.

iii. Let L be a language over alphabet Σ. Then, the complement of ¬L, is the set of strings in Σ* that are not in L.

8. Define Context Free Grammar (CFG)? (NOV 2013)

- A context free grammar (CFG) is a finite set of variables (also called non-terminals or syntactic categories) each of which represents a language.
- CFG is a way of describing languages by recursive rules or substitution rules called productions.
- A CFG consists of sets of variables, a set of terminal symbols and a start variable, and productions. Each production consists of a starting variable and a body consists of a string of zero or more variables or terminals.

9. Write the formal specification of Context Free Grammar (CFG)?

A Context Free Grammar (CFG) is denoted **G= (V, T, P, S),**

Where

1) V→ is a finite set of **variables or non-terminals.**

2) **T→**is a finite set of **terminal symbols.**

3) **P→** is a finite set of **productions**; each production is of the form **A→α,** where A is a non-terminal or variable and α is a string of symbols from **(V U T)*.**

4) **S→** is a special variable called the **start symbol.**

10. What are the notations used in Context Free Grammar (CFG)?

1) The capital letters ***A, B, C, D, E,*** and ***S*** denote ***variables or non-terminals.***

2) The lower case letters ***a, b, c, d, e, digits*** and ***boldface*** strings are ***terminals.***

3) The capital letters ***X, Y,*** and ***Z*** denote symbols that may be either ***terminals or non-terminals.***

4) The lower case letters ***u, v, w, x, y,*** and ***z*** denote strings are ***terminals.***

5) The lower case Greek letters ***α, β***, and ***γ*** denote strings having ***variables and terminals.***

11. Give the applications of context free grammar?

Context free languages are used in:

- Defining programming languages
- Formalizing the notation of parsing
- Translation of programming languages
- String processing applications

12. What are the uses of context free grammars?

- Construction of compliers.
- Simplified the definition of programming languages.
- Describes the arithmetic expressions with arbitrary nesting of balanced parenthesis {(,)}.
- Describes block structure in programming languages.
- Model neural nets.

13. What is meant by derivation? (NOV'15)

- Derivation is a process of producing any string derivable from start symbol of the grammar using the production of the grammar.
- Beginning with the start symbol, we derive terminal strings by repeatedly replacing a variable by the body of some production with that variable in the head. The language of the CFG is the set of terminal strings we can so drive; it is called a context-free language.

14. Define derivation trees? (NOV 2014)

- Derivation or parse trees, is structure on the words of a language that is useful in applications such as the compilation of programming languages.
- The vertices of a derivation tree are labeled with terminal or variable symbols of the grammar.

15. What is meant by parse tree?

- The parse tree is a tree that shows the essentials of a derivation. Interior nodes are labeled by variables, and leaves are labeled by terminals or ε.
- For each internal node, there must be a production such that the head of the production is the label of the node, and the labels of its children, read from right to left, from the body of that production.

16. What is yield of a parse tree?

If we look at the leaves of any parse tree and concatenate them from the left, we get a string, called the yield of the tree, which is always a string that is derived from the root variable. The fact that the yield is derived from the root will be proved shortly. Of special importance are those parse tree such that:

- **a.** The yield is a terminal string. That is, all leaves are labeled either with a terminal or with ε.
- **b.** The root is labeled by the start symbol.

17. Give the Formal definition of derivation tree?

Let G= (V, T, P, S) be a CFG. A tree is a derivation or parse tree for G if:

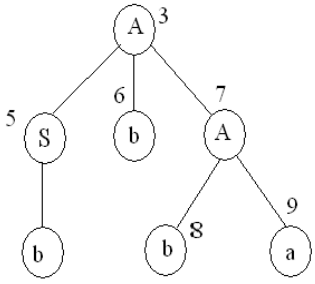
- **1)** Every vertex has a label, which is a variable or terminal or ε. i.e. *V U T U (ε).*
- **2)** The label of the root is *S* (start symbol).
- **3)** If a vertex is interior and has label *A*, then *A* must be in *V*.
- **4)** If n has label *A* and vertices n1, n2, n3, ……nk are the sons of vertex n, in order from the left, with labels x1, x2, ……xk respectively, thenA→x1, x2, x3, ……, xk must be a production in P.
- **5)** If vertex n has label ε, then n is a leaf and is the only son of its father.

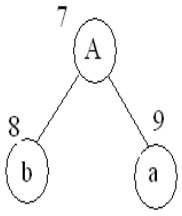
18. Define subtree?

- A subtree of a derivation tree is a particular vertex of the tree together with all its descendants, the edges connecting them and their labels.
- It looks like a derivation tree, except that the label of the root may not be the start symbol of the grammar. If a variable A labels the root, then we call the subtree an ***A-tree.***
- Thus ***"S-tree"*** is a synonym for ***"derivation tree"*** if S is the start symbol.

19. Give an example for a subtree?

- A Subtree ($A \overset{*}{\rightarrow}$ bbba) A→ ba which is derived from the above derivation tree
- **$S \overset{*}{\rightarrow}$ abbbab** $A \overset{*}{\rightarrow}$ **bbba**





A subtree looks like a derivation tree except that the label of the root may not be S. It is called A- Tree, if the label of the root is A.

20. What are the types of Derivation?

The two types of derivations are

- **i.** Left most derivation
- **ii.** Right most derivation

21. Define LMD with suitable examples? (APR 2013,MAY'15)

If at each step in a derivation a production is applied to the leftmost variable then the derivation is called "Left Most Derivation (LMD)".

Example:

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

P1 : E → E + E

P2 : E → E * E

P3 : E → id

The input string is **ω = id + id * id**

Left Most Derivation (LMD):

E →E + E

⇒ E + E * E

⇒ id + E * E

⇒ id + id * E

⇒ **id + id * id**

If the string is matched with the given production means, it is a correct grammar.

22. Define the RMD with example?

If at each step in a derivation a production is applied to the right most variable then the derivation is called "Right Most Derivation (RMD)".

Example:

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

P1 : E → E + E

P2 : E → E * E

P3 : E → id

The input string is **ω = id + id * id**

Right Most Derivation (RMD):

E → E + E

⇒ E + E * E

⇒ E + E * id

⇒ E + id * id

⇒ **id +id * id**

The above string is matched with the given string.

23. Consider a grammar given as G = ({S, A}, {a, b}, P, S), where P consists of

S→ aAS |a

A→ SbA | SS | ba

Find Derive the input string ω= aabbaa using LMD and RMD.

Solution

The Productions are

P1: S →aAS

P2: S → a

P3: A → SbA

P4: A → SS

P5: A→ba

LMD

S →aAS by P1

⇒ aSbAS by P3

⇒ aabAS by P2

⇒ aabbaS by P5

⇒ **aabbaa** by P2

The above string is matched with the given string.

RMD

S → aAS by P1

⇒ aAa by P2

⇒ aSbAa by P3

⇒ aabAa by P2

⇒ **aabbaa** by P5

The above string is matched with the given string.

24. Consider the production S→ aSb | ab. Find LMD and RMD for the input string aaabbb.

Solution

P1 : S→ aSb

P2 : S → ab

LMD

S→aSb

⇒ aaSbb by P1

⇒ aaabbb by P2

RMD

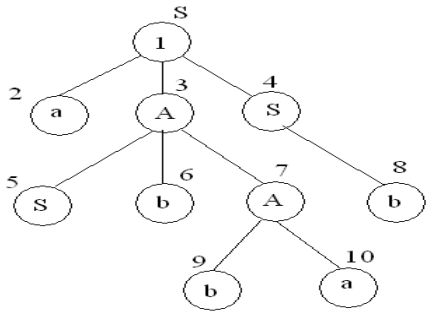
S→ asb

$\Rightarrow$ aasbb by P1

$\Rightarrow$ aaabbb by P2

The given input string is accepted.

Consider the grammar G = ({S, A},{a,b},P,S), where P consists of S→ aAS | b , A→ SbA | ba. Draw its equivalent derivation tree.



1) The vertices are numbered for reference 1, 2. . . 10
2) The label of the vertices are variables | terminals
3) The label of the root vertex is S-start symbol
4) The interior vertices are 1,3,4,5,7 which are variables

Thus the left-to-right ordering of derivation is: $S \overset{*}{\rightarrow} aAS \overset{*}{\rightarrow} aSBAb \overset{*}{\rightarrow} abbbab$

26. What is left sentential form?

If the string is produced or derived from the start symbol from left most derivation, then the string is called left sentential form.

27. What is right sentential form?

If the string is produced or derived from the start symbol from right most derivation, then the string is called right sentential form.

28. What is ambiguous grammar? (APR, NOV 2013)

- A context free grammar G such that string or word has two parse trees is said to be ambiguous.
- For some CFG's, it is possible to find a terminal string with more than one parse tree, or equivalently, more than one leftmost derivation or more than one rightmost derivation. Such a grammar is called ambiguous grammar.

29. What is unambiguous grammar?

For some CFG's, it is possible to find a terminal string with at most one parse tree, or equivalently, at most one leftmost derivation or at most one rightmost derivation. Such a grammar is called unambiguous grammar.

30. What is meant by inherent ambiguity?

A context-free language L is said to be inherent ambiguous if all its grammars are ambiguous. If even one grammar for L is unambiguous, then L is an unambiguous language.

31. What is meant by eliminating ambiguity?

For many useful grammars, such as those that describe the structure of programs in a typical programming language, it is possible to find an unambiguous grammar that generates the same language. Unfortunately, the unambiguous grammar is frequently more complex than the simplest ambiguous grammar for the language. There are also some context-free languages, usually quite contrived, that are inherently ambiguous, meaning that every grammar for that language is ambiguous.

32. What are the 3 ways to Simplify Context Free Grammar?

1) By removing the useless symbols from set of productions.

2) Elimination of empty or null production **(A→ε).**

3) Elimination of unit productions **(A→B).**

33. Give the productions obtained from CFG?

Every production should be in the form of A→BC, A→ a,

Where A, B, C are variables and 'a' is a terminal.

34. Define useless symbol?

Every production of G be one of the form A→aα where α is the string of variable only. This is called as useless symbols.

35. Define useful symbol?

- A symbol x is useful, if there exists a derivation $S \overset{*}{\rightarrow} \alpha X \beta \overset{*}{\rightarrow} w$, where α,β are sentential form (VUT)*, w is any string in T*.
- The variable or terminals which do not appear in any derivation of a terminal string from a start symbol are called as useless symbol.

36. What is meant by nullable productions?

The elimination of productions of the form A → ε, then A is called nullable or ε-productions.

37. Define unit production?

A production of the form A→B where A and B are variables is called unit productions.

38. What are the types of normal forms in CFG?

The two types of normal forms are

- Chomsky Normal Form (CNF)
- Greibach Normal Form (GNF)

39. Define Chomsky Normal Form (CNF)? (APR 2012,MAY'15)

The context free language is in Chomsky Normal Form (CNF) is generated by a grammar in which all productions are of the form:

- ✓ A→BC or
- ✓ A→a

where

A, B, and C are variables or non-terminals and *a* is a terminal.

40. Define Greibach Normal Form (GNF)? (APR 2012,NOV'15)

A context free grammar is said to be in Greibach Normal Form (GNF) if all productions are in the following form:

- ✓ $A \rightarrow \alpha X$
- ✓ $A \rightarrow a$

where

- A is a non-terminal symbols
- α is a terminal symbol
- X is a sequence of non-terminal symbols.

41. Consider the grammar G= ({S, C}, {a,b},P,S} where P consists of S→aCa,C→aCa/b. Find L(G).

Solution:

$S \rightarrow aCa$

$\Rightarrow aaCaa$

$\Rightarrow aaaCaaa$

$\Rightarrow \ldots\ldots\ldots$

$\Rightarrow a^iCa^i$

$\Rightarrow a^nba^n$ Because $C \rightarrow b$

The language $L(G)=\{ a^nba^n/n>0\}$

42. What is the language generated by the grammar G= (V, T, P, S) where P={S->aSb|a S->ab}?

S ==> aSb

==>aaSbb

==>aaabbb

i.e the general form is a^nb^n. Thus the language $L(G)=\{ a^nb^n \mid n\geq 1\}$. The language has strings with equal number of a's and b's.

43. Is the grammar G with following productions ambiguous? Justify. (NOV 2012)

S→SS | aSb | bSa | ε

Solution:

The given grammar is S→SS | aSb | bSa | ε

Let the input string is abab

LMD 1:

$S \rightarrow aSb$

$\Rightarrow abSab$

$\Rightarrow b\varepsilon ab$

$\Rightarrow$ **abab**

LMD2:

S→SS

⇒aSbS

⇒aSbε

⇒aSb

⇒aaSbb

⇒aaεbb

⇒**aabb**

Here, we get two leftmost derivations for the same input string. Hence it is ambiguous grammar.

44. **What is sentential form? (MAY 2015)**

String that are derived or produced from the start symbol is called sentential form.

Let G ={ V,T,P,S}be a CFG then a string of terminals and variables α is called a sentential form if $S \xrightarrow[G]{*} \alpha$

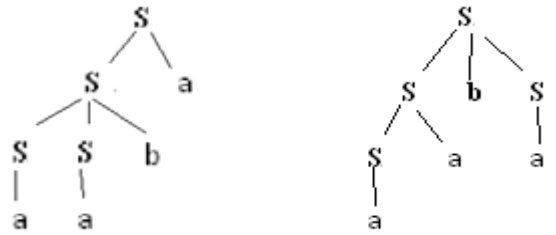
(ie) α in T* such that $S \xrightarrow[G]{*} \alpha$ is a sentential form.

45. Give an example for an ambiguous grammar (MAY'15)

Ambiguous grammar:

A grammar G is ambiguous if and only if there exist at least one string w εT* .For which two or more different parse tree exist by applying either by LMD or RMD

Let the string w = aaba , parse tree for the string shown below.



For the same string two different parse trees can be generated. Hence the given grammar is ambiguous.

11 Marks

1. Explain the closure properties of Regular Sets? (11 marks) (APR'14,NOV'15)

- A basic result called the ***pumping lemma***, which is a powerful tool for proving certain languages, is regular or not.
- It is also useful in the development of algorithms to answer certain questions concerning finite automata, such as whether the language accepted by a given FA is finite or infinite.

If a class of languages is closed under a particular operation we call that fact a closure property of the class of languages.

1. Union
2. Intersection
3. Complement
4. Difference
5. Homomorphism
6. Inverse homomorphism

REFER CLASS NOTES FOR THEOREM

2. Explain Context Free Grammar (CFG)? (11 marks) (APR 2012, 2013)

- A context free grammar (CFG) is a finite set of variables (also called non-terminals or syntactic categories) each of which represents a language.
- CFG is a way of describing languages by recursive rules or substitution rules called productions.
- A CFG consists of sets of variables, a set of terminal symbols and a start variable, and productions. Each production consists of a starting variable and a body consists of a string of zero or more variables or terminals.

A Context Free Grammar (CFG) is denoted G= (V, T, P, S),

Where

1. **V→** is a finite set of ***variables or non-terminals.***
2. **T→**is a finite set of ***terminal symbols.***
3. **P→** is a finite set of ***productions***; each production is of the form A→α, where A is a non-terminal or variable and α is a string of symbols from (V U T)*.
4. **S→** is a special variable called the ***start symbol.***

Notations:

- The capital letters A, B, C, D, E, and S denote variables or non-terminals.
- The lower case letters a, b, c, d, e, digits and boldface strings are terminals.
- The capital letters X, Y, and Z denote symbols that may be either terminals or non-terminals.
- The lower case letters *u, v, w, x, y,* and *z* denote strings are terminals.
- The lower case Greek letters α, β, and γ denote strings having variables and terminals.

Example:

Consider this grammar as ({E}, {+,*, (,), id}, P, E), where P consists of

E→E+E

E→E*E

E→(E)

E→id

Applications of context free grammar are

- ✓ Defining programming languages
- ✓ Formalizing the notation of parsing
- ✓ Translation of programming languages
- ✓ String processing applications

3. What are derivation trees with an example? (11 marks) (APR 2012, 2014)

- Derivation or parse trees, for CFG G=(V,T,P,S) is tree which satisfies following condition.
  - Every vertex has label which is variable or terminal symbol
  - The root has label S
  - The label of an internal vertex is a variable.
  - If S has a label A and vertices x1,x2,x3,…xn are son of vertex x in order from left to right with labels x1,x2,x3,….xk
  - A vertex x is a leaf if it is labeled as E then x is only son of its fathet

Example of derivation tree:

Consider this grammar as ({E}, {+,*, (,), id}, P, E), where P consists of

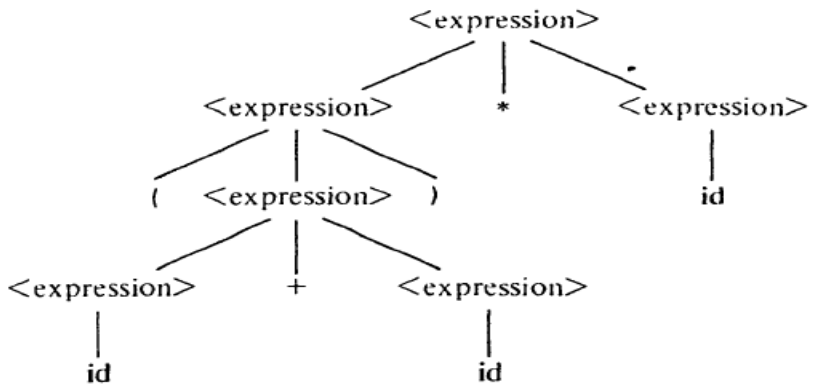
E→E+E

E→E*E

E→(E)

E→id

The input string is **(id+id) *id**



Derivation tree

Let G= (V, T, P, S) be a CFG. A tree is a derivation or parse tree for G if:

1. Every vertex has a label, which is a variable or terminal or ε. i.e. V U T U (ε).
2. The label of the root is *S* (start symbol).
3. If a vertex is interior and has label *A*, then *A* must be in *V*.
4. If n has label *A* and vertices n1, n2, n3, ……nk are the sons of vertex n, in order from the left, with labels x1, x2, ……xk respectively, then A→ x1, x2, x3, …… , xk must be a production in P.
5. If vertex n has label ε, then n is a leaf and is the only son of its father.

Consider the grammar G = ({S, A}, {a,b},P,S), where P consists of

S→ aAS | a

A→ SbA | SS|ba

Draw its equivalent derivation tree.

Solution:

S->aAS

S->a

A-.SbA

A->SS

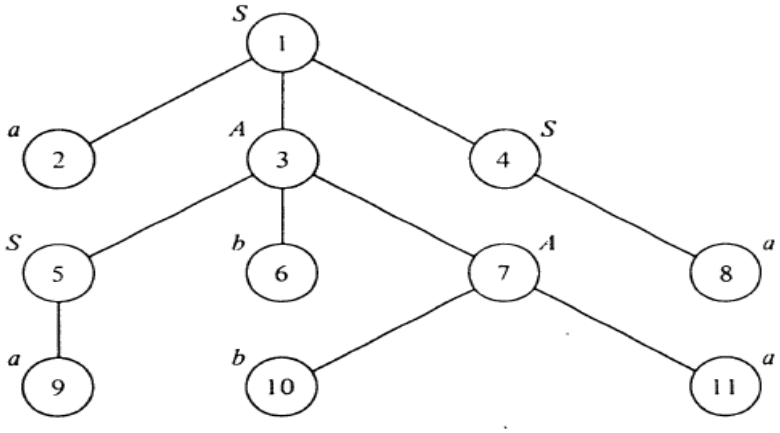
A->ba

aAS

aSbAS

aabAS

aabbaa



1) The interior vertices are numbered for reference 1,3,4,5, and 7.
2) The label of the vertices are variables | terminals
3) The label of the root vertex is S-start symbol
4) The interior vertices are 1,3,4,5,7 which are variables

Thus the left-to-right ordering of derivation is: **S** ⇒aAS ⇒aSbAS ⇒aabAS ⇒aabbaS ⇒**abbbab**

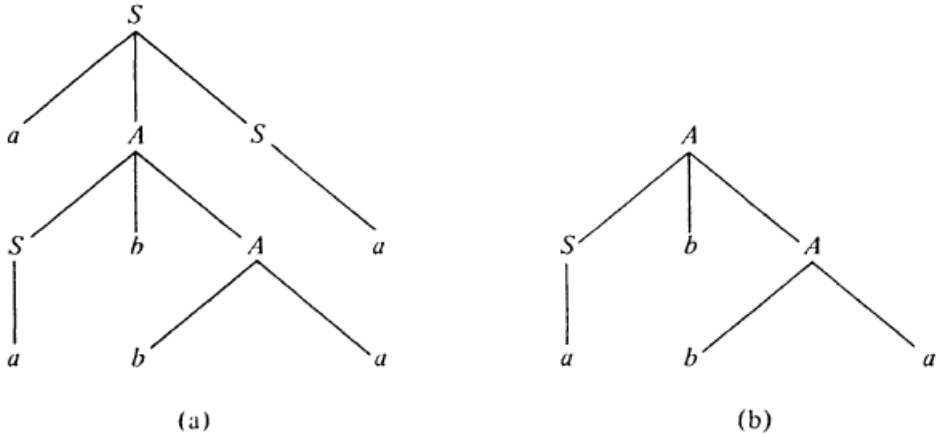
Subtree:

- A subtree of a derivation tree is a particular vertex of the tree together with all its descendants, the edges connecting them and their labels.
- It looks like a derivation tree, except that the label of the root may not be the start symbol of the grammar. If a variable A labels the root, then we call the subtree an ***A-tree.***
- Thus ***"S-tree"*** is a synonym for ***"derivation tree"*** if S is the start symbol.

Example for a Subtree:

- A Subtree ($A \Rightarrow Sba \Rightarrow abA \Rightarrow abba$) $A \rightarrow Sba$ which is derived from the above derivation tree

$S \overset{*}{\rightarrow} aabbaa$ $\qquad$ $A \overset{*}{\rightarrow} abba$



Derivation tree and subtree.

A subtree looks like a derivation tree except that the label of the root may not be S. It is called A-Tree, if the label of the root is A.

Types of Derivation:

The two types of derivations are

- If at each step in a derivation a production is applied to the leftmost variable then the derivation is called ***"Left Most Derivation (LMD)".***
- If at each step in a derivation a production is applied to the right most variable then the derivation is called ***"Right Most Derivation (RMD)".***

Example:

Consider a grammar given as G = ({S, A}, {a, b}, P, S), where P consists of

$S \rightarrow aAS \mid a$

$A \rightarrow SbA \mid SS \mid ba$

Derive the input string ω= aabbaa using LMD and RMD.

Solution

The Productions are

P1: $S \rightarrow aAS$

P2: $S \rightarrow a$

P3: A → SbA

P4: A → SS

P5: A → ba

Left Most Derivation (LMD):

S →aAS

⇒ aSbAS

⇒ aabAS

⇒ aabbaS

⇒ **aabbaa**

The above string is matched with the given string.

Right Most Derivation (RMD):

S → aAS by P1

⇒ aAa by P2

⇒ aSbAa by P3

⇒ aabAa by P2

⇒ **aabbaa** by P5

The above string is matched with the given string.

4. Give a grammar to generate the CFL L= {$a^n b^n$: n <= m+3}. (5 marks)

1. **S →aABb**

   A →aa

   B→ bb

   S ⇒aABb

   S⇒aaaBb

   S⇒aaabbb

   L= {$a^n b^n$: n <= m+3}

2. **S→aaSbb**

   S→ab

   (i) S⇒aaSbb

   S⇒aaabbb

5. Prove whether {0^n | n is a power of 2} is regular set or not? OR

Verify whether { o^{2^n} | n>=1} is regular set or not? (NOV'14) (5 marks)

Let the given language L={0^n, n is a power of 2}

L={00,0000,00000000, …}

CASE 1:

Consider z= 00

Here u=0, w=0, V=ε

V must not have ε value.

CASE 2:

Consider z= 0000

Here u=0, w=0 and v=00

$|uv| \leq n$

$3 \leq 4$

$|v| \geq 1$

$\Rightarrow \mathbf{uv^iw}$

$= 0(00)^i0$

When i=1

$=0(00)^10$

$=0000$

When i=2

$=0(00)^20$

$=000000$

Thus the string 000000 is not present in the set so it is not a regular set.

6. Prove whether $\{0^n1^n \mid n \geq 1\}$ is regular set or not? (5 marks)

Let the given language $L=\{0^n1^n, n \geq 1\}$

L={01,0011,000111, …}

CASE 1:

Consider z= 01

Here u=0, w=1, V=ε

V must not have ε value.

CASE 2:

Consider z= 0011

Here u=0, w=1 and v=01

$|uv| \leq n$

$3 \leq 4$

$|v| \geq 1$

$\Rightarrow \mathbf{uv^iw}$

$= 0(01)^i1$

When i=1

$=0(01)^11$

=0011

When i=2

=0$(01)^2$1

=001011

Thus the string 001011 is not present in the set so it is not a regular set.

7. Prove whether {0^n10^n | n ≥ 1} is regular set or not? (5 marks)

Let the given language L={0^n10^n, n≥1}

L={010,00100,0001000, …}

CASE 1:

Consider z= 010

Here u=0, w=0, V=1

|uv|≤n

2 ≤ 3

|v|≥1

⇒uv^iw

= 0$(1)^i$0

When i=1

=0$(1)^1$0

=010

When i=2

=0$(1)^2$0

=0110

Thus the string 0110 is not present in the set so it is not a regular set.

8. Prove whether {01^n1| n ≥ 1} is regular set or not? (5 marks)

Let the given language L ={01^n1, n≥1}

L={011,0111,01111, …}

CASE 1:

Consider z= 011

Here u=0, w=1, V=1

|uv|≤n

2 ≤ 3

|v|≥1

⇒uv^iw

= 0$(1)^i$1

When i=1

$=0(1)^1 1$

$=011$

When i=2

$=0(1)^2 1$

$=0111$

Thus the string 0111 is present in the set so it is a regular set or regular language.

9. Discuss about Chomsky normal form (CNF) with an example?

State and prove Chomsky normal form (NOV'14,NOV'15) (11 marks)

The each state that all context free grammars are equivalent to grammar with restrictions in the form of production.

The context free language is in Chomsky Normal Form (CNF) is generated by a grammar in which all productions are of the form

- ✓ A→BC or
- ✓ A→a

where

A, B, and C are variables or non-terminals and *a* is a terminal

For a given grammar, we have to eliminate

1. useless symbol
2. productions
3. unit productions

- Then check whether the production contains A→BC and A→a more than 2 non-terminals are present, so it is not a CNF and combination of terminals and non-terminals are present it is also not CNF.
- A context free grammar G is in CNF if every production is of the form A→a or A→BC and S→λ is in G if λ ∈ L (G). When λ is in L (G) we assume that S does not appear on the R.H.S of any production.

Example:

Consider the grammar for ({S, A, B}, {a,b},P,S) that has the productions:

S→bA|aB

A→bAA|aS|a

B→aBB|bS|b

Solution:

The given productions are

S→bA

S→aB

A→bAA

A→aS

A→a

B→aBB

B→bS

B→b

<u>Step 1:</u> Simplification of a Grammar

In the given grammar there are no useless symbols, no unit productions and no ε-productions.

<u>Step 2:</u> Converting CFG into CNF

P1: S→bA

P2: S→aB

P3: A→bAA

P4: A→aS

P5: A→a

P6: B→aBB

P7: B→bS

P8: B→b

<u>Step 3:</u>

The only productions P5, P8 are A→a and B→b are of the form CNF.

The following other production consists of more than two non-terminal and combination to replace to form a CNF

S→bA S→aB

A→bAA A→aS

B→aBB B→bS

Consider P1:

S→bA

Introduce a new temp variable **(C_b→b)**

∴ S→C_bA

Consider P2:

S→aB

Introduce a new temp variable **(C_a→a)**

∴ S →C_aB

Consider P3:

A→bAA

A →C_bAA (C_b→b)

Consider P4:

A→aS

∴ A→C_aS (C_a→a)

Consider P6:

B→aBB

B→C_aBB **(C_a→a)**

Consider P7:

B→bS

∴ B →C_bS (C_b→b)

Step 4:

The remaining productions **P3** and **P6** are A→CbAA , B→CaBB which are not in CNF form. Because it contains more than two non-terminals.

We have to replace them.

Again Consider P3:

A→C_bAA

Introduce a new temp variable **(D_1→AA)**

∴ **A →$C_b$$D_1$**

Again Consider P6:

B→C_aBB

Introduce a new temp variable **(D_2→BB)**

∴ **B→$C_a$$D_2$**

The Productions for the grammar in CNF form

S→C_bA | C_aB	**D_1→AA**
A→C_aS | $C_b$$D_1$| a	**D_2→BB**
B→$C_a$$D_2$| C_bS	**C_a→a**
	C_b→b

10. Explain Greibach Normal Form (GNF) with an example? (11 marks)

A context free grammar is said to be in Greibach Normal Form (GNF) if all productions are in the following form:

- ✓ A → αX
- ✓ A → a

where

- A is a non-terminal symbols
- α and a is a terminal symbol
- X is a sequence of non-terminal symbols.

Every CFL without ε can be generated by a grammar for which every productions of the form A→aα. Where A is a variable, *a* is a terminal, and α is a string of variables. This type of G is said to be GNF. So RHS should contain only one terminal symbol that should be the left most symbols on the RHS followed by 0 or more no of variables.

Greibach Normal Form (GNF) algorithm:

begin

1) **for** k=1 to n do

begin

2) **for** j=1 to k –1 do

3) **for** each production of the form $A_k \rightarrow A_j \alpha$ **do**

begin

for all production $A_j \rightarrow \beta$ **do**

add production $A_k \rightarrow \beta\alpha$;

remove production $A_k \rightarrow A_j \alpha$

end;

4) **for** each production $A_k \rightarrow A_k \alpha$ **do**

begin

add productions $B_k \rightarrow \alpha$ and $B_k \rightarrow \alpha B_k$;

remove productions $A_k \rightarrow A_k \alpha$

end;

5) **for** each production $A_k \rightarrow \beta$, where β does not

begin with A_k do

add production $A_k \rightarrow \beta B_k$

end;

end

Rules:

In Greibach Normal Form (GNF), it uses the production whose R.H.S. each starts with the terminal symbol perhaps followed by some variables or non-terminals.

Steps to solve GNF problem:

1) Simplification of CFG grammar

2) Converting CFG grammar into CNF form

3) Converting CNF form into GNF form

i. Change the variables to include suffix numbers

ii. Preliminary verification step

iii. Substitute a correct production into incorrect production

Example:

Convert the following grammar to GNF form

S→AA |a

A→SS | b

Solution:

Step1: Simplification of a Grammar

In the given grammar there are no useless symbols, no unit productions and no ε-productions.

Step2: Convert CFG into CNF form

Here all the productions are already in the CNF form.

Step3: Convert CNF form into GNF form

i. Change the variables to include suffix numbers

The given productions are in the form of CNF. So we have to convert this into GNF form,

Let rename above production by $S=A_1$ and $A=A_2$

P_1: $\mathbf{A_1 \rightarrow A_2A_2}$

P_2: $\mathbf{A_1 \rightarrow a}$

P_3: $\mathbf{A_2 \rightarrow A_1A_2}$

P_3: $\mathbf{A_2 \rightarrow b}$

ii. Preliminary verification step

The first productions i value=1 and j value=2. So i<j. No need to disturb now.

P_1: $\mathbf{A_1 \rightarrow A_2A_2}$

1<2 // temporary correct production

The Second production

P_2: $\mathbf{A_1 \rightarrow a}$ // already in GNF form

The first productions i value=2 and j value=1. So i<j. No need to disturb now.

P_3: $\mathbf{A_2 \rightarrow A_1A_2}$

2<1 // temporary correct production

The fourth production

P_4: $\mathbf{A_2 \rightarrow b}$ // already in GNF form

iii. Substitute a correct production into incorrect production

Substitute all A_1 Production value P3 production,

$\mathbf{A_1 \rightarrow A_2A_2}$ and $\mathbf{A_1 \rightarrow a}$

P_3: $A_2 \rightarrow A_1A_1$

$\mathbf{A_2 \rightarrow A_2A_2A_1}$ and $\mathbf{A_2 \rightarrow aA_1}$

Consider the production $A_2 \rightarrow A_2A_2A_1$

This productions of the form of $A_k \rightarrow A_k\ \alpha$ means

add $B_k \rightarrow \alpha$ and $B_k \rightarrow \alpha B_k$ and remove $A_k \rightarrow A_k\ \alpha$

$A_k=A_2$, k=2 and $\alpha= A_2A_1$

i) Add $B_k \rightarrow \alpha$

$B_k=B_2$

$\mathbf{B_2 \rightarrow A_2A_1}$

ii) add $B_k \rightarrow \alpha B_k$

$\mathbf{B_2 \rightarrow A_2A_1B_2}$

iii) remove $Ak \rightarrow A_k \alpha$

Now $\mathbf{A_2 \rightarrow A_2A_2A_1}$ is removed

$\therefore$ $\mathbf{B_2 \rightarrow A_2A_1}$

$\mathbf{B_2 \rightarrow A_2A_1B_2}$

Consider the production $\mathbf{A_2 \rightarrow aA_1}$

This production is of the form $A_k \rightarrow \beta$ means add $A_k \rightarrow \beta B_k$

Now $A_k=A_2$, $\beta = aA_1$, $B_k=B_2$ and k=2, then

Add $A_k \rightarrow \beta B_k$

$\therefore$ $\mathbf{A_2 \rightarrow aA_1B_2}$

$\mathbf{A_2 \rightarrow aA_1}$

Consider the production P_4: $\mathbf{A_2 \rightarrow b}$

This production of the form $A_k \rightarrow \beta$

Now $A_k=A_2$, $\beta = b$, $B_k=B_2$ and k=2, then

Add $A_k \rightarrow \beta B_k$

$\therefore$ $\mathbf{A_2 \rightarrow bB_2}$

$\mathbf{A_2 \rightarrow b}$

Substitute all A_2 production value in B_2 productions, we get

$B_2 \rightarrow A_2A_1$

$\mathbf{B_2 \rightarrow bB_2A_1 \mid bA_1 \mid aA_1A_1 \mid aA_1B_2A_1}$

$B_2 \rightarrow A_2A_1B_2$

$\mathbf{B_2 \rightarrow bB_2A_1B_2 \mid bA_1B_2 \mid aA_1A_1B_2 \mid aA_1B_2A_1B_2}$

P_1: $A_1 \rightarrow A_2A_2$

$\mathbf{A_1 \rightarrow aA_1B_2A_2 \mid aA_1A_2 \mid bB_2A_2 \mid bA_2}$

Consider the production P_2: $A_1 \rightarrow a$

This production of the form $A_k \rightarrow \beta$

Now $A_k=A_1$, β =a, $B_k=B_1$ and k=1, then

Add $A_k \rightarrow \beta B_k$

$\therefore$ $\boxed{\begin{array}{c} A_1 \rightarrow aB_1 \\ A_1 \rightarrow a \end{array}}$

Substitute all A_1 production value in above P_3 productions, we get

P_3: $A_2 \rightarrow A_1A_1$

$$\boxed{A_2 \rightarrow aB_1A_1 \mid aA_1}$$

The resulting GNF grammar productions are

$$\boxed{\begin{array}{l} A_1 \rightarrow aA_1B_2A_2 \mid aA_1A_2 \mid bB_2A_2 \mid bA_2 \mid aB_1 \mid a \\ A_2 \rightarrow aB_1A_1 \mid aA_1 \mid aA_1B_2 \mid bB_2 \mid b \\ B_2 \rightarrow bB_2A_1 \mid bA_1 \mid aA_1A_1 \mid aA_1B_2A_1 \mid bB_2A_1B_2 \mid bA_1B_2 \mid aA_1A_1B_2 \mid aA_1B_2A_1B_2 \end{array}}$$

11. Explain Ambiguous Grammar with example? (11 marks)

- A context free grammar G such that string or word has two parse trees is said to be ambiguous.
- For some CFG's, it is possible to find a terminal string with more than one parse tree, or equivalently, more than one leftmost derivation or more than one rightmost derivation. Such a grammar is called ambiguous grammar.
- The grammar G is said to be ambiguous when the same start symbol produces same language with 'n' number of left most and right most derivation.

Example 1:

Consider the Grammar G is the grammar

S→S b S | a

Prove that given grammar is ambiguous.

Solution:

Consider the string or word **ω= a b a b a b a**

LMD 1:

S→S b S

→a b S

➔a b S b S

➔a b a b S

➔a b a b S b S

➔a b a b a b S

➔**a b a b a b a**

RMD:

S → S b S

➔ S b a

➔ S b S b a

➔ S b a b a

➔ S b s b a b a

➔ S b a b a b a

➔**a b a b a b a**

If both LMD and RMD give same string, it is called ambiguous.

Example 2:

Show that the CFG with production S->a|Sa|bSS|SSb|SbS is ambiguous?

Solution:

Consider the input string or word w=babaa

LMD:

S →bSS

➔baS

➔babSS

➔ babaS

➔ babaa

RMD:

S→bSS

➔bSa

➔bSbSa

➔ bSbaa

➔babaa

Both LMD and RMD are same. Hence the given grammar is ambiguous.

Example 3:

Show that the grammar S→aB|ab

A→aAB|a

B→ABb|b is ambiguous.

Solution:

Consider the input string or word ω =ab

LMD

The LMD of G are

S→a B

→a b

S→a b

Here ab is ambiguous available by the grammar and so the given grammar is ambiguous.

12. Prove that the language, L= {a^n/n≥1} is unambiguous. Even though L there exist a ambiguous grammar generating it. (5 marks)

Let G= ({S},a,p,S}

Where p1={S→SS, S→1}

We observe that

L(G)={ a^n/n≥1}

Consider the word a^3 **ε L (G)**

Now,

LMD1:

S→SS

⇒SSS

⇒ aSS

⇒aaS

⇒aaa

LMD2:

S→SS

⇒aS

⇒aSS

⇒aaS

⇒aaa

Thus a^3 has different LMD Grammar is ambiguous. Thus there exist an unambiguous grammar generating { a^n/n≥1}.

13. Consider the grammar

E→E+E

E→E*E

E →(E)

E →id

The input string is id + id * id. Show that it is ambiguous grammar or not? (5 marks)

Solution:

The given grammar

E→E+E

E→E*E

E →(E)

E →id

LMD1:

E ===> E + E

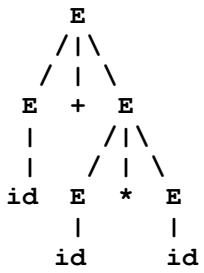
===> id+ E

===> id + E * E

===> id + id * E

===> **id + id * id**

Parse tree 1:



LMD2:

E ===> E * E

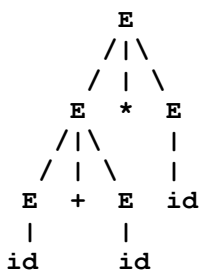
===> E + E * E

===> id + E * E

===>id + id * E

===> id + id * id

Parse tree 2:



The given input string id+id*id has two leftmost derivation and parse tree. Hence given grammar is ambiguous grammar.

Explain unambiguous grammar with an example? (6 marks)

For some CFG's, it is possible to find a terminal string with at most one parse tree, or equivalently, at most one leftmost derivation or at most one rightmost derivation. Such a grammar is called unambiguous grammar.

Example:

Consider the grammar E→E+T | T

T→T*F | F

F →(E) | id

The input string is **id + id * id** and **id * id + id**. Show that it is unambiguous grammar or not?

Solution:

The given grammar is

E→E+T | T

T→T*F | F

F →(E) | id

LMD:

E ===> E + T

===> T + T

===> F + T

===> id + T

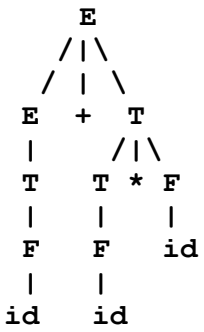
===> id + T * F

===> id + F * F

===> id + id * F

===> **id + id * id**

Parse tree:



RMD:

E ===> E + T

===> T + T

===> T * F + T

===> F * F + T

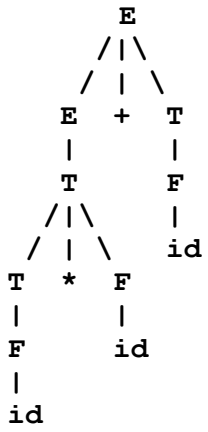
===> id * F + T

===> id * id + T

===> id * id + F

===> **id * id + id**

Parse tree:



Hence, we have unique leftmost derivation and unique rightmost derivation for the input string. Hence the given grammar is unambiguous grammar.

15.Write the applications of pumping lemma. (5 marks) (NOV 2013)

The applications of Pumping Lemma are

- A basic result called the ***pumping lemma***, which is a powerful tool for proving certain languages, is regular or not.

- It is also useful in the development of algorithms to answer certain questions concerning finite automata, such as whether the language accepted by a given FA is finite or infinite.
- If a language is regular, it is accepted by a DFA M= (Q, Σ, δ, q0, F) with particular number of states.
- Let L be a regular set. Then there is a constant n such that if Z is any word in L, and |z|≥n. we may write Z=uvw in such a way that |uv| ≤ n, |v| ≥ 1and for all L ≥0 then uv^iw is in L.

1. The pumping lemma is useful in proving that certain sets are not regular. The application is an "adversary arguments" of the following form.
2. Select the language L to prove non-regular.
3. The "adversary" picks n, the constants mentioned in the pumping lemma.
4. Select a string *z* in L.
5. The adversary breaks *z* into *u, v* and *w,* subject to the constraints that |uv|≤n and |v|≥ 1.
6. A contradiction to the pumping lemma by showing, for any *u, v,* and *w,* determined by the adversary, that there exists an *i* for which $\boldsymbol{uv^iw}$ is not in L. It may then be concluded that L is not regular.

16.Simplify the following grammar and find its equivalent in CNF

S→AB | CA, B→ BC | AB , A→a , C→ aB | b (5 mark) UQ MAY'15)

Solution:

CNF Format:

NT → NTNT
NT → T

Where , NT – Non terminal

T – Terminal

Given

P : S→AB | CA

B→ BC | AB

A→a

C→ aB | b

Step 1:

In the given grammar the below three production are already in CNF form. They are :

S→AB | CA

B→ BC | AB

A→a

To convert the remaining production into CNF, do the following

Simplification of grammar:

(i) Elimination of ε production

ε production of the form :

NT → ε

There is ε production in the given grammar. Therefore P' is same as P

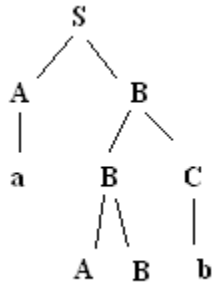
(ii) Elimination of Unit production

Unit production of the form :

NT → NT

There is Unit production in the given grammar. Therefore P'' is same as P'

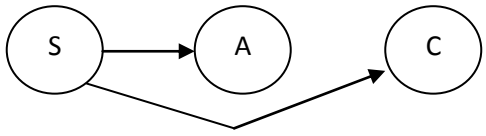
(iii) Elimination of Useless symbol production



The symbol B doesn't lead to a terminal string .therefore B is the useless symbol. Therefore eliminate B from P''

Before elimination		**After elimination**	
P'' :	S→AB | CA	P''' :	S→ CA
	B→ BC | AB		A→a
	A→a		C→ b
	C→ aB | b		

(iv) Elimination of symbol from P''' that is not reachable from start state



Now all the states are reachable from the start state. .Hence there is no elimination from p'''.

Simplified CNF Productions	
P''' :	S→ CA
	A→a
	C→ b

17.Find the GNF equivalent of the grammar S→ AA | 0 , A → SS | 1 (6 mark) (MAY 2015)

Solution:

GNF Format:

NT → TNTNTNT……..
NT → T

Where , NT – Non terminal

T – Terminal

OR

A context free grammar is said to be in Greibach Normal Form (GNF) if all productions are in the following form:

A →αX
A →a

where

- ✓ A is a non-terminal symbols
- ✓ α is a terminal symbol
- ✓ X is a sequence of non-terminal symbols.

Every CFL without ε can be generated by a grammar for which every productions of the form A→aα. Where A is a variable, *a* is a terminal, and α is a string of variables. This type of G is said to be GNF. So RHS should contain only one terminal symbol that should be the left most symbols on the RHS followed by 0 or more no of variables.

Given

S→ AA | 0

A → SS | 1

Convert the following grammar to GNF form

S→ AA

S→ 0

A → SS

A→ 1

Solution:

Step1: Simplification of a Grammar

In the given grammar there are no useless symbols, no unit productions and no ε-productions.

Step2: Convert CFG into CNF form

Here all the productions are already in the CNF form.

Step3: Convert CNF form into GNF form

iv. Change the variables to include suffix numbers

The given productions are in the form of CNF. So we have to convert this into GNF form,

Let rename above production by S=A_1 and A=A_2

P_1: $\mathbf{A_1 \rightarrow A_2A_2}$

P_2: $\mathbf{A_1 \rightarrow 0}$

P_3: $\mathbf{A_2 \rightarrow A_1A_1}$

P_3: $\mathbf{A_2 \rightarrow 1}$

v. Preliminary verification step

The first productions i value=1 and j value=2. So i<j. No need to disturb now.

P_1: $\mathbf{A_1 \rightarrow A_2A_2}$

1<2 // temporary correct production

The Second production

P_2: $\mathbf{A_1 \rightarrow 0}$ // already in GNF form

The first productions i value=2 and j value=1. So i<j. No need to disturb now.

P_3: $\mathbf{A_2 \rightarrow A_1A_1}$

2<1 // temporary correct production

The fourth production

P_4: $\mathbf{A_2 \rightarrow 1}$ // already in GNF form

vi. Substitute a correct production into incorrect production

Substitute all A_1 Production value P3 production,

$\mathbf{A_1 \rightarrow A_2A_2}$ and $\mathbf{A_1 \rightarrow 0}$

P_3: $A_2 \rightarrow A_1A_1$

$\mathbf{A_2 \rightarrow A_2A_2A_1}$ and $\mathbf{A_2 \rightarrow 0A_1}$

Consider the production $\mathbf{A_2 \rightarrow A_2A_2A_1}$

This productions of the form of $A_k \rightarrow A_k\ \alpha$ means

add $B_k \rightarrow \alpha$ and $B_k \rightarrow \alpha B_k$ and remove $A_k \rightarrow A_k\ \alpha$

$A_k=A_2$, k=2 and $\alpha= A_2A_1$

iv) Add $B_k \rightarrow \alpha$

$B_k=B_2$

$\mathbf{B_2 \rightarrow A_2A_1}$

v) add $B_k \rightarrow \alpha B_k$

$\mathbf{B_2 \rightarrow A_2A_1B_2}$

vi) remove $Ak \rightarrow A_k \alpha$

Now $\mathbf{A_2 \rightarrow A_2A_2A_1}$ is removed

$\therefore$ $\mathbf{B_2 \rightarrow A_2A_1}$

$\mathbf{B_2 \rightarrow A_2A_1B_2}$

Consider the production $\mathbf{A_2 \rightarrow 0A_1}$

This production is of the form $A_k \rightarrow \beta$ means add $A_k \rightarrow \beta B_k$

Now $A_k=A_2$, $\beta = 0A_1$, $B_k=B_2$ and k=2, then

Add $A_k \rightarrow \beta B_k$

$\therefore$ $\mathbf{A_2 \rightarrow 0A_1B_2}$

$\mathbf{A_2 \rightarrow 0A_1}$

Consider the production $\mathbf{P_4}$: $\mathbf{A_2 \rightarrow 1}$

This production of the form $A_k \rightarrow \beta$

Now $A_k=A_2$, $\beta = 1$, $B_k=B_2$ and k=2, then

Add $A_k \rightarrow \beta B_k$

$\therefore$ $\mathbf{A_2 \rightarrow 1B_2}$

$\mathbf{A_2 \rightarrow 1}$

Substitute all A_2 production value in B_2 productions, we get

$B_2 \rightarrow A_2A_1$

$\mathbf{B_2 \rightarrow 1B_2A_1 \mid 1A_1 \mid 0A_1A_1 \mid 0A_1B_2A_1}$

$B_2 \rightarrow A_2A_1B_2$

$\mathbf{B_2 \rightarrow 1B_2A_1B_2 \mid 1A_1B_2 \mid 0A_1A_1B_2 \mid 0A_1B_2A_1B_2}$

P_1: $A_1 \rightarrow A_2A_2$

$\mathbf{A_1 \rightarrow 0A_1B_2A_2 \mid 0A_1A_2 \mid 1B_2A_2 \mid 1A_2}$

Consider the production $\mathbf{P_2}$: $\mathbf{A_1 \rightarrow 0}$

This production of the form $A_k \rightarrow \beta$

Now $A_k=A_1$, $\beta = 0$, $B_k=B_1$ and k=1, then

Add $A_k \rightarrow \beta B_k$

∴

$A_1 \rightarrow 0B_1$
$A_1 \rightarrow 0$

Substitute all A_1 production value in above P_3 productions, we get

P_3: $A_2 \rightarrow A_1A_1$

$A_2 \rightarrow 0B_1A_1 \mid 0A_1$

The resulting GNF grammar productions are

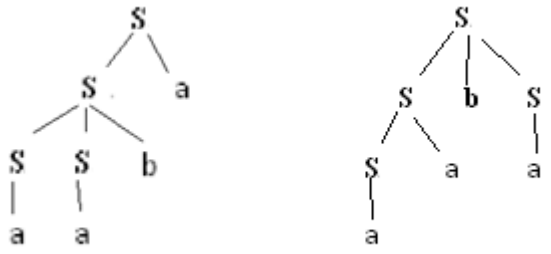
$A_1 \rightarrow 0A_1B_2A_2 \mid 0A_1A_2 \mid 1B_2A_2 \mid 1A_2 \mid 0B_1 \mid 0$
$A_2 \rightarrow 0B_1A_1 \mid 0A_1 \mid 0A_1B_2 \mid 1B_2 \mid 1$
$B_2 \rightarrow 1B_2A_1 \mid 1A_1 \mid 0A_1A_1 \mid 0A_1B_2A_1 \mid 1B_2A_1B_2 \mid 1A_1B_2 \mid 0A_1A_1B_2 \mid 0A_1B_2A_1B_2$

18. Show that the grammar S → a |Sa |bSS | SSb | SbS is ambiguous (5 mark) (MAY 2015)

Ambiguous grammar:

A grammar G is ambiguous if and only if there exist at least one string w εT* .For which two or more different parse tree exist by applying either by LMD or RMD

Let the string w = aaba , parse tree for the string shown below.



For the same string two different parse trees can be generated. Hence the given grammar is ambiguous.

19. Consider the productions

S→aB | bA

A→aS | bAA |a

B → bS |aBB |b

For the string aaabbabbba ,find a left most derivation. (6 mark) (UQ MAY'15)

S → a<u>B</u>

→ aa<u>B</u>B B →aBB

→ aaaBBB B →aBB

→ aaabBB B → b

→ aaabbB B → b

→ aaabbaBB B →aBB

→ aaabbabB B → b

→ aaabbabbS B → bS

→ aaabbabbbA S→ bA

→ aaabbabbba A→a

Hence the proof.

20. Convert the grammar G with productions to Chomsky normal form

S→aSA | aAA | b , A → bBBB , B→ b | ε (11 mark) UQ MAY'15)

Solution:

CNF Format:

NT → NTNT
NT → T

Given

P : S→ aSA | aAA | b

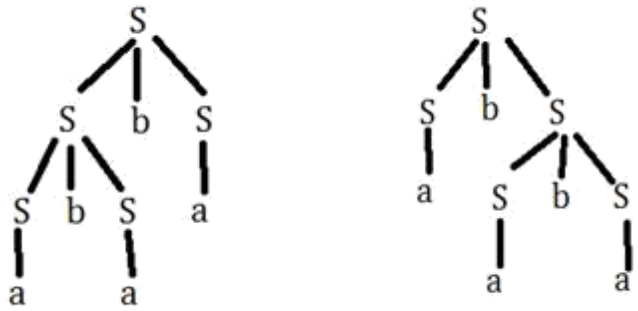
A → bBBB

B→ b | ε

Refer class notes

21. If G is the grammar S SbS | a.Show that G is ambiguous. (6 MARKS) (NOV 2015)

Let the string w = ababa, below is the two different derivation tree for the same string.



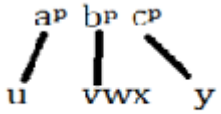
Hence the grammar is ambiguous.

22. Show that the language L = { $a^i b^i c^i$ | i>=1} is not context free .(7 marks)(NOV'15)

Solution:

The given language L = { $a^i b^i c^i$ | i>=1}

Let i=p



$u = a^p$

$vwx = b^{p-i}$

$y = c^p$

$z = uv^iwx^iy$

$= u\ v^{i-1}\ v^1\ w\ x^{i-1}\ x^1 y$

$= uvwx\ v^{i-1}\ x^{i-1}\ y$

$z = uvwx\ vx^{i-1}\ y$ --------(1)

$vx = b^{p-m}$

sub . in eqtn (1)

$z = a^p b^p (b^{p-m})^{i-1} c^p$

$z = a^p b^p b^{(p-m)(i-1)} c^p$ --------(2)

case (i)

I = 1 p = 3 m= 2

sub . in eqtn (2)

$z = a^3 b^3 b^{(3-2)(1-1)} c^3$

$z = a^3 b^3 b^{(1)(0)} c^3$

$z = a^3 b^3 c^3 \ \varepsilon\ L$

case (ii)

I = 2 p = 3 m= 2

sub . in eqtn (2)

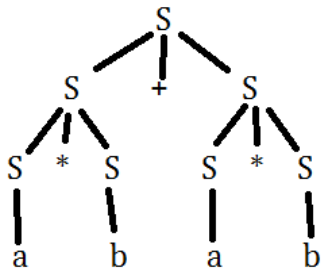
$z = a^3 b^3 b^{(3-2)(2-1)} c^3$

$z = a^3 b^3 b^{(1)(1)} c^3$

$z = a^3 b^4 c^3 \notin L$

the given language is not accepted.

23. Find the derivation tree of a * b + a* b given that a * b + a* b is in L(G).where G is given by S → S + S , S → S * S ,S → a|b (3 marks) (NOV'15)



Pondicherry University Questions

2 Marks

1. **List the applications of the pumping lemma? (APR 2014) Ref.Qn.No.2**
2. What is Homomorphism? **(APR 2014) (Ref.Qn.No.4)**
3. What is context free grammar? **(NOV 2013) (Ref.Qn.No.8)**
4. What is left most derivation? **(APR 2013,MAY'15) (Ref.Qn.No.21)**
5. What is ambiguous grammar? Give example. **(APR, NOV 2013) (Ref.Qn.No.28)**
6. Define Chomsky Normal Form (CNF)? (APR 2012,MAY'15) **(Ref.Qn.No.39)**
7. Is the grammar G with following productions ambiguous? Justify. **(NOV 2012)**

   S→SS | aSb | bSa | ε **(Ref.Qn.No.43)**
8. List any two applications of Regular expressions? **(APR 2014) (Ref.Qn.No.2)**
9. Specify CNF theorem. **(APR 2012) (Ref.Qn.No.39)**
10. Write down GNF. **(APR 2012) (Ref.Qn.No.40)**
11. Eliminate useless productions from the following: **(NOV 2012) (Ref.Qn.No.43)**

    S→a| aA | B| C A→aB| ε B→Aa C→cCD D→ddd
12. What is meant by derivation? **(NOV'15) (Ref.Qn.No.13)**
13. Define derivation trees? (NOV 2014) **(Ref.Qn.No.14)**
14. Define Greibach Normal Form (GNF)? **(APR 2012,NOV'15) (Ref.Qn.No.40)**
15. What is sentential form? (MAY 2015) **(Ref.Qn.No.44)**
16. Give an example for an ambiguous grammar (MAY'15) **(Ref.Qn.No.45)**

11 Marks

1. Describe on Closure Properties of Regular Sets? **(APR'14,NOV,15) (Ref.Qn.No.1)**
2. Discuss on Derivation tree or Parse trees? **(APR 2012, 2014) (Ref.Qn.No.3)**
3. Discuss on Pumping Lemma for regular language and give its applications. **(APR 2013)**
4. Discuss briefly about Context-Free Grammars. **(APR 2012, 2013) (Ref.Qn.No.2)**
5. Write the applications of pumping lemma. **(5) (NOV 2013) (Ref.Qn.No.15)**
6. Verify whether the set of all strings that do not have consecutive 0's is regular set or not. **(6) (NOV 2013)**
7. Let G= (V, T, P, S) be a context free grammar. Then prove that s→a a if and only if there is a derivation tree in grammar G with yield a. **(NOV 2013)**
8. State and prove the properties of regular language. Let L the set of all strings over the alphabet of the form " x + y = z ", where x, y and z are the binary representation of three numbers such that the sum of x and y equals z. Prove that the language L is not regular using the Pumping Lemma. **(NOV 2012)**
9. Construct a RE for the language which accepts all strings with at least two c's over the set ∑={c,b} and design an equivalent finite automata for the same. **(NOV 2012)**
10. Simplify the following grammar and find its equivalent in CNF

    S→AB | CA, B→ BC | AB , A→a , C→ aB | b (5 mark) (MAY 2015) **(Ref.Qn.No.16)**

11. Find the GNF equivalent of the grammar S→ AA | 0 , A → SS | 1(6 mark) (MAY 2015) **(Ref.Qn.No.17)**

12. Consider the productions

S→aB | bA A→aS | bAA |a B → bS |aBB |b

For the string aaabbabbba ,find a left most derivation. (6 mark) (MAY 2015) **(Ref.Qn.No.19)**

13. Show that the grammar S → a |Sa |bSS | SSb | SbS is ambiguous (5 mark) (MAY 2015) **(Ref.Qn.No.18)**

14. State and prove Chomsky normal form (NOV'14,NOV'15) **(Ref.Qn.No.9)**

15. Verify whether { o^{2^n} | n>=1} is regular set or not? (NOV'14) **(Ref.Qn.No.5)**

16. Convert the grammar G with productions to Chomsky normal form

S→aSA | aAA | b , A → bBBB , B→ b | ε (11 mark) UQ MAY'15) **(Ref.Qn.No.20)**

17. If G is the grammar S→ SbS | a.Show that G is ambiguous (4 marks) **(NOV'15)**

(Ref.Qn.No.21)

18. Show that the language L = { $a^i b^i c^i$ | i>=1} is not context free **.(NOV'15) (Ref.Qn.No.22)**

19. Find the derivation tree of a * b + a* b given that a * b + a* b is in L(G).where G is given by S → S + S , S → S * S ,S → a|b (3 marks) (NOV'15) **(Ref.Qn.No.23)**

UNIT III

Pushdown Automata and Parsing Algorithms: Pushdown Automata and Context-Free Languages; Top-down parsing and Bottom-up parsing, Properties of CFL, Applications of Pumping Lemma, Closure properties of CFL and decision algorithms

---

2 Marks

1.What is push down automata? (NOV'14)

An automata equivalent to context free language is push down automata Finite automata cannot recognize all context free languages. Since some CFG's are not regular.

Finite automaton has finite memories, whereas recognition of CF language may require storing unbounded productions.

2. Give an example for PDA

Let l={ww^k :wε{a,b}*} to handle this language,we need more than unlimited counting. We need to store an matches the sequence of symbols in reverse order.

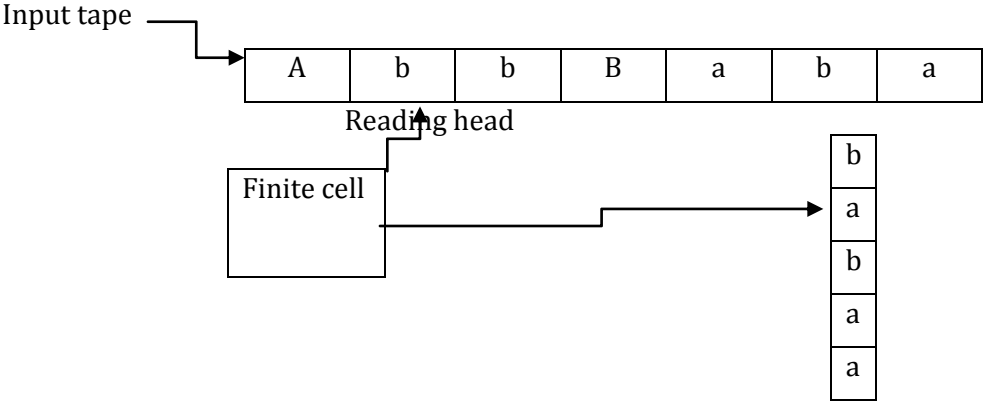
PDA is a machine similar to FA that will accept CFL has more powerful.

3. What technique is used for PDA?

STACK(FIRST IN FIRST OUT) Is the technique used for PDA.

4.How stack is being implemented in PDA?

The symbol can be entered or removed only on the top of storage. When a symbol is added on top, the symbol previously on the top becomes the second and so on. Similarly when a symbol is removed from top of stack the symbol previously second from top becomes top symbol and so on.



5.Difference between context free language and regular language?

S.No	Context Free Language	Regular Language
1	A language cannot be described by DFA/NFA.	Can be described by DFA/NFA
2	There is no memory.	There is memory
3	This may be ambiguous or unambiguous.	this is always unambiguous.
4	With the help of context free grammar we can generate this language.	With the help of regular grammar we can generate Regular Language.

6. Difference between CFL and PUSHDOWN AUTOMATA?

S.No	Context Free Language	Push Down Automata
1	A language cannot be described by DFA/NFA.	Push down automata can accept CFL. It is essentially an NFA with a stack.
2	There is no memory.	It has a memory, which can be used to count the number.

7. Comparison of NFA and PDA

NFA

1. (P,a,q) ε Δ means if machine M is in state P, then on reading "a" from input tape go to stack q.
2. (P, ε,q) ε Δ means if machine M is in state P, goes to state q, without consuming input.

PDA

1. ((P,a,β),(q,y)) ⟶ if machine is in state P, the symbol read from input state is 'a', and β is on top of stack, goes to state q, and replace β by y on top of stack.
2. ((s,a,e),(s,a)) ⟶ if machine M is in state s reads 'a', remains in state s and push a onto stack.
3. ((s,c,e),(f,e)) ⟶ if read 'c' in state s and stack is empty, goes to final state f and nothing to push onto the stack.
4. ((s,e,e),(f,e)) ⟶ if in state s, goes to state f.
5. ((f,a,q),(f,φ)) ⟶ if read a in stack f, remain in state f and pop a from stack.
6. PDA's are non-deterministic.

8.Say true or false

Is PDA is non deterministic?- true

9. What are the properties of PDA?

A push down automata M is defined by $(Q,\Sigma,\acute{\Gamma},\delta,q_0,z_0.F)$ where,

Q is the finite set of states

Σ is the alphabet called input alphabet

Γ́ is the stack alphabet , is a finite alphabet os stack symbols.

q_0 ε Q is the stack state/initial state

z_0 in Γ́ is the particular stack symbol called start symbol.

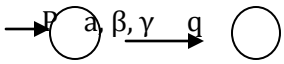
F≤Q is the set of final states.

δ is the transition relation.

i.e, δ is a subset of $(Q \times \Sigma \cup \{e\} \times \acute{\Gamma}^*)$ ⟶ $(2^{Q \times T^*})$

10. What is the transition function?

1. Let ((P,a,β),(q,γ)) ε δ
2. It means that we,
   a. Read a from tape.
   b. Pop the string β from stack.
   c. Move from stack P to state q.
   d. Push string γ onto stack.
3. We will draw it as



11. What are the functions of the PDA?

a. Read a from tape.

b. Pop the string β from stack.

c. Move from stack P to state q.

d. Push string γ onto stack.

12. What do you mean by push operation?

1.When we push β, we must push the symbols of β as we read them right to left.

2.i.e, if we push abc, then we can pop a, then pop b, then pop c. so e push like this

Left most

a
b
c

Rightmost

13. What do you mean by pop operation?

When we pop γ, we pop the symbols of γ as we read them from left to right.

14. How will we know whether the PDA has accepted the given string or not?

1. Acceptance by final state

   Let M=(Q,Σ,Ŕ,δ,q_0,z_0.F) be a PDA. Then, the language accepted by M is set of strings that,

   L(M)=(w/(q_0,w, z_0) ⊢* (p, ε, γ) for some P ε F, γ ε Ŕ*).

2. Acceptance by empty stack:

   Let N(M) be the language accepted by M is given by,

   N(M)= (w/(q_0,w, z_0) ⊢* (p, ε, ε) for some P ε Q.

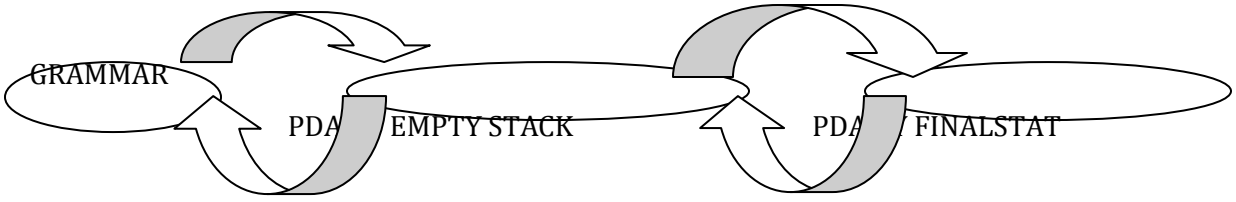
15. What is the equivalence of PDA and CFG?

Equivalence of PDA and CFG

A language is generated by a CFG.

a) If and only if it is accepted by a PDA by empty stack.

b) If and only if it is accepted by a PDA by final state.



16. What are the steps to convert CFL TO PDA?

1) For a context free grammar G, there is a equivalent CFG with GNF.
2) We can construct PDA for the CFG + GNF.
3) The PDA we are about to construct will represent the derivation by keeping the variables in the right part of the sentential form on its stack, while the left part, consisting of terminals is identical with the input read.
4) We begin by pumping the start symbol on the stack. After that, to simulate the application of production A→ ax, we must have the variable A on top of stack and terminal a as the input symbol. The variable on the stack is removed and replaced by the variable string x.

17. Give an example for the PDA that accepts the language of the grammar?

S → AB

A → aA / e

B → aBb / e (e = null state) and check for string aaaabb.

Solution:

We can construct PDA

M = (Q, Σ, Γ, δ, q0, Z, F) where

Q = {s, f}

Σ = {a, b} terminals of grammar

Γ = variables of grammar U {Z0}

= {S, A, B, a, b, Z0}

F = {f}

δ is

R1 : δ(S, *ε*, Z0) = { (f, SZ0)}

R2 : δ(f, *ε*, S) = { (f, AB)} for prod 1

R3 : δ(f, ϵ, A) = { (f, aA)} for prod 2

R4 : δ(f, ϵ, A) = { (f, ϵ)} for prod 2

R5 : δ(f, ϵ, B) = { (f, aBb), (f, ϵ)} for prod 2

R6 : δ(f, a, a) = { (f, ϵ)}

R7 : δ(f, b, b) = { (f, ϵ)}

R8 : δ(f, ϵ, Z0) = { (f, ϵ)}

Processing of string aaaabb:

(S, aaaabb, Z0) ⊢ (f, aaaabb, SZ0) – R1

⊢ (f, aaaabb, ABZ0) – R2

⊢ (f, aaaabb, aABZ0) – R3

⊢ (f, aaabb, ABZ0) – R6

⊢ (f, aaabb, aABZ0) – R3

⊢ (f, aabb, ABZ0) – R6

⊢ (f, aabb, BZ0) – R4

⊢ (f, aabb, aBbZ0) – R5

⊢ (f, abb, BbZ0) – R6

⊢ (f, abb, aBbbZ0) – R5

⊢ (f, bb, BbbZ0) – R6

⊢ (f, bb, bbZ0) – R5

⊢ (f, b, bZ0) – R7

⊢ (f, ϵ, Z0) – R7

⊢ (f, ϵ) – R8

Hence the string is accepted.

18.Construct A PDA USING THIS INPUT abcbaGiven G(V, T, P, S) where V= {S, a, b, c} T= {a, b, c} P= {S → aSa, S→ bSb, S → c}.

Solution

M = (Q, Σ, Γ, δ, q0, Z0, F) be a PDA.

Q = {P, q} P = start state q= final state

Σ = {a, b, c} terminals of grammar

Γ = {S, a, b, c} U {Z0}

F = {q}

δ is

R1 : δ(P, ϵ, Z0) = {(q, SZ0)} start symbol put on the stack

R2 : δ(q, ϵ, S) = {(q, aSa)} prod 1

R3 : δ(q, ϵ, S) = {(q, bSb)} prod 2

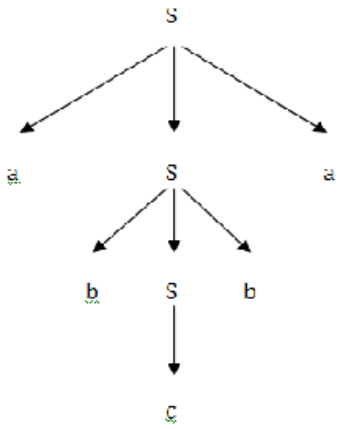
R4 : δ(q, ϵ, S) = {(q, c)} prod 3

R5 : δ(q, a, a) = {(q, ϵ)}

R6 : δ(q, b, b) = {(q, ϵ)}

R7 : δ(q, c, c) = {(q, ϵ)}

R8 : δ(q, ϵ, Z0) = {(q, ϵ)}



Processing of abcba:

(P, abcba, Z0) ⊢ (q, abcba, SZ0) – R1

⊢ (q, abcba, aSaZ0) – R2

⊢ (q, bcba, SaZ0) – R5

⊢ (q, bcba, bSbaZ0) – R3

⊢ (q, cba, SbaZ0) – R6

⊢ (q, cba, cbaZ0) – R4

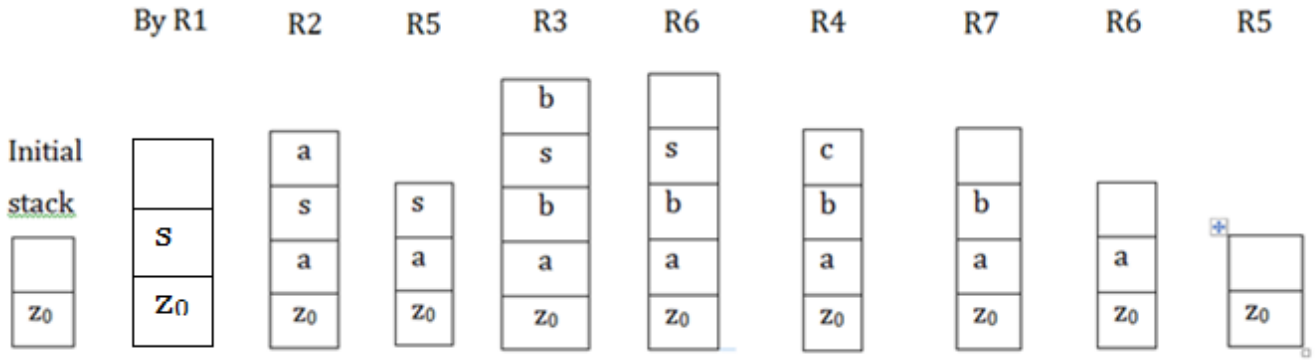
⊢ (q, ba, baZ0) – R7

⊢ (q, a, aZ0) – R6

⊢ (q, ϵ, Z0) – R5

⊢ (q, ϵ) – R8

Hence string is accepted. The stack variable is as follows.

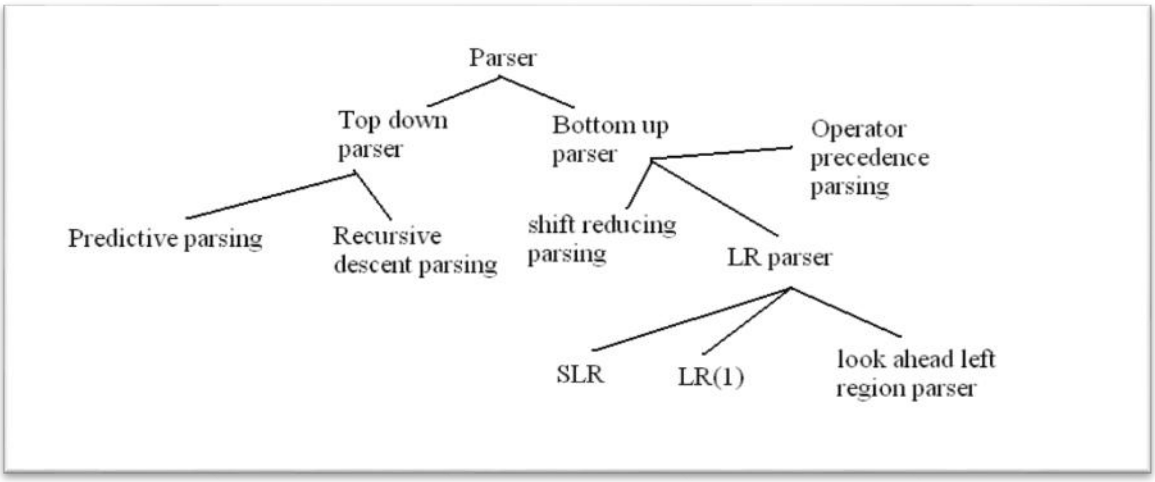


19. What is parsing?

Processor recognizes the input. It is a machine which is present in the automata m/c. Tokens are identified by the parsing. Tokens are nothing but identifiers or keywords. There are two types of parsing. They are

1. Bottom up
2. Top down (Recursive descent parsing)

20. Give a schematic representation of parsing?



21. What is bottom up parser?

It is also called as shift reducing parsing. Implementation of SRP called operator parsing. A general format or method of SRD is LR parsing which is used in automatic parser. Shift reduce parser consists for a given i/p string "reducing" a string ω to start symbol of a G. At each reduction a particular substring is matching right side of a production is replaced by symbol on the left of that production.

22. What is the other name of bottom up parser?

It is also called as shift reducing parsing

23. What are the steps included in bottom up parsing?

1. Right most derivation
2. Left most derivation
3. Identify handles/production
4. Stack implementation

24. What are the two operations performed in stack implementation?

Action contains 2 operations

1. Shift
2. Reduce

25.Give an example for bottom up parsing?

E→E+E/E * E /(E)/id w = id1+id2*id3

1. Right most derivation 2.Left most derivation
3.identify handles/production 4.Stack implementation

Right most derivation

E→E+E

➔E+E*E

➔E+E*id3

➔E+id2*id3

➔id1+id2+id3

Left most derivation

E→E*E

➔E+E*E

➔id1+E*E

➔id1+id2*E

➔id1+id2*id3

Identify production handles

E→E*E

E→E+E

E→id1|id2|id3

Stack implementation:

Action contains 2 operations

1. Shift 2.Reduce

STACK	INPUT	ACTION
$	id1+id2+id3$	Shift id1 to stack
$id1	+id2*id3$	Reduce E→id1
$E	+id2*id3$	Shift
$E +	+id2*id3$	Shift
$E + id2	*id3$	Reduce E→id2
$E + E	*id3$	Reduce E→E+E
$E	*id3$	Shift *
$E *	id3$	Shift
$E * id3	$	Reduce E→id3
$E * E	$	Reduce E→E+E
$E	$	Accepted

26. What do you mean by top down parsing?

It involves backtracking ie marking repeated scans of the input. It can be viewed as an attempt to find a LMD for an input string. Equivalently, it can be viewed as attempting to construct a parse tree. For the input starting from the root and creating the nodes of the parse tree in preorder.

27. What is the other name for top down parsing?

TOP DOWN PARSING is otherwise called as RECURSIVE DESCENT PARSING

28. What are the steps involved in top down parsing?

It has 5 steps. They are

1. Elimination of left recursion.
2. Leftmost derivation
3. Rightmost derivation
4. Identify handles
5. Stack implementation

29. What are the difficulties in top down parsing?

1. Left recursion

A grammar a is said to be left recursion if it has a non-terminal A such that there is a derivation A➔ Aα for some α. A left- recursion grammar can cause a top-down parser to go into an infinite loop. This cycling will surely occur on an erroneous input string, it may also occur on legal inputs, depending on the order in which the alternatives for A are tried. Therefore to use top-down parsing, we must eliminate left recursion from the grammar.

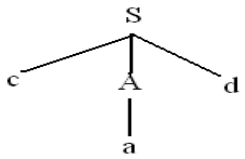
2. Backtracking

If we make a sequence of erroneous expansion and subsequently discover a mismatch, we may have to undo the semantic effects of machine this erroneous expansions.

Eg: Entries made in the symbol table might have to remove. Since undoing semantic actions requires a substantial overhead, it is reasonable to consider top-down parsers that do no backtracking.

The order in which alternatives are tried can affect language accepted

Eg: We used a and ab as the order of alternatives for A. we could fuel to accept cabd.



With parse tree already matched, the failure of next input symbol, B to match, would imply that the alternate C A d for s was wrong.

30. What are the closure properties of CFL? (NOV'15)

1) Closure under union
2) Closure under concatenation
3) Closure under kleen star

31.The set of all strings aver alphabet { a,b} with exactly twice as many a's and b's.

Solution

L ={ the set of all the strings over the alphabet { a,b } with exactly twice as many a's and b's }

Construct the grammar, G = {N,T,P,S}

Where N= {S}

T= {a,b}

P= { S→ s a s b s a s,

S→ s a s a s b s,

S→ s b s a s a s,

S→ε }

S→s a s b s a s

➔s as a s b s a s b s a s

➔s b s a s a s a s a s b s a s b s a s

➔b a a a a b a b a

Since all the productions rules are involved on the rhs 2 a's and 1 b's.

The effect of applying each of them at any point of derivation to provide twice as many a's and b's to get terminal strings. So we have to finally apply S→ ε certain no. of times. Hence G generates the language L.

32. **What do you mean by homomorphism?**

A string homomorphism is a function on strings that works by substring a particular sting for each symbol. Eg. h(0) = ab h(1) = is a homomorphism, where replace all 0's by ab and replace all 1's by . Let w = 0011 h(w) = abab

33. **Mention any two applications of pumping lemma (UQ:MAY'12)**

Pumping lemma is used to check if a language is regular or not.

(i) Assume that the language(L) is regular.z=uvw

(ii) Select a constant 'n'.

(iii) Select a string(z) in L, such that |z|>n.

(iv) Split the word z into u,v and w such that |uv|<=n and |v|>=1.

(v) You achieve a contradiction to pumping lemma that there exists an 'i' Such that uviw is not in L.Then L is not a regular language.

34. **Define Chomsky normal form. (UQ:MAY'13)**

Chomsky normal form(CNF):

If the CFG is in CNF if it satisfies the following conditions - All the production must contain only one terminal or only two variables in the right hand side.

35. **What is multiple tracks Turing machine? Or Give the significance of multiple tracks. (UQ:APR/MAY'13)**

A Turing machine in which the input tape is divided into multiple tracks where each track is having different inputs is called multiple tracks Turing machine.

36. **State the properties that are not closed under CFL (UQ:NOV'12)**

CFL are not closed under intersection , complementation. Closure properties of CFL's are used to prove that certain languages are not context free.

37. **State the difference between top down parsing and bottom up parsing (UQ:NOV'12)**

• Top-down parsers: starts constructing the parse tree at the top (root) of the parse tree and move down towards the leaves. Easy to implement by hand, but work with restricted grammars.

examples:

- Predictive parsers (e.g., LL(k))

• Bottom-up parsers: build the nodes on the bottom of the parse tree first. Suitable for automatic parser generation, handle a larger class of grammars.

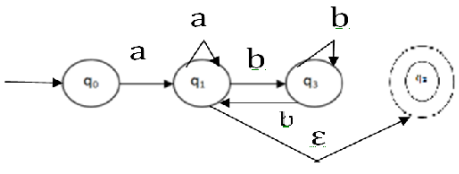
examples:

– shift-reduce parser (or LR(k) parsers)

38. What is recursively enumerable languages (UQ:NOV'14)

The languages that is accepted by TM is said to be recursively enumerable (r. e) languages. Enumerable means that the strings in the language can be enumerated by the TM. The class of r. e languages includes CFL's.

39. **Design PDA for the language L={ $a^n b^{3n}$ | n >= 0}**



36. State decision algorithm?

Decision algorithm for CFL is mainly to check the string generated by a language is

- ✓ empty,
- ✓ finite and
- ✓ infinite

11 marks

1.Discuss about PDA or Explain the Deterministic push down automata (UQ:DEC'09,MAY'10,MAY'15)

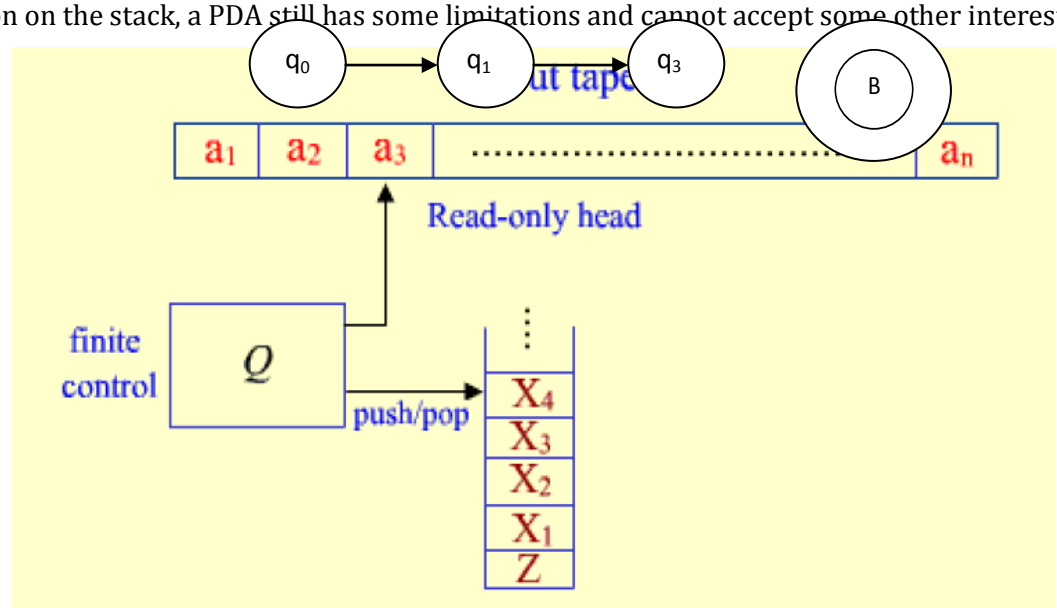
PUSH DOWN AUTOMATA(PDA):-

DEFNITION

Regular language can be characterized as the language accepted by finite automata. Similarly, we can characterize the context-free language as the language accepted by a class of machines called "Pushdown Automata" (PDA). Pushdown automation is an extension of the NFA. It is observed that FA has limited capability. (in the sense that the class of languages accepted or characterized by them is small). This is due to the "finite memory" (number of states) and "no external memory" involved with them.

A PDA is simply an NFA augmented with an "external stack memory". The addition of a stack provides the PDA with a last-in, first-out memory management capability. This "Stack" or "pushdown store" can be used to record a potentially unbounded information. It is due to this memory management capability with the help of the stack that a PDA can overcome the memory limitations that prevents a FA to accept many interesting languages like $\{a^n b^n \mid n>=0\}$. Although, a PDA can store an unbounded amount of information on the stack, its access to the information on the stack is limited.

It can push an element onto the top of the stack and pop off an element from the top of the stack. To read down into the stack the top elements must be popped off and are lost. Due to this limited access to the information on the stack, a PDA still has some limitations and cannot accept some other interesting languages.



As shown in figure, a PDA has three components: an input tape with read only head, a finite control and a pushdown store.The input head is read-only and may only move from left to right, one symbol (or cell) at a time. In each step,the PDA pops the top symbol off the stack; based on this symbol, the input symbol it is currently

reading, and its present state, it can push a sequence of symbols onto the stack, move its read-only head one cell (or symbol) to the right, and enter a new state, as defined by the transition rules of the PDA. PDA are nondeterministic, by default. That is, - transitions are also allowed in which the PDA can pop and push, and change state without reading the next input symbol or moving its read-only head. Besides this, there may be multiple options for possible next moves.

Formal Definitions : Formally, a PDA M is a 7-tuple .

A push down automata M is defined by $(Q,\Sigma,\acute{\Gamma},\delta,q_0,z_0.F)$ where,

Qis the finite set of states

Σ is the alphabet called input alphabet

Ґ is the stack alphabet , is a finite alphabet os stack symbols.

q_0 ε Q is the stack state/initial state

z_0 in Ґ is the particular stack symbol called start symbol.

F≤Q is the set of final states.

δ is the transition relation.

i.e, δ is a subset of $(Q \times \Sigma \cup \{e\} \times \acute{\Gamma}^*) \rightarrow (2^{Q \times T^*})$

Example Processing of 100001:

$(P, 100001, Z0) \vdash (q, 100001, Z0)$

$\vdash (q, 00001, 1Z0)$

$\vdash (q, 0001, 01Z0)$

$\vdash (q, 001, 001Z0)$

$\vdash (q, 01, 01Z0)$

$\vdash (q, 1, 1Z0)$

$\vdash (q, \epsilon, Z0)$

$\vdash (q, \epsilon)$

Hence string is accepted

Refer class notes for more examples

Note:

Regular language

A language can be described by DFA/NFA.

Context free language:

Cannot be described by DFA/NFA, since there is no memory.

Push down automata:

It has a memory, which can be used to count the number.

Push down automata can accept CFL. It is essentially an NFA with a stack

2.Comparison of NFA AND PDA

NFA

- (P,a,q) ε Δ means if machine M is in state P, then on reading "a" from input tape go to stack q.
- (P, ε,q) ε Δ means if machine M is in state P, goes to state q, without consuming input.

PDA

1. ((P,a,β),(q,y)) ⟶ if machine is in state P, the symbol read from input state is 'a', and β is on top of stack, goes to state q, and replace β by y on top of stack.
2. ((s,a,e),(s,a)) ⟶ if machine M is in state s reads 'a', remains in state s and push a onto stack.
3. ((s,c,e),(f,e)) ⟶ if read 'c' in state s and stack is empty, goes to final state f and nothing to push onto the stack.
4. ((s,e,e),(f,e)) ⟶ if in state s, goes to state f.
5. ((f,a,q),(f,φ)) ⟶ if read a in stack f, remain in state f and pop a from stack.
6. PDA's are non-deterministic.

Properties Of Pda/Characteristics Of Pda

A push down automata M is defined by $(Q,\Sigma,\acute{\Gamma},\delta,q_0,z_0.F)$ where,

Qis the finite set of states

Σ is the alphabet called input alphabet

Ґ is the stack alphabet , is a finite alphabet os stack symbols.

q_0 ε Q is the stack state/initial state

z_0 in Ґ is the particular stack symbol called start symbol.

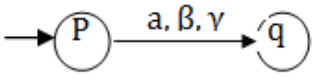
F≤Q is the set of final states.

δ is the transition relation.

i.e, δ is a subset of $(Q \times \Sigma \cup \{e\} \times \acute{\Gamma}^*)$ ⟶ $(2^{Q \times T^*})$

Transition Function

1. Let ((P,a,β),(q,γ)) ε δ
2. It means that we,
   a. Read a from tape.
   b. Pop the string β from stack.
   c. Move from stack P to state q.
   d. Push string γ onto stack.
3. We will draw it as

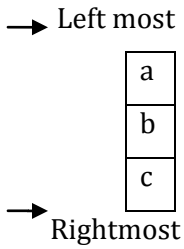


Pushing And Poping

When we push β, we must push the symbols of β as we read them right to left.

When we pop γ, we pop the symbols of γ as we read them from left to right.

i.e, if we push abc, then we can pop a, then pop b, then pop c. so e push like this



Thus, if we push the string abc and then pop it, we will get back abc.

Accepting Strings

1. After processing the string on the tape.

- The PDA is in either a favorable or an unfavorable state, and
- The stack is either empty / not empty.

2.The input string is accepted if

- The final state is favorable, and
- The stack is empty.

Acceptance By PDA

3. Acceptance by final state

Let M=(Q,Σ,Ґ,δ,q_0,z_0.F) be a PDA. Then, the language accepted by M is set of strings that,

L(M)=(w/(q_0,w, z_0) ⊢* (p, ε, γ) for some P ε F, γ ε Ґ*).

4. Acceptance by empty stack:

Let N(M) be the language accepted by M is given by,

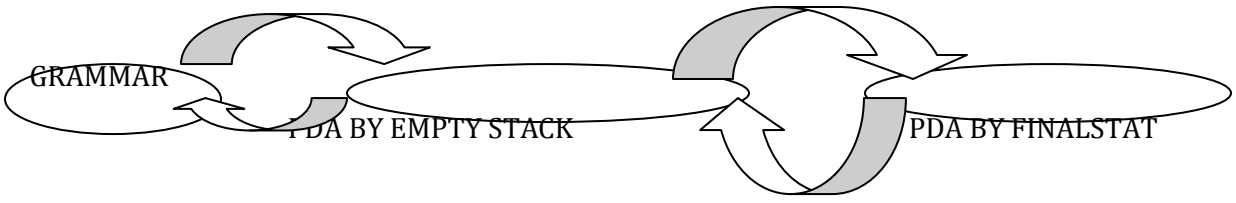
N(M)= (w/(q_0,w, z_0) ⊢* (p, ε, ε) for some P ε Q.

3. Explain PUSH DOWN AUTOMATA and CFL(Context Free Languages)

Equivalence of PDA and CFG

A language is generated by a CFG.

a) If and only if it is accepted by a PDA by empty stack.
b) If and only if it is accepted by a PDA by final state.



We already know how to go between null. Stack and final state.

From CFL's to PDA's

Let G=(V,T,P,S) be CFG , construct PDA P that accepts L(G) be empty stack as follows:

P={ (q), T, V ∪ T, δ,q,S) where δ is defined by

1. For each variable A,

   $\delta(q, \varepsilon, A)=\{(q,B) / A->_{\beta}$ is a production in G$\}$

2. For every terminal symbol a,

   $\delta(q, a, a)=(q, \varepsilon)$

Example:

Convert the grammar S->0S1/A, A->1A0/S/ ε into PDA that accepts same language by empty stack. Check whether 0101 belongs to N(P).

Sol:

P=({q},{0,1},{S,A,0,1}, δ, q, S) where δ is defined by

δ (q, ε, S)={(q,0S1),(q,A)}

δ (q, ε, A)={(q,1A0),(q,S),(q, δ (q, ε, A)={()}

δ (q, 0,0)=(q, ε)

δ (q, 1, 1)=(q, ε)

w=0101

(q,0101,S) = (q,0101,S1)

=(q,0101,0S1)

=(q,101,S1)

=(q,101,1A01)

=(q,01,A01)

=(q,01,01)

=(q,1,1)

=(q, ε, ε)

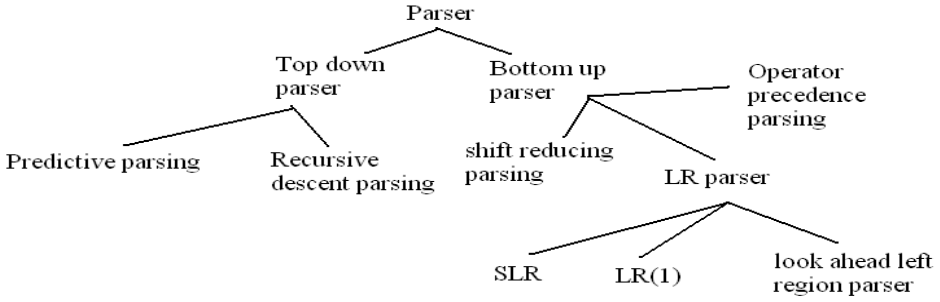
Therefore, 0101 ε N(P)

4. Explain Top down Parsing and Bottom up Parsing (UQ:NOV'07,MAY'10)

PARSING

Processor recognizes the input. It is a machine which is present in the automata m/c. Tokens are identified by the parsing. Tokens are nothing but identifiers or keywords. There are two types of parsing. They are

1. Bottom up
2. Top down (Recursive descent parsing)



BOTTOM UP PARSER

It is also called as shift reducing parsing. Implementation of SRP called operator parsing. A general format \ method of SRD is LR parsing which is used in automatic parser. Shift reduce parser consists for a given i/p string "reducing" a string ω to start symbol of a G. At each reduction a particular substring is matching right side of a production is replaced by symbol on the left of that production.

EXAMPLE PROBLEMS:

1) Consider the following grammar, implement bottom up parsing.

E→E+E/E * E /(E)/id

w = id1+id2*id3

1. Right most derivation
2. Left most derivation
3. Identify handles/production
4. Stack implementation

Right most derivation

E→E+E

➔E+E*E

➔E+E*id3

➔E+id2*id3

➔id1+id2+id3

Left most derivation

E→E*E

➔E+E*E

➔id1+E*E

➔id1+id2*E

➔id1+id2*id3

Identify production handles

E→E*E

E→E+E

E→id1|id2|id3

Stack implementation

Action contains 2 operations

1. Shift
2. Reduce

STACK	INPUT	ACTION
$	id1+id2+id3$	Shift id1 to stack
$id1	+id2*id3$	Reduce E→id1
$E	+id2*id3$	Shift
$E +	+id2*id3$	Shift
$E + id2	*id3$	Reduce E→id2
$E + E	*id3$	Reduce E→E+E
$E	*id3$	Shift *
$E *	id3$	Shift
$E * id3	$	Reduce E→id3
$E * E	$	Reduce E→E+E
$E	$	Accepted

TOP DOWN PARSING or RECURSION DESCENT PARSING

It has 5 steps. They are

1. Elimination of left recursion.
2. Leftmost derivation
3. Rightmost derivation
4. Identify handles
5. Stack implementation

Introduction of Top down parsing

It involves backtracking ie marking repeated scans of the input. It can be viewed as an attempt to find a LMD for an input string. Equivalently , it can be viewed as attempting to construct a parse tree. For the input starting from the root and creating the nodes of the parse tree in preorder.

1.Left recursion

A grammar a is said to be left recursion if it has a non-terminal A such that there is a derivation A➔ Aα for some α. A left- recursion grammar can cause a top-down parser to go into an infinite loop. This cycling will surely occur on an erroneous input string, it may also occur on legal inputs, depending on the order in which the alternatives for A are tried. Therefore to use top-down parsing, we must eliminate left recursion from the grammar.

2. Backtracking

If we make a sequence of erroneous expansion and subsequently discover a mismatch, we may have to undo the semantic effects of machine this erroneous expansions.

With parse tree already matched, the failure of next input symbol, B to match, would imply that the alternate C A d for s was wrong.

1. Elimination of left recursion

A→ Aα / β

Where β does not begin with an A, there we can eliminate the left recursion by replacing this pair of productions with

A → βA′

A → αA′/ε

Example:

A→Aα

A→Aαα

A→Aααα……

This left recursion can be avoided by above formula.

Problems:

1.) Construct top down parser for the given grammer

E→E+T|T

T→T*F|F

F→(E)|id

w= id +id*id

Soln:

1. Elimination of the left Recursion:

Consider the production

E→E+T|T

Apply the formula for removing left recursion

A→βA′

A′→αA′| ε

E→TE′

E′→+TE′|ε

Consider the next production

T→T*F|F

T→FT′

T′→*FT′|ε

F→(E)|id

The productions are

E→TE′

E′→+TE′|ε

T→FT′

T′→*FT′|ε

F→(E)|id

LMD

E→TE′

➔FT′E′

➔idT′E′

➔idεE′

➔idE′

➔id+TE′

➔id+FT′E′

➔id+idT′E′

➔id+id*FT′E′

➔id+id*idT′E′

➔id+id*id

RMD

E→TE′

➔T+TE′

➔T+FT′E′

➔T+F*FT′E′

➔T+F*FT′E′

➔T+F*Fεε

➔T+id*id

➔FT′+id*id

➔id+id*id

Identify Handles

T→FT′

E′→+TE′|ε

T′→*FT′|ε

F→id

Stack Implementation

STACK	INPUT	ACTION
$E	id+id*id$	E→TE′
$E′T	id+id*id$	T→FT′
$E′T′F	id+id*id$	F→id
$E′T′id	id+id*id$	T′→ε
$E′	+id*id$	E′→+TE′

$E′T+	+id*id$	T→FT′
$E′T′F	id*id$	F→id
$E′T′id	id*id$	T′→*FT′
$E′T′F*	*id$	F→id
$E′T′id	id$	T′→ε, E′→ε
$	$	Accepted

5.**Explain the closure properties of CFL or State the properties of CFL (UQ:DEC'09,NOV'07,DEC'08,MAY'15)**

CLOSURE PROPEERTIES OF CFL:-

1) Regular verses CFL

Theorem

Statement

Every regular language is CFL.

Proof:

Let L be a regular.

Given a DFA for L, add a stack , but don't use the stack. Change each DFA transition (P,a,q) to a deterministic productions automata.

δ (p,aq) ={ (q,x)}

The result is deterministic P A whose language is L. Therefore L is a context free language.

2) Closure under union

Statement

L1 and L2 be CFL. Then L1 U L2 is also a CFl.

Proof

Let L1 have grammar (V1,T1,P1,S1) and L2 have grammar (V2,T2,P2,S2).

L1 U L2 has grammar V3= V1 U V2 U S3

T3= T1 U T2

S3= new start symbol.

P3= P1 U P2 U {S3→S1/S2}

Therefore L1 U L2 is CFL.

3) Closure under concatenation

Statement

Let L1 and L2 be CFl. Then L1 and L2 is also CFL.

Proof:

Let L1 have grammar (V1,T1,P1,S1)

Let L2 have grammar (V2,T2,P2,S2)

Then L1L2 has grammar (V,T,P,S)

Where V= V1 U V2 {S}

T= T1 U T2

P= P1 U P2 U { S→S1/S2 }

S= start symbol

Therefore L1 L2 is a CFL.

4) Closure under kleen star

Statement

Let L be a CFL. Then L * is also a CFL.

Proof

Let L have grammar (V1,T1,P1,S1)

Then L* has grammar (V,T,P,S)

Where V= V1 U {S}

T=T1

P= P1 U { S→e , S→S1}

S= start symbol

Therefore L is a CFL.

5) Intersection of CFL and regular expression

Statement

Intersection of a CFL under RE is a CFl.

Proof

Given

Let L1=L(M1) for some PDA.

M1= (Q1, ∑1,Γ1,δ1,S1,F1)

L2= L(M2) for some DFA

M2=(Q2, ∑2,Γ2,δ2,S2,F2)

Need to show

L1 ∩ L2 = L (M) for some PDA , M where M=(Q, ∑,Γ,δ,S,F)

Idea

Construct a PDA M that operates same way as M1 except that it also keeps track of the change in the M2 caused by reading the same input.

Construction

Q = Q1 x Q2 , ∑ = ∑1 U ∑2 , Γ= F1,

S = {S1, S2}, F = F1 x F2 for each transition { (q1,a,β) , (p,γ) }Є δ for each state q2 Є Q2 add to δ transition (((q1,q2),a, β), ((p1,δ,(q2,a)),γ)).

Pumping lemma for CFL and its applications:

- The pumping lemma for CFL will allow us to show that some languages are context free.
- If a CFL, contains a word 'ω' with sufficiently long derivations ($S \overset{*}{\rightarrow} \omega$). Then some non-terminal

must appear more than once.

(ie) we have S $\overset{*}{\rightarrow}$ uAz $\overset{*}{\rightarrow}$ uvAyz $\overset{*}{\rightarrow}$ uvxyz.

Thus A $\overset{*}{\rightarrow}$ vAy and A $\overset{*}{\rightarrow}$ x. We may repeat the derivation A $\overset{*}{\rightarrow}$ vAy has many times as we like. (0 or more time) producing uv^nxy^nz for any $n \geq 0$.

6.Write short notes on Decision algorithm (UQ:DEC'08)

Decision algorithm for cfl is mainly used to check whether the given language is finite,empty or non finite(infinite).

TO PROVE:

To check whether the language is finite,infinite or empty.

ASSUMPTIONS:

1.the grammar should be in Chomsky normal form(cnf) without ε production.

2.if 's' is the start symbol and 'r' is the rank of the symbol, then the string length will be greater then or equal to 2^r.

If 'A' is the reverse vertex associated with 's' is the root node ,then the string length will not be greater than 2^r-1.

If 'B' is the reverse vertex associated with 's' is the root node,then the string length will not be greater

than 2^r-2.

BASIS PART: (r=0)

Consider a production A->a,it is a directed acyclic graph denoted by (A) .by the assumption a non terminal with rank 'r' cannot generate a string of length greater than 2^r.

Here, the rank of A is '0'.i.e(r=0)

By the condition l<=2^r,substitute 'r' value,

We get l<=2^0

L<=1

Hence it is proved by considering the form A->a,we may derive only a string of length '1',which is finite.

INDUCTION PART:

If we use a production of the form A->a ,we may derive only a string of length 1.if we begin with A->BC, then as B and C are of rank r-1 or less,by the inductive hypothesis,they derive only strings of length 2^r-1 or less.thus BC cannot derive a string of length greater than 2^r.

Since s is of finite rank r0,and in fact,is of rank no greater than the number of variables,s derives strings oflength no greater than the number of variables,s derives strings of length no greater than 2 r^0.thus the language is finite.

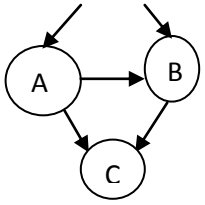
Consider an example,

S->AB

A->BC

B->CC

C->a



r (s)=3

r(A)=2

r(B)=1

r(c)=0

rank of starting symbol ‘s’ (r(s)=3).

Rank ‘A’ ,r(A) depends on root node ‘s’,it should be less than’s’ and rank of’b’,r(B) depends on both S and A ,it should be less than both S and A .

From, the figure check the condition l<=2^r

r (s)=3, l<=2^3

l<=8

r(A)=2, l<=2^2

l<=4

r(B) =1, l<=2^1

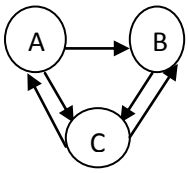
l<=2

r(c) =0, l<=2^0

l<=1

hence the finite length for each string is derived ..hence it proved.

Now,consider the directed cyclic graph,



For the figure, the rank cannot be specified. hence for the cyclic graph the language is infinite.

For cyclic graph the language is finite. hence the statement is proved.

7.Given G(V, T, P, S) where V= {S, a, b, c} T= {a, b, c} P= {S → aSa, S→ bSb, S → c}. Construct a PDA and run for string abcba.

Solution

Let M = (Q, Σ, Γ, δ, q0, Z0, F) be a PDA.

Q = {p, q} p = start state q= final state

Σ = {a, b, c} terminals of grammar

Γ = {S, a, b, c} U {Z0}

F = {q}

δ is

R1 : δ(P, ϵ, Z0) = {(q, SZ0)} start symbol put on the stack

R2 : δ(q, ϵ, S) = {(q, aSa)} prod 1
R3 : δ(q, ϵ, S) = {(q, bSb)} prod 2

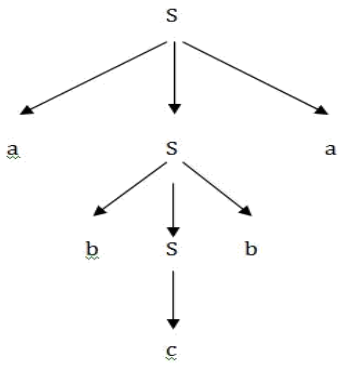
R4 : δ(q, ϵ, S) = {(q, c)} prod 3

R5 : δ(q, a, a) = {(q, ϵ)}

R6 : δ(q, b, b) = {(q, ϵ)}

R7 : δ(q, c, c) = {(q, ϵ)}

R8 : δ(q, ϵ, Z0) = {(q, ϵ)}



Processing of abcba:

(P, abcba, Z0) ⊢ (q, abcba, SZ0) – R1

⊢ (q, abcba, aSaZ0) – R2

⊢ (q, bcba, SaZ0) – R5

⊢ (q, bcba, bSbaZ0) – R3

⊢ (q, cba, SbaZ0) – R6

⊢ (q, cba, cbaZ0) – R4

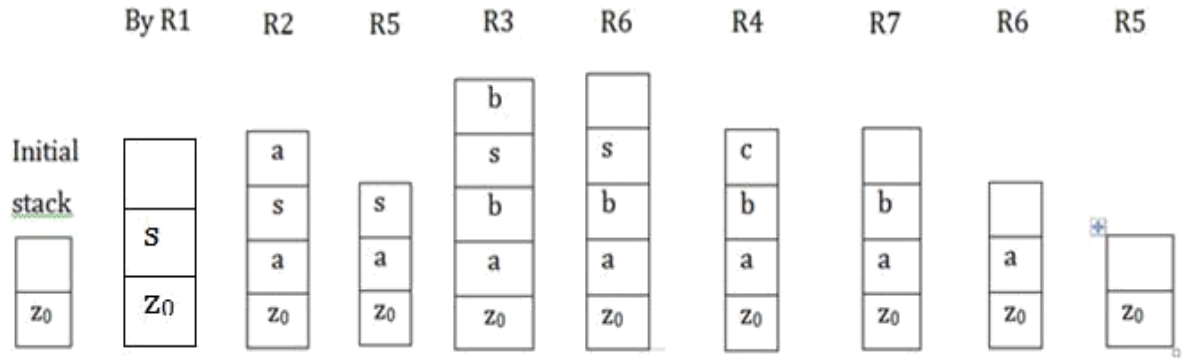
⊢ (q, ba, baZ0) – R7

⊢ (q, a, aZ0) – R6

⊢ (q, ϵ, Z0) – R5

⊢ (q, ϵ) – R8

Hence string is accepted. The stack variable is as follows.



8.Construct PDA that accepts the language generated by grammar with productions

S →aSbb/a (UQ:MAY'08)

Solution:

Refer Class notes

9.IF L is N(M) for some PDA M then there is a CFL prove OR If L is context free language ,then prove that there exists a PDA M such that L=N(M) (UQ:NOV'14)

THEOREM

For any context free language L, there exists an PDA M such that L = L(M).

Proof

Let G= (V, P, T, S) be a grammar. There exists GNF, we can construct PDA which simulates leftmost derivations in this G'r.

M = (Q, Σ, Γ, δ, q0, Z, F)

Q = {q0, q1, qf} set of states

Σ = terminals of grammar G.

Γ = V U {Z} where V is variables in G'r.

F = {q2} final state.

The transition function will include δ(q1, ϵ, Z) = {(q1, S)} so that after the first move of M, the stack contains the start symbol S of the derivation. In addition, the set of transition rules is such that

δ(q1, ϵ, A) = {(q, α)} for each A → α in P.

δ(q, a, a) = {(q, ϵ)} for each a ϵ Σ

E.g. consider the grammar G= (V, T, P, S) where

S → aA

A → aABC / bB/ a

B →b

C →c and find the PDA and process aaabc.

10. Design a PDA that accepts the language {$a^n b^n$ | n>=0 }

Here is a PDA that accepts the language like {$a^n b^n$ | n>=0 }

M= (Q,Σ,Ґ,δ,q_0,z_0.F)

Q = {q_1,q_2,q_3,q_4}

Σ = {a,b}

Ґ = {a,b,z}

F= {q_1,q_4}

Solution:

δ (q_1, a , z) = {(q_2 , az)}

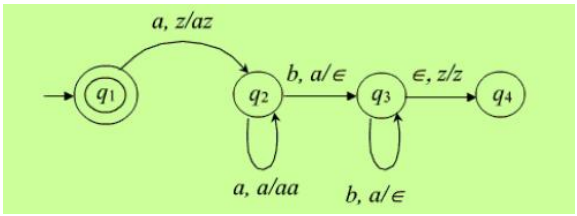
δ (q_2, a , a) = {(q_2 , aa)}

δ (q_2, b , a) = {(q_3 , ε)}

δ (q_3, b , a) = {(q_3 , ε)}

δ (q_3, ε , z) = {(q_4 , z) }

The PDA can also be described by the adjacent transition diagram.



11. Explain Pumping Lemma for CFG (or) Application of Pumping Leema (NOV'15)

A "Pumping Lemma" is a theorem used to show that, if certain strings belong to a language, then certain other strings must also belong to the language. Let us discuss a Pumping Lemma for CFL. We will show that , if *L* is a context-free language, then strings of *L* that are at least '*m*' symbols long can be "pumped" to produce additional strings in *L*. The value of '*m*' depends on the particular language. Let *L* be an infinite context-free language. Then there is some positive integer '*m*' such that, if *S* is a string of *L* of Length at least '*m*', then

(i) S = uvwxy (for some u, v, w, x, y)

(ii) | vwx| <= m

(iii) | vx| >=1

(iv) $uv_iwx_iy \varepsilon L$.

for all non-negative values of *i*.

It should be understood that

(i) If *S* is sufficiently long string, then there are two substrings, *v* and *x*, somewhere in *S*. There is stuff (*u*) before *v*, stuff (*w*) between *v* and *x*, and stuff (*y*), after *x*.

(ii) The stuff between *v* and *x* won't be too long, because | *vwx* | can't be larger than *m*.

(iii) Substrings *v* and *x* won't both be empty, though either one could be.

(iv) If we duplicate substring *v*, some number (i) of times, and duplicate *x* the same number of times, the resultant string will also be in *L*.

Definitions

A variable is useful if it occurs in the derivation of some string. This requires that

(a) the variable occurs in some sentential form (you can get to the variable if you start from *S*), and

(b) a string of terminals can be derived from the sentential form (the variable is not a "dead end"). A variable is "recursive" if it can generate a string containing itself. For example, variable *A* is recursive if

$$S \overset{*}{\Rightarrow} uAy$$

for some values of u and y.

A recursive variable A can be either

(i) "Directly Recursive", i.e., there is a production

$$A \to x_1 A x_2$$

for some strings $x_1, x_2 \in (T \cup V)^*$, or

(ii) "Indirectly Recursive", i.e., there are variables x_i and productions

$$A \to X_1 \ldots$$
$$X_1 \to \ldots X_2 \ldots$$
$$X_2 \to \ldots X_3 \ldots$$
$$X_N \to \ldots A \ldots$$

Proof of Pumping Lemma

(a) Suppose we have a CFL given by *L*. Then there is some context-free Grammar *G* that generates *L*. Suppose

(i) *L* is infinite, hence there is no proper upper bound on the length of strings belonging to *L*.

(ii) *L* does not contain l.

(iii) *G* has no productions or l-productions.

There are only a finite number of variables in a grammar and the productions for each variable have finite lengths. The only way that a grammar can generate arbitrarily long strings is if one or more variables is both useful and recursive. Suppose no variable is recursive. Since the start symbol is non recursive, it must be defined only in terms of terminals and other variables. Then since those variables are non recursive, they have to be defined in terms of terminals and still other variables and so on.

After a while we run out of "other variables" while the generated string is still finite. Therefore there is an upper bond on the length of the string which can be generated from the start symbol. This contradicts our statement that the language is finite. Hence, our assumption that no variable is recursive must be incorrect. **(b)** Let us consider a string *X* belonging to *L*. If *X* is sufficiently long, then the derivation of *X* must have involved recursive use of some variable *A*. Since *A* was used in the derivation, the derivation should have started as

$$S \overset{*}{\Rightarrow} uAy$$

For some value of u and y .Since A was used recursively the derivation must have continued as

$$S \overset{*}{\Rightarrow} uAy \overset{*}{\Rightarrow} uvAxy$$

Finally the derivation must have eliminated all variables to reach a string X in the language

$$S \overset{*}{\Rightarrow} uAy \overset{*}{\Rightarrow} uvAxy \overset{*}{\Rightarrow} uvwxy = x$$

This shows the derivation steps

$$A \overset{*}{\Rightarrow} vAx$$

And

$$A \overset{*}{\Rightarrow} w$$

Are possible .Hence the derivation

$$A \overset{*}{\Rightarrow} vwx$$

must also be possible.

It should be noted here that the above does not imply that a was used recursively only once. The * of $\overset{*}{\Rightarrow}$ could cover many uses of A, as well as other recursive variables.

There has to be some "last" recursive step. Consider the longest strings that can be derived for v, w and x without the use of recursion. Then there is a number 'm' such that $|vwx| < m$.

Since the grammar does not contain any λ-productions or unit productions, every derivation step either introduces a terminal or increases the length of the sentential form. Since $A \overset{*}{\Rightarrow} vAx$, it follows that $|vx| > 0$.

Finally, since $uvAxy$ occurs in the derivation, and $A \overset{*}{\Rightarrow} vAx$ and $A \overset{*}{\Rightarrow} w$ are both possible, it follows that $uv^i wx^i y$ also belongs to L.

This completes the proof of all parts of Lemma.

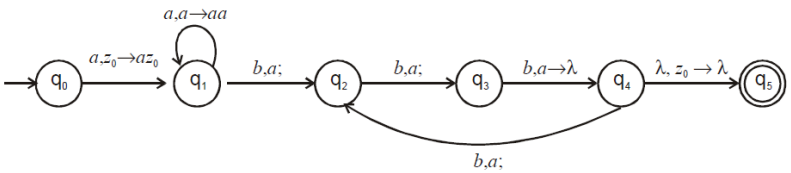
12. Construct a PDA accepting $L = \{a^n b^{3n} : n \geq 1\}$ by empty store.

Solution :

PDA should accept input strings with symbols '*a*' and 'b' where number of *b*'s is thrice that of *a*'s.

$\delta(q_0, a, z_0) = (q_1, a\, z_0)$
$\delta(q_1, a, a) = (q_1, aa)$
$\delta(q_1, b, a) = (q_2, a)$
$\delta(q_2, b, a) = (q_3, a)$
$\delta(q_3, b, a) = (q_4, \lambda)$

d(q_4, b, a) = (q_2, a)
d(q_4, λ, z_0) = (q_5, λ)



Consider the input string *aabbbbbb*
d(q_0, a, z_0) = (q_1, az_0)
d(q_1, a, a) = (q_1, aa)
d(q_1, b, a) = (q_2, a)
d(q_2, b, a) = (q_3, a)
d(q_3, b, a) = (q_4, λ)
d(q_4, b, a) = (q_2, a)
d(q_2, b, a) = (q_3, a)
d(q_3, b, a) = (q_4, λ)
d(q_4, λ, z_0) = (q_5, l) it halts in empty store and hence string is accepted.

10. Design a PDA to accept palindromes over {a, b}. **(APR 2015)**

Solution:

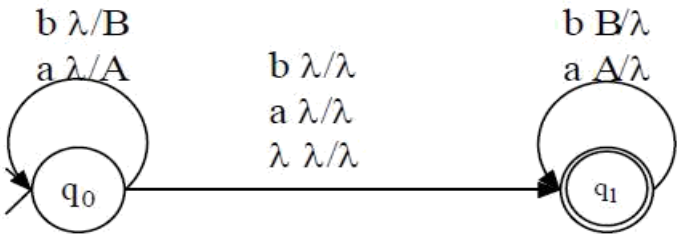
Construct the PDA:

M: Q= {q0, q1}

Σ = {a, b} Γ

= {a,} F=

{q1}



δ consists of the following transitions

δ (q_0, a , λ) = {[q_0 ,A], [q_1, λ]}

δ (q_0, b , λ) = {[q_0 ,B], [q_1, λ]}

δ (q_0, λ , λ) = {[q_1, λ]}

δ (q_1, a , A) = {[q_1, λ]}

δ (q_1, b , B) = {[q_1, λ]}

Pondicherry University Questions

2 Marks

1. Mention any two applications of pumping lemma **(UQ:MAY'12) (Qn.No.33)**
2. Define Chomsky normal form. **(UQ:MAY'13) (Qn.No.34)**
3. What is multiple tracks Turing machine? Or Give the significance of multiple tracks. **(UQ:APR/MAY'13) (Qn.No.35)**
4. State the properties that are not closed under CFL **(UQ:NOV'12) (Qn.No.36)**
5. State the difference between top down parsing and bottom up parsing **(UQ:NOV'12) (Qn.No.37)**
6. What is push down automata? **(UQ:NOV'14)N 1 (Qn.No.1)**
7. What is recursively enumerable languages **(UQ:NOV'14) (Qn.No.38)**
8. What are the closure properties of CFL? **(NOV'15) (Qn.No.30)**

11 Marks

1. Discuss about PDA **(UQ:DEC'09,MAY'10) (Qn.No.1)**
2. Explain Top down Parsing and Bottom up Parsing **(UQ:NOV'07,MAY'10) (Qn.No.4)**
3. Explain the closure properties of CFL**(UQ:DEC'09,NOV'07,DEC'08,MAY'15, APR'16) (Qn.No.5)**
4. Write short notes on Decision algorithm (**UQ:DEC'08 APR,16) (Qn.No.6)**
5. Construct PDA that accepts the language generated by grammar with productions
    a. S →aSbb/a **(UQ:MAY'08) (Qn.No.8)**
6. IF L is N(M) for some PDA M then there is a CFL prove OR If L is context free language ,then prove that there exists a PDA M such that L=N(M) **(UQ:NOV'14) (Qn.No.9)**
7. Explain Pumping Lemma for CFG **(NOV'15) (Qn.No.11)**

UNIT IV

Turing machines: Turing machines (TM) – computable languages and functions –Turing Machine constructions – Storage in finite control – variations of TMs – Recursive and Recursive. Enumerable languages, Recursive Function, Partial and Total Recursive Function, Primitive Recursive Function.

2 Marks

1.What is a turning machine? (UQ:APR/MAY'12)

Turing machine is used for computing mathematical functions. It can be divided into n-no of cells (or) squares. TM is similar to finite automaton but with an unlimited and unrestricted memory.

Each cell contain finite no of symbols along with blank spaces.

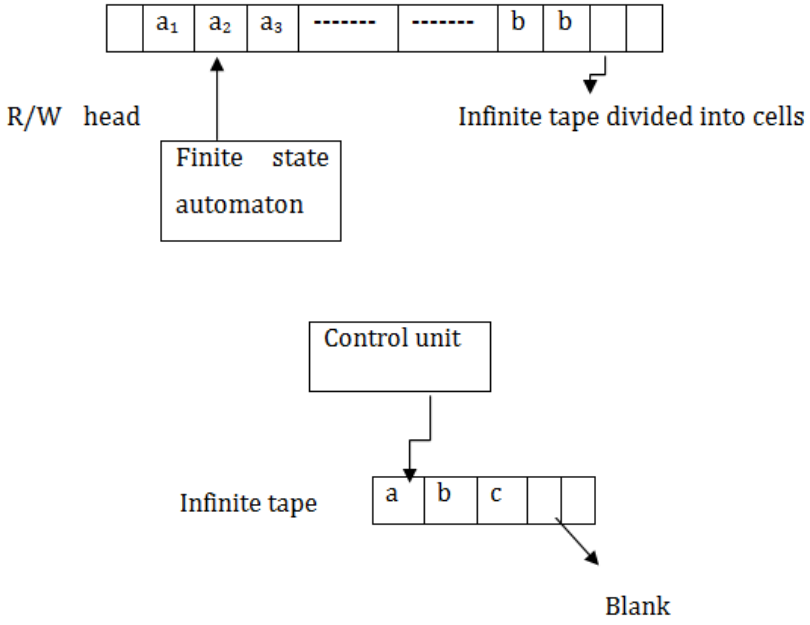
2.What is the operation of the turning machine?

The operation turing machine are follows,

The turing machine(TM) reads the start symbol i.e, is under the TM tape head. This symbol is referred as current symbol.The TM is similar to finite automaton but with an unlimited and unrestricted memory.

3.Draw the diagram of the model of turning machine

Model of TM



4.What is the function of the turning machine?

The turing machine can be thought of as a finite state automaton connected to a R/W head. It has a infinite tape which is divided into no of cells. Each cell stores one symbol. The input to and output from finite state automata or control unit are affected by R/W head which can examine one cell at a time.

5.What does the first move performs?

1) A new symbol to be written on the tape in the cell under the R/W head.
2) A motion of the R/W head along the tape, either the head moves one cell left (L), or one cell to right (R).
3) The next state if automaton.
4) Whether to halt/not.

6.What is the general form of the turning machine?

M = (Q, Σ, Γ, δ, q0, B, F) where

Q = finite set of states.

Γ = finite set of tape symbols.

Σ = not including B, set of input symbols.

B = symbols of Γ, is the blank.

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

q0 belongs to Q $\longrightarrow$ initial state.

F belongs to Q $\longrightarrow$ set of final state.

7.What are the representations of the turning machine?

1) Instantaneous descriptions.
2) Transition table.
3) Transition diagram.

8.What is the instantaneous description OF A TM (MAY'15)

An Instantaneous Description (ID)of the turing machine M is denoted by $\alpha_1 q\ \alpha_2$.Henceqε Q is the current state of M. $\alpha_1\alpha_2$ is the string in Γ^* that is the contents of the tape upto the rightmost non blank symbol or symbol to the left of the head, whichever is rightmost.

We define a move of M as follows. Let $x_1x_2\ldots\ldots x_{i-1}qx_i\ldots\ldots x_n$ be an ID. Let $(q_i,x_i)=(p,y,L)$.If i-1=n, then take x_i as B(blank).If i=1, then there is no next ID, as the tape head is not allowed to fall off the left end of the tape. If i>1, then we write

$X_1x_2\ldots\ldots x_{i-1}qx_i\ldots.x_n \vdash x_1x_2\ldots\ldots x_{i-2}px_{i-1}yx_{i+1}\ldots.x_n$ ----------→ 1

If $\delta(q,x_i)=(p,y,R)$, thgen change of ID is

$X_1x_2\ldots\ldots x_{i-1}qx_i\ldots.x_n \vdash x_1x_2\ldots\ldots x_{i-1}ypx_{i+1}..x_n$ ----------→ 2

If i-1=n, the string xi……xn is empty and the right side of 2 is longer that left side.

(Or)

The ID of a TM M is denoted as α1q α2 . Here q is the current state of M is in Q;

α1 α2 is the string in Γ * that is the contents of the tape up to the rightmost nonblank symbol or the symbol to the left of the head, whichever is the rightmost.

9.What are the techniques of tuning machine construction? (NOV'15)

- Storage in finite control.
- Multiple tracks.
- Checking off symbols.
- Shifting over
- Subroutines.

10.Construct the TM to compute the concatenation function?

Solution

Idea

The symbol 1 seperates n1 and n2. The concatenation is implemented by removing the separator (removal is done by replacing separator input) and last 0 of n2 will be replaced by blank space B.

Concept

From the start state it moves right until it finds the sytmbol 1.

1 is replaced by 0 and moves right until blank is reached. Then moves towards left cell symbols and it is replaced by blank symbol to get the derived symbol.

The input is $0^{n1},0^{n2}$. The output should be 0^{n1n2}

Let M= (Q, Σ, Γ, δ, q0, B, F) where

Q={ q0,q1,q2,q3}

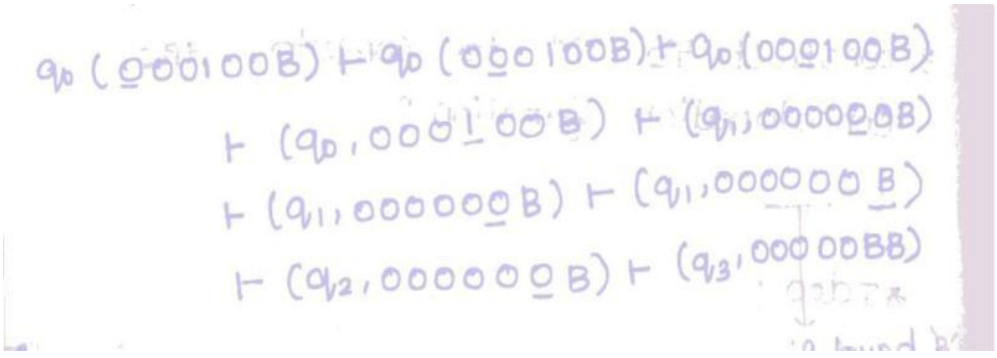
Σ ={0,1}

F={q3}

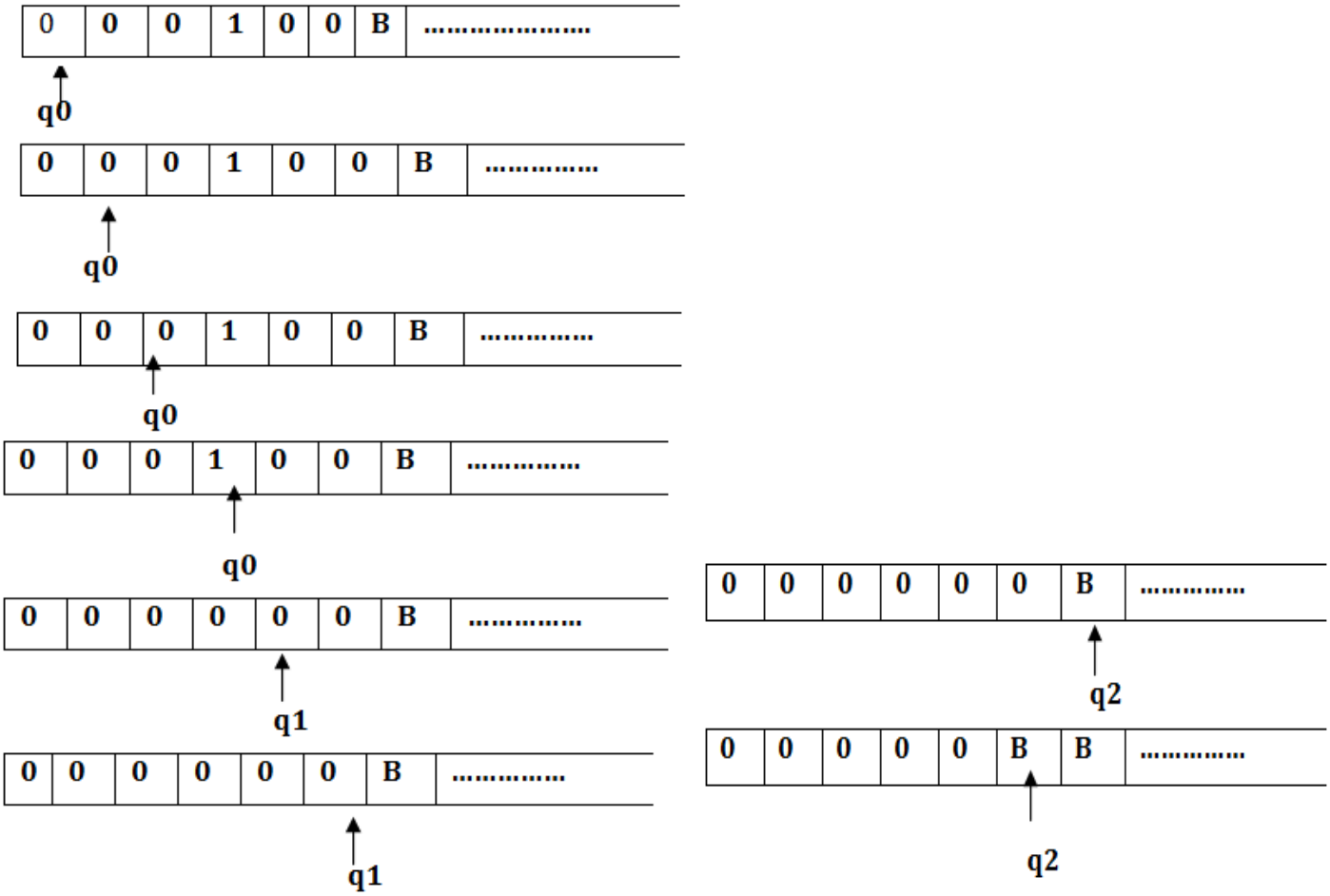
Initial State={q0}

Γ={0,1,B}

Let n1=3,n2=2



The machine falls at a acceptance state q3 leaving the output on the tape.



11. Construct a TM that performs addition operators f(n,m)=n+m

Solution

The input to the function f(n,m) are encoded as 0^n10^m. The output will be 0^{n+m} place on the tape starting from the first '0' in the 0^n, the tape head moves till it finds a separator '1'. Replace it by '0' move right to find the blank symbol. Then move left to one symbol replaces 's' by 0 in that cell by a blank symbol.

Let M= (Q, Σ, Γ, δ, q0, B, F) where

Q={ q0,q1,q2,q3}

Σ ={0,1}

F={q3}

Initial State={q0}

Γ={0,1,B}

12. What is undeciability?

A problem whose language is recursive is said to be decidable. Otherwise the problem is undecidable i.e, a problem is undecidable if there is no algorithm that takes as input an instance of the problem and determines whether the answer to that instance is yes/no.

13.Show that The language l is defined as {"M","W": M is a DFSM that accepts w } is recursive.

The TM M_D decides L.

On input "M" "W", where M is a DFSM.

1. M_D emulates M on input w.
2. If the emulation ends in a final state of M, them M_D halts in state y. if it ends in a non-final state of M, then M_D halts in state n.

One corresponding way to encode M is to adopt the TM encoding for FSMs and encode Q,Σ,δ,S and F. Before starting the emulation, M_D first checks whether the input string is a legal encoding of a DFSM and DFSM's input string.

14. Show that the The language L defined as {"M";M is a DFSM and L(M) = φ} is recursive.

The following Tm M_φ decides L: on input "M", where M is a DFSM,

1. M_φ marks the start state of M.
2. Repeat until no new state are marked.
   If (P,σ,q) ε δ and p is already marke, then mark q.
3. If no final state is marked, M_φ halts in y; otherwise it halts in n.

15.Show that The language L defined as {"M_1" U "M_2": DFSM M_1, M_2 and $L(M_1) \leq L(M_2)$ is recursive.

The following TM M_E decides L: on input "M_1" and "M_2", where M_1 and M_2 are DFSMs.

1. Construct DFSM M for the language $L(M_1) \cap L(M_2)$.
2. Apply M_φ to the input string "M".
3. If M_φ halts in y, M_E halts in y.
4. If M_φ halts in n, M_E halts in n.

16. Which are called as Turing Decidable and Turing acceptable languages?

The Recursive and Recursive Enumerable languages are called as Turing Decidable and Turing acceptable languages respectively

17. What is the general form of recursive language?

Let M=(Q,∑,δ ,s,H) ba TM such that:

H: {y,n}, where y means yes and n means no.

18. What is decidable problem?

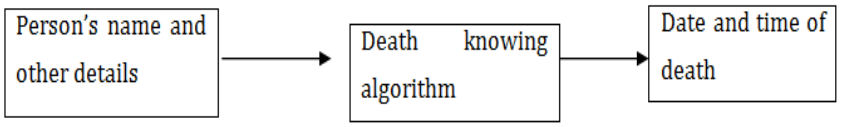
1) Decidable problem:

   Given two DFSM's M1 and M2 is L(M1)≤L(M2)?

   That is an algorithm is exist for the problem to decide the answer as yes/no.

19. What is undecidable problem?

Halting problem of turing machine. That is, given any TM M and any input string w, does M halt on w? Undecidabality is the situation where we could not develop an algorithm which will correctly give a solution to a given problem.



Undecidability of death

20. What is off-line Turing Machine? What is a multidimensional TM? (UQ:APR/MAY'12,MAY'15)

off-line Turing Machine

An Off-line Turing Machine is a multitape TM whose input tape is read only. The Turing Machine is not allowed to move the input tape head off the region between left and right end markers.

multidimensional TM

The device has a finite control , but the tape consists of a k dimensional array of cells infinite in all 2k directions, for some fixed k. Depending on the state and symbol scanned , the device changes state , prints a new symbol and moves its tape head in one of the 2k directions, either positively or negatively ,along one of the k axes.

21. **What is non deterministic TM (UQ:APR'13)**

A **non-deterministic Turing machine** (**NTM**), by contrast, may have a set of rules that prescribes more than one action for a given situation. For example, a non-deterministic Turing machine may have both "If you are in state 2 and you see an 'A', change it to a 'B' and move left" and "If you are in state 2 and you see an 'A', change it to a 'C' and move right" in its rule set.

22.What are the required fields of an instantaneous description or configuration of a TM. Or Define the basic guidelines for designing a TM (UQ:NOV'11)

It requires

- The state of the TM
- The contents of the tape.
- The position of the tape head on the tape.

23.State the powers of single tape and multitape TM (UQ:NOV'12)

The following models are single tape Turing machines but restricted with (i) restricted tape symbols { mark, blank }, and/or (ii) sequential, computer-like instructions, and/or (iii) machine-actions fully atomized.

Multi-tape machines are similar to single-tape machines, but there is some constant *k* number of independent tapes.

24. Define partial recursive function and total recursive function. (UQ:NOV'12)

Total recursive

If f(i1,i2,…….ik) is defined for all i1,…..ik then we say f is a total recursive function. They are similar to recursive languages as they are computed by TM that always halt.

Partial recursive

A function f(i1,…ik) computed by a Turing machine is called a partial recursive function. They are similar to r.e languages as they are computed by TM that may or may not halt on a given input.

25. What is off-line Turing Machine? (UQ:NOV'14)

An Off-line Turing Machine is a multitape TM whose input tape is read only. The Turing Machine is not allowed to move the input tape head off the region between left and right end markers.

26. **Differentiate recursive and recursively enumerable languages.**

Recursive languages	Recursively enumerable languages
A language is said to be recursive if and only if there exists a membership algorithm for it.	1A language is said to be r.e if there exists a TM that accepts it.
A language L is recursive iff there is a TM that decides L. (Turing decidable languages). TMs that decide languages are algorithms.	L is recursively enumerable iff there is a TM that semi-decides L. (Turing acceptable languages). TMs that semi-decides languages are not algorithms.

27. **What are UTMs or Universal Turing machines?**

Universal TMs are TMs that can be programmed to solve any problem, that can be solved by any Turing machine. A specific Universal Turing machine U is:

Input to U: The encoding "M " of a Tm M and encoding "w" of a string w. Behavior : U halts on input "M" "w" if and only if M halts on input w.

28.When a recursively enumerable language is said to be recursive ? Is it true that the language accepted by a non-deterministic Turing machine is different from recursively enumerable language? (APR'15)

A language L is recursively enumerable if there is a TM that accepts L and recursive if there is a TM that recognizes L. Thus r.e language is Turing acceptable and recursive language is Turing decidable languages.

No , the language accepted by non-deterministic Turing machine is same as recursively enumerable language.

29.When we say a problem is decidable? Give an example of undecidable problem?

A problem whose language is recursive is said to be decidable. Otherwise the problem is said to be undecidable. Decidable problems have an algorithm that takes as input an instance of the problem and determines whether the answer to that instance is "yes" or "no". (eg) of undecidable problems are (1)Halting problem of the TM.

30. Differentiate PDA and TM.

PDA	TM
PDA uses a stack for storage.	TM uses a tape that is infinite .
The language accepted by PDA is CFL.	Tm recognizes recursively enumerable languages.

31. What is a 2-way infinite tape TM?

In 2-way infinite tape TM, the tape is infinite in both directions. The leftmost square is not distinguished. Any computation that can be done by 2-way infinite tape can also be done by standard TM.

32. What is the storage in FC?

The finite control(FC) stores a limited amount of information. The state of the Finite control represents the state and the second element represent a symbol scanned.

33. What are the possibilities of a TM when processing an input string?

TM can accept the string by entering accepting state. It can reject the string by entering non-accepting state. It can enter an infinite loop so that it never halts.

34. What are the various representation of TM?

We can describe TM using: Instantaneous description. Transition table. Transition diagram.

35. What is the basic difference between 2-way FA and TM?

Turing machine can change symbols on its tape , whereas the FA cannot change symbols on tape. Also TM has a tape head that moves both left and right side ,whereas the FA doesn't have such a tape head.

36. What are the applications of TM?

TM can be used as:

Recognizers of languages.

Computers of functions on non negative integers. Generating devices.

37. What are the special features of TM?

In one move ,TM depending upon the symbol scanned by the tape head and state of the finite control:

1. Changes state.
2. Prints a symbol on the tape cell scanned, replacing what was written there. Moves the R/w head left or right one cell.

38. What is a recursively enumerable language?

The languages that is accepted by TM is said to be recursively enumerable (r. e) languages. Enumerable means that the strings in the language can be enumerated by the TM. The class of r. e languages include CFL's.

39. Define nondeterministic TM?

• Arbitrarily chooses move when more than one possibility exists

• Accepts if there is at least one computation that terminates in an accepting state

40. What are the difference between finite automata and Turing Machines?

Turing machine can change symbols on its tape, whereas the FA cannot change symbols on tape. Also TM has a tape head that moves both left and right side, whereas the FA doesn't have such a tape head.

41. What is Primitive recursive function? (APR 2016)

The primitive recursive functions are the number- theoretic functions, which are functions, which are functions from the natural numbers {0, 1, 2…}. The basic primitive recursive functions are

- ✓ Constant function
- ✓ Successor function
- ✓ Projection function

43. Define recursive language? (NOV 2015)

The Turing machine is a finite sequence of symbols as input, accepts it if belongs to the language and rejects it otherwise. Recursive languages are also called decidable.

A recursive language is a formal language for which there exists a Turing machine with any finite input string, halts and accepts if the string is in the language and halts and rejects otherwise. The Turing machine always halts; it is known as decider and is said to decide the recursive language.

44. List the properties of recursive and recursively enumerable languages?

1) The complement of a recursive language is recursive.
2) The union of two recursive languages is recursive. The union of two recursively enumerable language languages is recursively enumerable.
3) If a language L and its complement L are both recursively enumerable, the L (and hence L) is recursive.

45. What is Church's Hypothesis?

The notion of computable function can be identified with the class of partial recursive functions is known as Church-hypothesis or Church-Turing thesis. The Turing machine is equivalent in computing power to the digital computer.

46. What is multi-stack machines?

A deterministic two-stack machine is a deterministic Turing machine with a read-only input and two storage tapes. If a head moves left on either tape, a blank is printed on that tape.

47. What is counter machines?

Counter machines, which are off-line Turing machines whose storage tapes are semi-infinite, and whose tape alphabets contain only two symbols, Z and B (blank).

11 Marks

1. Explain in detail about Turing Machine (TM)? (6 MARKS)

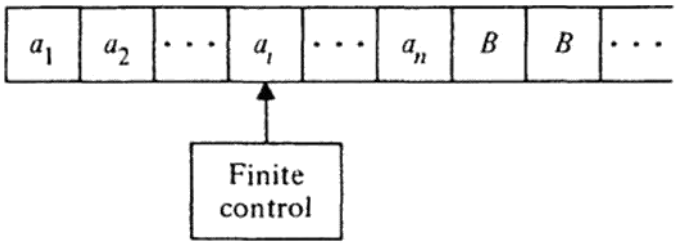
Turing Machine (TM), a simple mathematical model of a computer. The Turing Machine models the computing capability of a general-purpose computer. The Turing Machine is studied both for the class of languages it defines (called the recursively enumerable sets) and the class of integer functions it computes (called the partial; recursive functions). TM can be divided into n-no of cells (or) squares. TM is similar to finite automaton but with an unlimited and unrestricted memory. Each cell contain finite no of symbols along with blank spaces.

Turning Machine Model

It is a more powerful model than many models. It was introduced by Alan Turing in 1936. It can do everything that a real computer can do. It is mathematical model which recognize recursive enumerable languages

The basic model of a Turing Machine has

- ✓ a finite control,
- ✓ an input tape that is divided into cells, and
- ✓ a tape head that scans one cell of the tape at a time.



The tape has a leftmost cell but is infinite to the right. Each cell of the tape may hold exactly one of a finite number of tape symbols. Initially, the n leftmost cells, for some finite n≥0, hold the input, which is a string of symbols chosen from a subset of the tape symbols called the input symbols. The remaining infinity of cells each hold the blank, which is a special tape symbol that is not an input symbol.

In one move the Turing machine, depending upon the symbol scanned by the tape head and the state of the finite control,

1. Changes state,
2. Prints a symbol on the tape cell scanned, replacing what was written there, and
3. Moves its head left or right one cell.

Formally, a Turing Machine (TM) is denoted

$M = (Q, \Sigma, \Gamma, \delta, q0, B, F)$

where

Q is the finite set of states,

Γ is the finite set of allowable tape symbols,

B, a symbol of Γ, is the blank,

Σ, a subset of Γ not including B, is the set of input symbols,

δ is the next move function, a mapping from Q x Γ to Q x Γ x {L, R}.

q0 in Q is the start state,

F⊆Q is the set of final states.

An Instantaneous Description (ID) of the Turing machine M is denoted by α1qα2. Here q, the current state of M is in Q; $\alpha_1\alpha_2$ is the string in Γ* that is the contents of the tape up to the rightmost non blank symbol or the symbol to the left of the head, whichever is rightmost. Finally, the tape head is assumed to be scanning the leftmost symbol of α_2 or if $\alpha_2 = \varepsilon$, the head is scanning a blank.

We define a move of M as follows. Let X1X2…….XI-1qxi…….Xn be an ID. Let (qi, Xi) = (p, y, L). If i-1=n, then take xi as B(blank). If i=1, then there is no next ID, as the tape head is not allowed to fall off the left end of the tape. If i>1, then we write

X1X2……..XI-1qxi…..Xn ├ X1X2…….XI-2pXI-1YXi+1…..Xn ---------- 1

If δ(q,xi)=(p,y,R), thgen change of ID is

X1X2……..XI-1qXI…..Xn ├ X1X2…….XI-1YpXi+1..Xn ---------- 2

If i-1=n, the string Xi…….Xn is empty and the right side of 2 is longer that left side.

The language accepted by M, denoted L(M), is the set of words in Σ * that cause M to enter a final state when placed, justified at the left on the tape of M, with M in state q0, and the tape head of M at the leftmost cell. The language accepted by M = (Q, Σ, Γ, δ, q0, B, F) is

{w | w in Σ * and q0w |---* α1 p α2 for some p in F and α1 and α2 in Γ* }.

2. Show that the class of Turing Machines with stay option is equivalent to the class of standard Turing machine (UQ:MAY'08)

Turing machine with stay options:

Instead of always moving left or right, the head can stay in a move with a stay option. The transition f n is modified to be δ: QxΓ'x{L,R,S} The class of Turing machines with stay option is equivalent to class of standard Turing machine A. Tm M with stay option can be stimulated by a standard, the Turing machine $\hat{M}$. A move of M involving S can be stimulated by 2 moves in $\hat{M}$. δ(p,a)=(q,b,s)=> { δ^m(p,a)=(r,b,R) δ^m(r,*)=(q,*,L)} if M accepts some string , it will be eventually get printed out.

Computation of an enumerator:

An enumerator starts out with a blank input tape. If an enumerator does not halt , it may print an infinite list of strings.

The language recognizable by the enumerator is the collection of strings that is eventually prints out. An enumerator may generate strings of the language, it recognizes in any order possible with repetition.

The class of TM with stay option is equivalent to the class of standard TM. Since a TM with stay option is clearly an extension of the standard model, it is obivious that any standard TM can be stimulated by one with a stay option .

To show the converse, let

M=(Q, Σ, Γ, δ, q0,B, F) be a TM with stay option to be stimulated by a S TM ^m=(^Q,Σ,

Γ',^δ,q0,B,F)

For each move of M, the stimulating machine ^m does the following:

If the move of M does, not involve the stay option, the stimulating machine performs one move, essentially identical to the move to be stimulated.

δ : QxΓ'^ ->QxΓ' ^x{L<R}^ multidimensional.

δ : QxΓ'->2^QxΓ'x{L,R}

q0w ⊢———— x1qfx2, qf€F

if S is involved in the move of M, then ^M will take 2 moves: the first rewrite the symbolsand moves the read – write head right; the second moves the RW head left, leaving the tape contents unaltered.

The simulating machine can be constructed by M by defining ^δ as follows:

For each transistion

δ(qi,a)=(qj,b,LorR)

For each S transistion δ(qi,a)=(qj,b,s)

We put into ^δ , the corresponding transistion ^δ(^qi,a)=(^qjs,b,R)

And ^δ(^qjs,c)=(^qj,c,L) for each c in Γ'.

It is reasonable obvious that every computation of M has a corresponding computation of ^m. so that ^M can stimulate M.

3. Explain Turing machine as a computer of integer functions (NOV'07,NOV'15)

The turing machine can be viewed by as a computer of integer functions.

1. Represent integer in unary the integer i>=0 represented by a string 0i.
2. If a function has k arguments i1,i2,i3…..ik then these integers are placed on tape represented by its as 0i1,0i2,…….0ik
3. If TM halts (whether or not in an accepted state) with a tape consisting of om for some m,f(i1,i2,…..in)=m where f-> function of k arguments computed by TM.
4. Note that one TM may compute a function of one argument a different function of two arguments and so on.

5. If TM computers functions of k arguments, then f need not have a value for all different k-tuples of integers i1,i2,….ik
6. If f(i1,….il) is defined for all i1….ik then f is total recursive function.
7. A function f(i1…..ik) partial recursive function similar to regular expression language since they are computed by TM's that they may or may not halt on given input.
8. The total recursive functions corresponds to the recursive language, since they are comuted by TM's that always halt.

Example:

F:N->N

F(x)=x+1(note: blank space replaced by zero)

Solution

Assume that the input is encoded in unary form. Find first blank(B) and changes it as 0.

Let M= (Q, Σ, Γ, δ, q0, B, F) where

Q={q0,q1}

F={q1}

Initial State ={q0}

Γ={0,B}

Σ={0}

δ transition function,

States	**Inputs**	
	0	**B**
q0	(q0,0,R)	(q1,0,R)
q1	-	-

Let us consider output x=3. This can be encoded as 0^3 and placed on the tape

(q0,000B) ⊢ (q0,000B)

⊢(q0,000B)

⊢(q1,0000B)

Note:

If found 0-> no state change

If found 1 -> state changed

If found 'B'-> state changed control transferred to towards left cell.

The machine halts an accepting state q1 computing the successor of x . This can be shown in following figure.

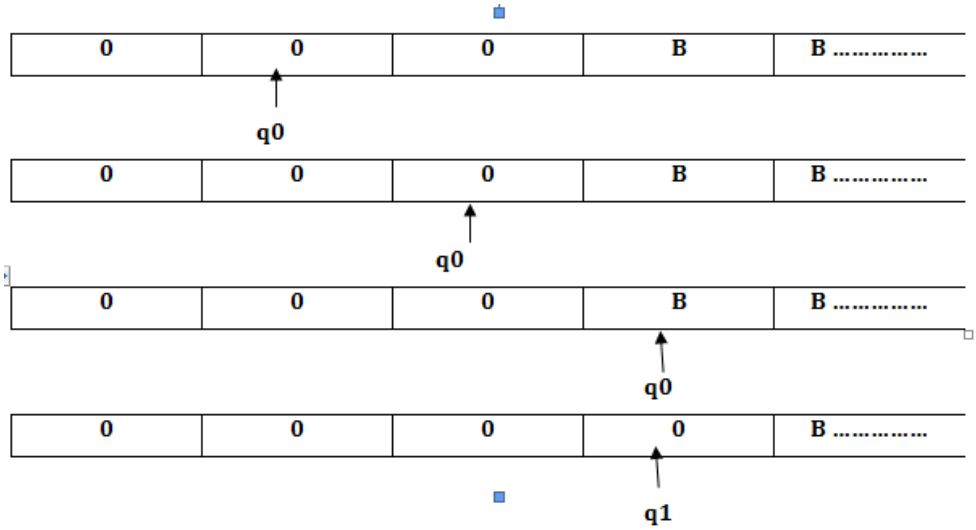
0	0	0	B	B……..

The input 0^3 placed on the tape followed by blanks.

Initial state =q0

Current input symbol=0

δ (q0,0)=(q0,0,R)



The first blank symbol is replaced by 0 and the state is changed to q1 which is the acceptance state. The machine halts as state q1 leaving the output x+1 on the tape.

Concept

From the state q0 on input symbol , it moves right until it sees a blank symbol without changing the tape symbol or state.

On seeing the first blank symbol it is changed to 0, moves right and enters in a halt string . The tape now will contain the output.

4. Give the technique for TM construction in detail (UQ:NOV'07,NOV'09,NOV'14,MAY'15)

Technique of TURING MACHINE CONSTRUCTION

Storage in Finite control:

Finite control can be used to hold finite amount of information. It is considered as pair of elements like [q0,a].

Where first component exercising control.

Second Component stores symbol infinite control.

Let M = (Q, Σ, Γ, δ, [q0, B],B, F)

Q is {q0,q1}X{0,1,B}

Q={[q0,0],[q0,1],[q0,B],[q1,0],[q1,1],[q1,B]}

F={[q1,B]}

Where δ is defined as;

1)a) δ([q0,B],0)=([q1,0],0,R)

b) δ([q0,B],1)=([q1,1],1,R)

2a) δ([q1,0],0)=([q1,0],1,R)

b) δ([q1,0],0)=([q1,1],0,R)

Multiple tracks:

Tape of Turning Machine composed of several track. Each track can hold one symbol and tape occupied of turning machine , consist of tuples one component on each track.

0	1	0	1	1	$
B	B	1	0	B	B
B	B	1	0	B	B

Procedure:

1. Input is placed on tape.
2. Turning machine writes number two in binary on second track.
3. Copies the first track onto third track.
4. Then second track is subtracted as many as possible from third track effectively, dividing third track by second track and leaving remainder.
5. So third track contains remainder after execution. If remainder is Zero, the number on first track is not prime else if remainder is non zero, then increment second track by one and process is repeated until first track equals to second track.
6. If first track equals second track, then number on first track is prime.

Eg.

Find the number 5 is prime or not

5	5
2	2
5	3

5
2
1

5
3
5

5
3
2

5
4
5

5
4
1

5
5
1

Here first track is equal to second track.

Hence the number is prime.

FINITE CONTROL: Eg: construct a turing machine that takes an input and checks whether it is even or not, the input is placed into first tape between ¢ and $. the integer 2 is placed on the second track.

The input on the first track is copied into third track.

The number on the second track, is subtracted from the third track if the remainder is same as the number ,in the second track then the number on the first track is even.

If it is >2, continue this process until the remainder on the third track <=2, if =2 , no. is even otherwise odd.

TRACK1:	8	8	8	8	8
TRACK2:	2	2	2	2	2
TRACK3:	8	6	4	2	0

Checking off symbols:

Checking of symbols is the useful method when a Turing Machine recognizes language definition by repeated strings and to compare the length of the substrings.

{ww/w in S*} (or) {ww^R/win Z*}

The checking off symbol method is implemented in Turing machine by introducing a new track on the tape with symbols blank.

It is also useful when length of the substrings must be compared.

{a^I b^i>=1}

Or

{a^ib^jc^k/ i!=j, j!=k} in turing machine.

Checking of symbols implemented by introducing new track on tape+ symbols blank of v.

Eg: consider a turing machine m=(Q, Σ, Γ, δ, q0,B, F) which recognizes in the language.

{wcw/ w in (a+b)*} let

Q={{q,d}, q=q1,q2,……q9 and d=a,bor B}

The second component of the state is used to store an input symbol.

€={B,d}/d=a,b,or c}

Shifting over:

Shifting all non blank symbols a finite number of cells to the right to left. The turing machine can return to the vacated cells and print symbols.

Subroutines:

* It is used for computer language which performs some task repeatedly.
* A turing machine can simulate any type of subroutine found in any programming language .
* A part of the turing machine can be used as subroutines. This subroutine can be called for any number of times in the main Turing machine programming.
* Turing machine suboutine designed with a initial state ans a return stateand temporarily halts.
* Then call is effected by entering the initial state for the subroutine and return is effected by the move from the return state.
* To design a TM that "calls " the subroutine a new set of states for the subroutine is made and move from the return state is specified.

5. Explain in detail the various types of Turing machine with neat sketch. (MAY'15)

The basic model of the Turing Machine is equivalent to many other modified versions. They are

1. Non deterministic Turing machine
2. Two way infinite tape.
3. Multi tape Turing machine
4. Offline Turing machine
5. Multi head Turing machine
6. Multidimensional Turing machine
7. Restricted Turing machine

1. Non Deterministic Turing machine

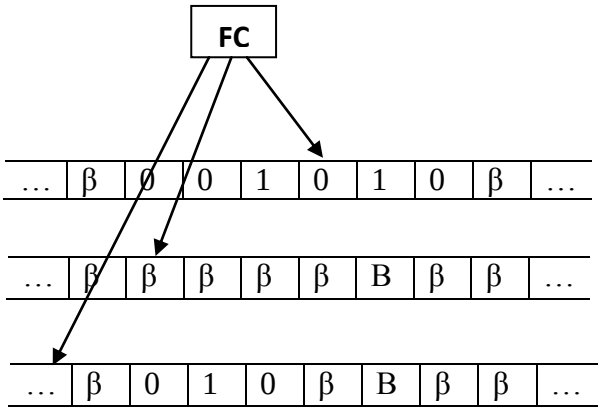
- It has a finite control and a single one way infinite tape.
- For the given state and the tape symbol scanned by the tape head the TM has a finite number of possibilities for its next move.
- Here the computation is like a tree whose branches correspond to different possibility.
- If some branch of the computation leads to the accept state the machine accepts the input.

2. Two way Infinite Turing machine

- This is similar to the basic model.
- But here the tape is infinite to the left as well as to the right.
- Infinite number of blanks will be in both sides.
- The initial ID is q_0w

3. Multi tape Turing machines

- It has a finite control with many tapes say 'k' tape heads.
- Each tape is too way infinite.
- Initially the input will be on the first tape and the other tapes are kept blank.
- Depending on the states of the PC and the symbol scanned by each of the tape heads the TM can
    i. Change the state
    ii. Print a new symbol on each of cells scanned by the tape heads.
    iii. Move each tape head independently [some to right, some to left and some other stationary]



4. Offline Turing Machine

- This is a multi tape TM whose input tape is read only.
- The input is surrounded by some end markers.
- The TM is allowed to move the input head in between the end markers.

5. Multi head Turing machine

- It has 'k' heads, k>1.
- Depending on the state and the symbol scanned by each head, the heads move independently.

6. Multidimensional Turing machine

- It has a finite control and a tape consisting of k-dimensional arrangement of the cells infinite in all directions for some 'k'.
- Initially the input string is along one axis and the tap
- Depending on the state and the symbol scanned the TM
    i. Changes state
    ii. Prints a new symbol.
    iii. Moves the tape head in one of the2k direction positively or negatively along the k-axes.

7. Restricted Turing machine

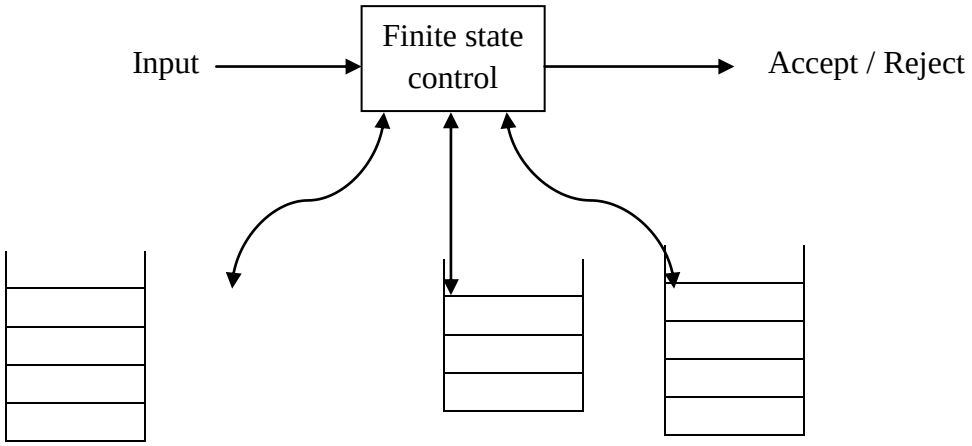
i. Counter machines
ii. Multi stack machines
iii. Turing machine with semi infinite tapes.

(i) Counter machine

- It has the same structure as the multistack machine.
- It has ability to store a finite number of the integers. (counters)
- It can able to add or subtract one from the counter but it cannot tell two different nonzero counts from each other.

(ii) Multistack machine

- A k stack machine is a deterministic PDA with k stacks.
- It has a Finite control which is in one of the state.
- It has a finite stack alphabet which is used for all stacks.
- A move of the multistack machine is based on
  - **a.** The state of the finite control.
  - **b.** The input symbol read.
  - **c.** The top stack symbol on each stack.



A multi stack machine with 3 stacks

In one move the multi stack machine can

(i) Change to a new state
(ii) Replace the top symbol of each with a string or more stack symbols.

(iii) Turing machine with semi infinite tapes

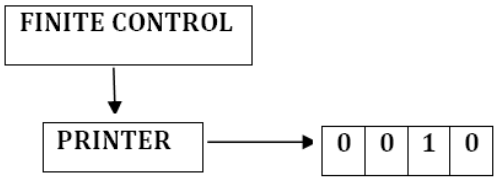
- Here the tape is semi infinite that is, there are no dells to the left of the initial position of the tape head.
- It can simulate one whether is infinite in both directions.
- It accepts any regular language.

Universal Turing machine [UTM]

It is a programmable computer. Since when it is given a program a description of another computer it makes itself as if it were that machine while processing the input.

6. Explain the TMS as enumerations (UQ:NOV'07)

Turing machine as enumerator:



A Turing machine with a printer . the language enumerated by E is all strings printed to the printer . strings can be repeated or in any order.

A language is turing machine recognizable iff some enumerator enumerates it.

PROOF:

If an enumerator E, enumerates a language A, then A is turing recognizable. We need to construct M, the turing machine that recognize A, TM M works the following way:

M=" on input w"

1. rune: every time that E outputs a string , compare it with w.
2. if the outputof E matches w, accept else go to1.

   Ie) if w=x, M accepts else go to 1.

Clearly, M accepts those strings that appear on E's list.

If a language A is turing recognizable, then some enumerator enumerates it.

We need to show, if some TM recognizes A, then an enumerator E can be constructed to enumerate A must be over some alphabet Σ. Then A<=Σ*.

Let s1,s2,s3…….. be aninfinite list of all strings in Σ*.E=" ignore the input"

1) for i=1,2,3

a)run M for I steps on each input s1,s2,……si.

b)if M accepts any of these strings , print it.

Eg:

Run M for I steps , i

i	String tested	Printer output
1	S1	
2	S1,s2	
3	S1,s2,s3	

4	S1,s2,s3,s4	S2
5	S1,s2,s3,s4,s5	S2,s5
6	S1,s2,s3,s4,s5,s6	S2,s3
7	S1,s2,s3,s4,s5,s6,s7	S5
8	S1,s2,s3,s4,s5,s6,s7	

7. Discuss about Multidimensional and Non-Deterministic TM in detail (UQ:NOV'09)

MULTIDIMENSIONAL TURING MACHINE

A multidimensional Turing Machine has a finite contol and the tape consist of "K-dimensional " array of cells infinite in all 2k directions, for some fixed "K". Depending on the state and the symbols scanned, the devices change the state, prints new symbols and moves its tape head in one 2kdirection,either positively or negatively, along one of the K axes. Initially the input is along one axis, and the head is at the left hand of the input.

At anytime, only a finite number of rows on any dimension contain non_blank symbols.

B	B	B	a1	B	B	B
B	B	a2	a3	a4	a5	B
a6	a7	a8	a9	B	a10	B
B	a11	a12	a13	B	a14	a15
B	B	a16	a17	B	B	B

Figure: a Two dimensional Tape

**BBBa$_1$BBB*BBa$_2$a$_3$a$_4$a$_5$B*a$_6$a$_7$a$_8$a$_9$Ba$_{10}$B*Ba$_{11}$a$_{12}$a$_{13}$Ba$_{14}$a$_{15}$*BBa$_{16}$a$_{17}$BBB

Figure: b one dimensional Tape

Figure: a represents a two dimensional Turing machine. The rectangle can be represented row by row on a single tape which makes one dimensional simulation. The *'s represents the rows.

NON-DETERMINISTIC TURING MACHINE:

A non deterministic turing machine is advice with the finite control and a single one-way infinite tape.

For the given state and a tape head symbol scanned by the tape head. The machine has a finite number of choices for the next moves.

For δ(q,r) there is a set of triples like

{(q1,Y1,D1),(q2,Y2,D2),……(qk,Yk,Dk)}

Where k is any integer

M accepts an input 'w' if there is any sequence of choices of move that leads from the initial ID with the w as input to an ID with an accepting state.

8. Give the properties of recursive and recursively enumerable languages (UQ:NOV'09)

Recursive and Recursively Enumerable Languages

Result 1:

If a language is recursive then it is recursive enumerable.

Result 2: Recursive languages are closed under complementation.

i.e. If L is recursive then its complement is recursive

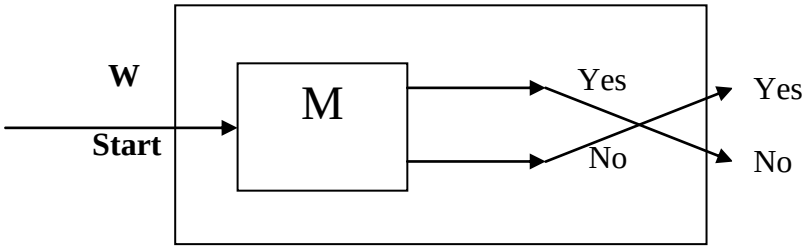
To prove: The complement of L say L' is recursive

Proof: Let L be a recursive language. Then there is a Turing Machine M that accepts L.

.

Construct a Turing machine M' from M such that : M enters a final state on the input W in L, then M' halts in a non accepting state. (Without accepting)

If M halts without accepting then M' accepts the input 'w' in L'. Hence M' is an algorithm and is given as below.



Here 'Yes' corresponds to 'accept' and 'No' corresponds to 'Reject'. The language of M' is L (M') and is the complement of L. Hence the proof.

Result 3: The union of two recursive language is recursive.

To prove: $L_1 U L_2$ is recursive.

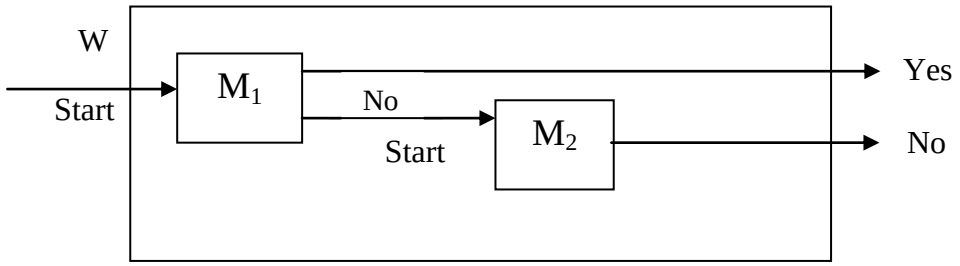
Proof: Let L_1 and L_2 two recursive languages. Then there exists Turing machine M_1 and M_2 such that M_1 and M_2 accept L_1 and L_2 respectively.

Construct a Turing machine M that accepts $L_1 U L_2$ as below:

- First simulate M on M_1.
- If M_1 accepts inputs w in $L_1 U L_2$ then M accepts.
- If mot, M simulates M_2 and accepts iff M_2 accepts.

Since M_1 and M_2 are algorithms (used to say Yes or No), M_1 and M_2 will not hang, they will halt (in accepting state or non accepting state). M will surely halt (It will not hang).

Hence M is an algorithm. M is given below:



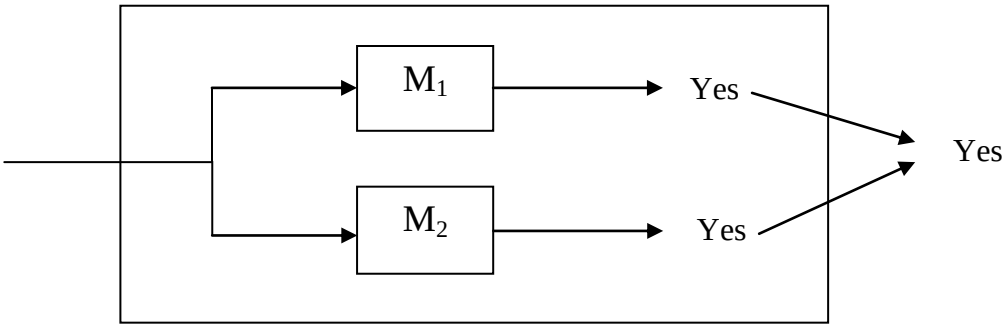
Now the language of M is $L_1 \cup L_2$. Hence the result.

Result 4: The union of two recursively enumerable languages enumerable.

Proof: Let L and L_2 be two recursively enumerable languages.

There are Turing machine M_1 and M_2 such that M_1 accepts L_1 and M_2 accepts L_2. Construct a Turing machine M that accepts $L_1 \cup L_2$ as below:

The above procedure will not work here. Since M_1 may not halt for some input W in L_1. Hence M is constructed as below:



Now the language of M is $L_1 \cup L_2$. Hence $L_1 \cup L_2$ is recursive enumerable.

Result 5: If a language L and its complement L' are both recursively enumerable, then L (and hence L' by **result 2**) is recursive.

To prove: L is recursive

Proof: Let a language L and its complement L' are recursively enumerable. Then there are Turing machine M_1 and M_2 that accepts L and L' respectively.

Construct a Turing machine M that accepts L and say either 'Yes' or 'No' as below: M is connected in such a way that M simulate M_1 and M_2 simultaneously.

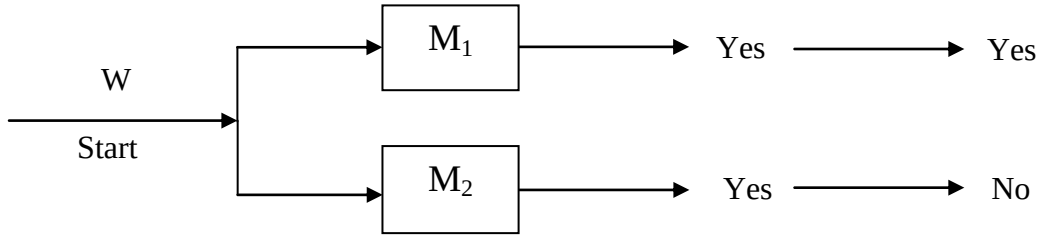
- If M_1 accepts then M will accept it.

- If M_2 accepts then M will reject it.

Hence M will always say either 'yes' or 'no' and not both.

Hence M is an algorithm that accepts that accepts L. Hence L is recursive. {Proved Ans (1) } Hence L' is recursive. {Ans (2)}

M is shown below:



Hence we have If L and L' are a pair of complementary languages then either

i. Both L and L' are recursive (or)
ii. Neither L nor L' are recursively enumerable.

9. **Explain the Universal TM**

UNIVERSL TURING MACHINE:

Turing machine that can be performed to solve any problem that can be solved by any Turing machine of any type is universal Turing machine.

INPUT TO U:

The encoding 'M' of TM, M and encoding of w of string w€∑*.

BEHAVIOUR:

U halts on input 'M', 'w, if only if M halts on input w encoding of Turing machine and Input strings:

Let M={Q,{0,1},{0,1,B},$^{\delta}$,q1,{q2}}

∑={0,1}

┌={0.,1,B}

Q={q1,q2,…q4}

q1=start state

q2=final state

the universal language Lu consists of set of binary string in the form of pairs(M, W)

where

M-> Turing machine encoded in binary

W-> its binary input string

GENERIC MOVE:

1.Symbols 0,1,B => X1,X2,X3 respectively

0=X1 initial state

1=X2

0=X3 final state

Such that w€Lu(1,l)

Lu=L(u)

Where (M, W)<=L(u)

Since the input to u is binary string u is in fact some Lj in the form of binary string.TM from the table developed in the construction of language. The universal TM

Describes the motion of the universal computability of the computer.

10. Explain variation of TM.

Write about Multitape TM (Refer above answer)

It consist of n tapes.

It has 5 tuples.

T=(Q, Σ, Γ, δ, q0,B, F)

$\Delta : Q * (\Gamma \cup \{\Delta\}^n\}$

$Q \cup \{ h_a. h_r \} * (\Gamma \cup \{\Delta\}^n * (R,L,S)^n$

(q1x1a1y1,x2a2y2,……xnanyn)

(q0, Δx1,…… Δxn)

11. Explain Primitive Recursive Functions.

Primitive Recursive Function over N and Σ

1. **Initial Function.**

Zero function Z=Z(x)=0

Successor function S=S(x)=x+1

Projection function $U^n=U^n(X1,…..Xn)=xi$

Nil(x)=A

Cons a(x)=ax

Cons b(x)=bx

A=abab

Nil(abab)=A

Cons a(abab)=aabab

Cons b(abab)=babab

Theorem: If f1,f2,….fk are partial function of n and g is Partial Function of k then g(f1(x1,x2,….xn),f2(x1,x2,….xn)……..fk(x1,x2…..xn))

Poof:

Let f1(x,y)=x+y

f2(x,y)=2x

f3(x,y)=xy

g(x,y,x)=x+y+z

g(f1(x,y),f2(x,y),f3(x,y)) =g(x=y,2x,2y)

= x+y+2x+xy

=3x+y+xy

H(x,y) = 3x+y+xy

Partial recursive Function over N

A total function f over N is called primitive recursive.

i. If it is any one of three initial function

ii. If it can be obtained by applying composition and recursion a finite number of times to set of initial funxtion.

EXAMPLE:

Show that function f1(x,y)=x+y is primitive recursive

Solution

F1 is function with 2 variables

Consider one variable x.

F1(x,0)=x+0=x

$G(x)=x=U_1^{1(x)}$

f1(x,y+1)=x+(y+1)

=(x+y)+1

=f1(x,y)+1

H(x,y,f1(x,y))=f1(x,y)+1=S(f1(s,y))

$=S(U_3^3(x,y,f1(x1,y1))$

Note: f(x1,x2,….xn,0)=g(x1,x2,…..xn)

f(x1,…..xn,y+1)=h(x1,x2…..xn,yn) f(x1,x2…xn,y+1)

Primitive Function:

a. **Predecessor function:**

P(x)=x-1 if x is not equal to 0

P(x)=0 if x=1

b. **Proper Subtraction function:**

x-y=x-y if x is greater than equal to y

c. **Absolute value function**

|x|=x if x>=0

|x|=-x if x<0

d. **Min(x,y)**

Used to find minimum of x and y

Primitive function over {a,b}

Σ={a,b} w=ab

f(A)=W

f(ax)=h1(a,f(x))

f(bx)=h2(b,f(x))

where h1 and h2 are variables.

Example:

a. Constant function a and b

a(x)=a

b(x)=b

a(x)=consa(nil(x))

a is pointing initial function so it is primitive recursive function

b. Identity function

id(A)=A

id(ax1)=cons a(x1)

id(bx2)=cons b(x2)

c. Concatenation

Concat(x1,x2)=x1,x2

d. Transpose

trans(x)=x^T

trans(A)=A

trans(ax)=concat(trans(x),a(x))

trans(bx)=concat(trans(x),b(x))

e. Head Function (i.e head(a1,a2,….an)=a)

head(A)=a

head(ax)=ax

head(bx)=bx

f. Tail function (i.e. tail(a1,a2,….an)=a)

tail(A)=A

tail(ax)=id(x)

tail(bx)id(x)

g. Conditional function x1≠A then x2 else x3

Cond(x1,x2,x3)=if x1 ≠ A then x2 else x3

12. Explain Recursive function.

A function is recursive if it can be obtained from initial function by a finite number of application of composition, recursion and minimization over regular function.

g(x,y)=2y-x

2y-x where x is evev it will be achieved

And x is equal to x/2 . So it is called partial function.

Ackermann's Function:

A(0,y)=y+1

A(x+1,0)=A(x,1)

A(x+1,y+1)=A(x,A(x+1),y)

Example:

1. **A(1,1)**

A(1,1)=A(0+1,0+0)

=A(0,A(1,0))

=A(0,2)

=3

2. **A(1,2)**

A(1,2) =A(0+1,1+1)

=A(0,A(1,1))

=A(0,3)

=4

Pondicherry University Questions

2 Marks

1. What is a turning machine? (UQ:APR/MAY'12, APR'16) (Qn.No.1)
2. What is off-line Turing Machine? What is a multidimensional TM? (UQ:APR/MAY'12) (Qn.No.20)
3. What is non deterministic TM (UQ:APR'13) (Qn.No.21)
4. What are the required fields of an instantaneous description or configuration of a TM. Or Define the basic guidelines for designing a TM (UQ:NOV'11) (Qn.No.22)
5. State the powers of single tape and multitape TM (UQ:NOV'12)) (Qn.No.23)
6. Define partial recursive fuction and total recursive function. (UQ:NOV'12)) (Qn.No.24)
7. What is off-line Turing Machine? (UQ:NOV'14,MAY'15) (Qn.No.25)
8. What is the instantaneous description OF A TM **(**MAY'15) (Qn.No.8)
9. What are the techniques of tuning machine construction? (NOV'15) (Qn.No.9)
10. What is primitive recursive function ?(APR'16) (Qn.No. 41)

11 Marks

1. Show that the class of Turing Machines with stay option is equivalent to the class of standard Turing machine (UQ:MAY'08) (Qn.No.1)
2. Explain Turing machine as a computer of integer functions (NOV'07,NOV'15) (Qn.No.2)
3. Give the technique for TM construction in detail (UQ:NOV'07,NOV'09,NOV'14,MAY'15) (Qn.No.3)
4. Explain the TMS as enumerations (UQ:NOV'07) (Qn.No.4)
5. Explain TM model in detail (UQ:NOV'09) (Qn.No.5)
6. Discuss about Multidimensional and Non-Deterministic TM in detail (UQ:NOV'09) (Qn.No.6)
7. Give the properties of recursive and recursively enumerable languages (UQ:NOV'09 APR'16) (Qn.No.8)
8. Explain in detail the various types of Turing machine with neat sketch**. (MAY'15 ,APR'16)** (Qn.No.4)

UNIT V

Introduction to Computational Complexity: Time and Space complexity of TMs –Complexity classes – Introduction to NP-Hardness and NP-Completeness

---

2 Marks

1.What is meant by Space complexity? (NOV'14)

It is a Storage representation of language. . Turning machine cannot rewrite on the input and that only the length of the storage tapes used for counting. The language recognized by M is said to be of space complexity.

2.What is meant by space bounded turing machine?

M scans at most S(n) cells on any storage tapes then M is said to be an S(n). Which represents " Space complexity S(n) or Space bounded Turing machine".

3.What is meant by semi finite storage?

Turing machine M can be considered as a machine which contains input tape with "end markers" K can be referred as semi-finite storage tapes.

4.What is meant by marker?

It denotes the end of the edge.

5.Give the diagram of space complexity?

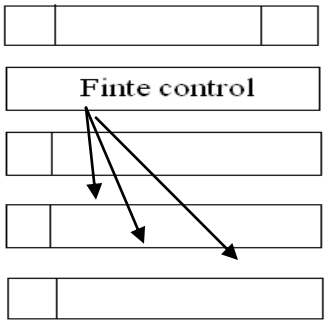


Fig) Multiple TM + Read Only Input

6.What is meant by time complexity of TM? Nov'11,May '2012, Dec 2013

The language recognized by M is said to be T(n)- Time bounded TM\ Time complexity T(n). The turning machine has K , two-way infinite tapes, one of which contains the input tape may be written upon.

7.Differentiate between space and time complexity? (NOV'15)

SPACE COMPLEXITY	TIME COMPLEXITY
In space complexity we can use more than one tape at a time.	But in Time complexity, possible to use only one at a time
In space complexity, one way infinite tapes are available.	In time complexity, two-way infinite tapes are available.

8.What are complexity classes? What are they?

All the families of language are called complexity classes. They are

- DSPACE (S(n))
- NSPACE (S(n))
- DTIME (T(n))
- NTIME (T(n))

9.What is NP complete problem?

It includes many problems that are nature and examine seriously to get efficient result. This problem will have polynomial solutions, it has 2 types

1. NP
2. Satisfiability problem.

10. What is the satisfiability problem? APR 2014

It is used for Boolean expressions which are composed of variables, parenthesis and operators. The operators have logical and , logical or and negation.

The order is ך,^,v.

→ variables are referred as 0 and 1.

11. What are the example for NP complete problems?

1.Vertex cover problem.

2.Hamilton circuit problem.

3.Linear integer programming.

12. What is the vertex cover problem?

- Three-CNF, Satisfiability is a convenient problem is reduced to another problem in order to show in complete problem.
- Another NP - complete problem is easy to reduce other problem is the vertex cover problem.
- Let G={V,E} - undirected graph with set of vertices and edges.

- A subset A $\subseteq$ V said to be a vertex cover of G if for every edge (V,W) in E.
- It represents in terms of language Lvc, (Lvc-vertex cover problem).

13. Define Hamilton circuit problem?

→Parallel to directed graph.

→represented in terms of language Lh and Ldh by encoding graphs in vertex cover problem.

14.What is DSPACE (S(n))?

Family of space complexity S(n) is denoted by DSPACE (S(n)). DSPACE (n)→ $\log_2 n$

15.What is NSPACE (S(n)?

Non-Deterministic space complexity S(n) is called as NSPACE (S(n)). NSPACE (n)→$\log_2 n$→$n^{1/2}$

16.What is DTIME (T(n))?

The family of time complexity T(n) is denoted as DTIME (T(n)). DTIME (n) → n→n^2

17.What is NTIME(T(n))?

Non-Deterministic time complexity T(n) is denoted as NTIME (T(n)). NTIME (n)→n.

18.What are the Conditions of satisfiability problem?

E1	E2	E1∧E2
1	1	1
E1	E2	E1∨E2
0	1	1
E1	¬E1	
1	0	

19.what is the vertex cover problem?

- Three-CNF, Satisfiability is a convenient problem is reduced to another problem in order to show in complete problem.
- Another NP - complete problem is easy to reduce other problem is the vertex cover problem.
- Let G={V,E} - undirected graph with set of vertices and edges.
- A subset A $\subseteq$ V said to be a vertex cover of G if for every edge (V,W) in E.
- It represents in terms of language Lvc, (Lvc-vertex cover problem).

20.Define NP Hardness. Nov'11

NP-hard (Non-deterministic Polynomial-time hard), in computational complexity theory, is a class of problems that are, informally, "at least as hard as the hardest problems in NP". More precisely, a problem *H* is NP-hard when every problem *L* in NP can be reduced in polynomial time to *H.*As a consequence, finding a polynomial algorithm to solve any NP-hard problem would give polynomial algorithms for all the problems in NP, which is unlikely as many of them are considered as hard.[

21. Specify any two complexity classes May '2012

Example for a complexity class , the class NP is the set of decision problems whose solutions can be determined by a non-deterministic Turing machine in polynomial time, while the class PSPACE is the set of decision problems that can be solved by a deterministic Turing machine in polynomial space.

22. What is NP completeness? Nov'13,MAY'15

In computational complexity theory, a decision **problem** is **NP-complete** when it is both in **NP** and **NP**-hard. The set of **NP-complete problems** is often denoted by **NP**-C or NPC. The abbreviation **NP** refers to "nondeterministic polynomial time".

23.What are P classes APR'13

Class P: The Turing machine restricted to a number of moves bounded by a fixed polynomial in n, p(n), where n = |x|, n is the length of the input string. The machine must accept all strings in a language in polynomial time in order to be in P. P is a set of languages, also called a class of languages.

24. State Travelling Salesperson problem? APR'13.

Given a set of cities and the distance between each possible pair, the **Travelling Salesman Problem** is to find the best possible way of 'visiting all the cities exactly once and returning to the starting point

25.What is DFA With NFA APR'14

Deterministic vs nondeterministic

For every nondeterministic automata, there is an equivalent deterministic automata

Finite acceptors are equivalent iff they both accept the same language

$$L(M_1) = L(M_2)$$

In DFA, label resultant state as a set of states

{q1, q2, q3,…}

For a set of |Q| states, there are exactly 2^Q subsets

Finite number of states

26.What are NP classes? (APR'14,NOV'15)

Problems solvable in time which increases faster than polynomial in N, but once the solution is obtained , it can be verified in time polynomial in N

27.When will you call an expression is satisfiable? Dec 2013

Does there exist a truth assignment making the function true.

11 Marks

1.Discuss the Time and Space complexity of TMs APR'11, NOV'2011

SPACE COMPLEXITY

- Storage representation of language.
- Consider the off line TM of fig 1(a). M has a read only input tape . Here M can be considered as a machine which contains input tape with "end markers" K can be referred as semi-finite storage tapes.
- M scans at most S(n) cells on any storage tapes then M is said to be an S(n). Which represents " Space complexity S(n) or Space bounded Turing machine"?
- Turning machine cannot rewrite on the input and that only the length of the storage tapes used for counting. The language recognized by M is said to be of space complexity.

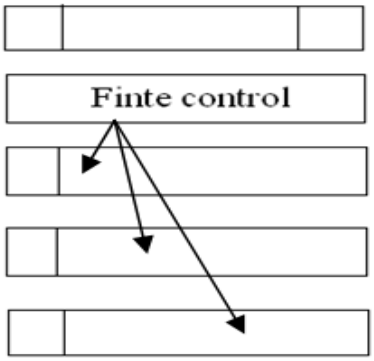


Fig 1(a): Multiple TM + Read Only Input

First, we prove the existence of a space hierarchy. For this we will need the following lemma.

Lemma 3.4 *For any function s* : IN *!* IN*, any O*(*s*(*n*)) *space bounded multi-tape Turing machine can be simulated by a single-tape Turing machine with O*(*s*(*n*)) *space.*

Proof. Let *M* = (*Q;§; ¡; ±; q*0*; F*) be an arbitrary *k*-tape Turing machine. Our aim will be to simulate *M* by a single-tape Turing machine *M0* = (*Q0;§0; ¡0; ±0; q0*0*; F0*) that uses (asymptotically) the same amount of space. To achieve this, we choose a tape alphabet of ¡*0* = (¡*[f"g*)2*k* (\ "" is a symbol that is not in ¡). This is possible, because *k* is a constant. The alphabet allows us to view the tape of *M0* as consisting of 2*k* tracks. Let these tracks be numbered from 1 to 2*k*. The purpose of the odd tracks is to store the contents of the tapes of *M*, and the purpose of the even tracks is to store the positions of the heads of *M*:

track 1 contents of tape 1

track 2 " (position of head 1)

track 3 contents of tape 2

track 4 " (position of head 2)

: : : : : :

Let *Q0* = *Q £ ¡k*. This set of states allows *M0* to store the current state of *M* plus the *k* symbols that are read by the heads of *M*. In this case, a single step of *M* can be simulated by *M0* in two phases. In the ¯rst phase, the head of *M0* searches for the positions of the heads of *M* and stores the symbols that are read by the heads in its state. In the second phase, *M* replaces the symbols and moves the head pointers according to the transition function ± and stores in its state the next state of *M*. Since the space required by *M0* is obviously bounded by the space required by *M*, the lemma follows. *Ut* Given a function *s* : IN *!* IN, the language *Ls* is de¯ned as *Ls* = *fhMiw j M* is a single-tape TM that started with *hMiw* uses at most *s*(*jhMiwj*) cells*g :*

TIME COMPLEXITY

- Consider the multiple tapes TM of fig 1(b). The turning machine has K , two-way infinite tapes, one of which contains the input tape may be written upon.
- N-input word of length \ total no of strings.
- M-makes at most T(n) moves before halting.
- The language recognized by M is said to be T(n)- Time bounded TM\ Time complexity T(n).
- The difference between 2 concepts is , in space complexity we can use more than one tape at a time. But in Time complexity, possible to use only one at a time.

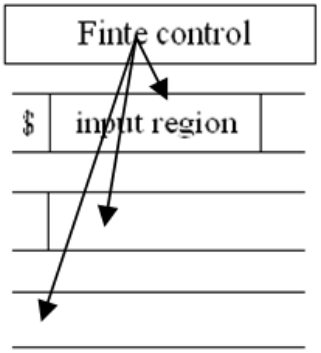


Fig 1(b): Multiple tape TM

Gaps and Speed-ups

First, we show that in certain situations there can be large gaps between complexity classes. Theorem 3.1 *For every computable function f* : IN *!* IN *there are monotonically increasing functions s; t* : IN *!* IN *with*

DSPACE(*s*(*n*)) = DSPACE(*f*(*s*(*n*)) *and*

DTIME(*t*(*n*)) = DTIME(*f*(*t*(*n*))) *:*

The theorem implies that there are functions *t; s* with

DSPACE(*s*(*n*)) = DSPACE(22*s*(*n*)) and

DTIME(*t*(*n*)) = DTIME(22*t*(*n*))

At first glance, this seems to be quite surprising. However, it has been shown, for instance, that DSPACE(*o*(log log *n*)) = DSPACE(1), which explains why such gaps can occur. We will see that for \well behaved" functions *s* and *t* it is not possible to create such gaps.Another phenomenon is that it is quite easy to achieve constant improvements in space or time.

Theorem 3.2 *If L can be decided by an s*(*n*) *space-bounded Turing machine, then L can be also decided by an s*(*n*)=2 *space-bounded Turing machine.*

Proof. Let *M* be any *s*(*n*) space-bounded Turing machine that decides *L*, and let ¡ be the tape alphabet of *M*. In order to obtain an *s*(*n*)=2 space-bounded Turing machine for *L*, simply extend the alphabet by ¡*£*¡. This will allow to encode two cells of *M* in one cell and therefore to reduce the space requirement by a factor of two. *Ut*

Theorem 3.3 *If L can be decided by a t*(*n*) *time-bounded Turing machine, then L can be also decided by an n + t*(*n*)=2 *time-bounded Turing machine.*

Proof. The proof will be an assignment. *Ut* Next we will show that there are hierarchies of complexity classes. Recall that we defined the complexity classes over multi-tape Turing machines.

2.Discuss the Complexity classes of Turning machines? (NOV'15)

Family of space complexity S(n) is denoted by DSPACE (S(n)).

Non-Deterministic space complexity S(n) is called as NSPACE (S(n)).

The family of time complexity T(n) is denoted as DTIME (T(n)).

Non-Deterministic time complexity T(n) is denoted as NTIME (T(n)).

All the families of language are called complexity classes.

DTIME (n) → n→n^2

DSPACE (n)→ $\log_2 n$

NTIME (n)→n

NSPACE (n)→$\log_2 n$→$n^{1/2}$

THEOREM

TAPE COMPRESSION

STATEMAENT

IF L is accepted by an S(n), storage tape = K, for any C>0, L is accepted by C.S(n)

PROOF

Let M1 be an S(n) tape bounded off-line turning machine accepting L.

M2→ new turning machine which simulates M1.

r→constants

M2→keeps track of which of the cells of M1 scanned.

Let r ➔r, C≥2.

M2→ can simulate M, using no more than (S(n)/r) cells.

If S(n)≥r, this number is no more than C·S(n)

If S(n)<r, then m2 can store in one cell, the contents of any tapes.

3.What is Hardness in NP?Explain NOV 2013,NOV'14,MAY;15

Let NTIME[pk(n)] be the nondeterministic time-complexity class with pk(n) = nk. The class NP is defined as NP = k_0 NTIME[pk(n)] Many problems of practical importance, which have been extensively investigated by researchers and for which no deterministic polynomial-time algorithm has been found, belong to NP.

Example satisfiability is a generalization of 2-satisfiability introduced in Example 4.4 where each clause Ci may include any number of literals rather than exactly two.A nondeterministic algorithm for satisfiability can be obtained by guessing any of the 2n assignments of values to the n variables of f and verifying whether it satisfies f:begin {input: f} guess t in set of assignments of values to the n variables of f;if t satisfies f then accept else reject; end. Since both the guessing and the checking of the ith assignment can be done in polynomial time, it turns out that such a nondeterministic algorithm is a polynomial-time one

Example 5.2 3-colorability is a generalization of 2-colorability introduced in where the colours available are three instead of two.1A nondeterministic algorithm is said to be polynomial-time if the required number of steps is O[nk] with k constant.A polynomial-time nondeterministic algorithm for 3-colorability can be obtained by guessing any of the 3n colourings of a graph of n nodes and verifying whether it has the required property.

Example traveling salesman: given a complete weighted graph G and a natural number k, does a cycle exist passing through all nodes of G such that the sum of the weights associated with the edges of the cycle is, at most, k?

A polynomial-time nondeterministic algorithm for traveling salesman can be obtained by guessing any of the n! permutations of n nodes and verifying whether it corresponds to a cycle whose cost is, at most, k. The richness of the class NP is justified by the following theorem. This intuitively states that a problem belongs to this class if and only if the possible solutions are words of a length polynomially related to the length of the instance and it is polynomial-time decidable whether a possible solution is a feasible one. (Note that most of the combinatorial problems we usually encounter share this property.)

Theorem

A language L belongs to NP if and only if a language Lcheck 2 P and a polynomial p exist such that L = {x : 9y[hx, yi 2 Lcheck ^ |y| _ p(|x|)]}.

Proof. If L = {x : 9y[hx, yi 2 Lcheck ^ |y| _ p(|x|)]} where Lcheck 2 P and p is a polynomial, then the following nondeterministic algorithm decides L in polynomial time:

```
begin {input: x} guess y in set of words of length, at most, p(|x|);
if hx, yi 2 Lcheck then accept else reject;
end.
```

Conversely, let L be a language in NP. Then a nondeterministic Turing machine

NT exists which decides L in polynomial time. It is easy to verify that, for any x, each computation path of NT(x) can be encoded into a word of a length of,at most, p(|x|) where p is a polynomial (see Problem 2.7). The language Lcheck is then defined as follows. hx, yi 2 Lcheck if and only if y encodes an accepting computation path of NT(x). It is clear that Lcheck 2 P and that, for any x, x 2 L $ 9y[|y| _ p(|x|) ^ hx, yi 2 Lcheck]. This concludes the proof. 2
Our aim is to identify within NP problems that are inherently more complex.For this, we need the basic concept of an NP-complete problem.

4.Explain NP Complete Problems APR'11, APR'13,MAY'15,NOV'15 (or)

Neat diagram about vertex cover in NP Completeness problem NOV'2011 ,NOV'14

It includes many problems that are nature and examine seriously to get efficient result. This problem will have polynomial solutions, it has 2 types

1. NP
2. Satisfiability problem.

Satisfiability problem

It is used for Boolean expressions which are composed of variables, parenthesis and operators. The operators have logical and , logical or and negation.

The order is ٦,^,v.

→ variables are referred as 0 and 1.

Conditions

E1	E2	E1^E2
1	1	1
E1	E2	E1ᵛE2
0	1	1
E1	٦E1	
1	0	

- An expression is satisfied if there are some assignments of to the variables that give the expression value is 1.
- Satisfiable determines given a Boolean expression is giving a value or not.
- It represents Satisfiability problem has a language L sat as follows.
- Variables of some expressions x1,x2……xm code xi as the symbol x followed by I, so alphabet Lsat is { ^,v, ٦,(),x,0,1}.
- A Boolean expression is said to be conjunctive normal form (CNF) . It is said to be E1^E2^……^Ek and each E1- clause of form α_{i1} v α_{i2}

  Eg for NP complete problems:

  1.Vertex cover problem.

  2.Hamilton circuit problem.

  3.Linear integer programming.

VERTEX COVER PROBLEMS

DEFINITION

- Three-CNF, Satisfiability is a convenient problem is reduced to another problem in order to show in complete problem.
- Another NP - complete problem is easy to reduce other problem is the vertex cover problem.
- Let G={V,E} - undirected graph with set of vertices and edges.
- A subset A $\subseteq$ V said to be a vertex cover of G if for every edge (V,W) in E.
- It represents in terms of language Lvc, (Lvc-vertex cover problem).
- Contains strings of the form: K in binary followed by a marker, followed by list of vertices.

THEOREM

STATEMENT

Lvc , the vertex cover problem is NP complete.

PROOF

- To show Lvc is NP , guess a subset of K vertices and check that it covers all edges.
- Let F=F1^F2^……..^Fq be an expression in 3-CNF where each Fi- clause of form (α_{i1} v α_{i2} v α_{i3}) each α_{ij} being a literal.
- We construct unidirectional graph. G=(V,E) where vertices are pairs of integer (i,j,1<=i<=q,1<=j<=3)

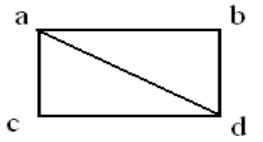
Edge of graph is :

1.) [(i,j),(i,k)] , j≠k

2.) [(i,j),(k,l)] , if $\alpha_{ij} = \neg \alpha_{k1}$

- Each pair of vertices corresponding to same clause is connected by an edge in the first rule.
- Each pair of vertices corresponding to literal and its complement connected by an edge in rule 2.

Eg:



Complement of a and b is c and d

- G has been constructed so that it has a vertex cover of size (2q,q) vertices from 0 to n iff F is satisfiable.
- Each clause expression & must have a literal whose value is 1.
- Select one such literal for each clause delete the q vertices corresponding to those literals from vertex V. some (i,j) will be missing so that we should maintain 2 counters to consider all pair of literals . Thus we conclude that Lvc is the Np complete problem.

HAMILTON CIRCUIT PROBLEM

Definition

→Parallel to directed graph.

→represented interms of language Lh and Ldh by encoding graphs in vertex cover problem.

Theorem:

Statement:

L_d → Directed Hamilton circuit problem NP- complete .

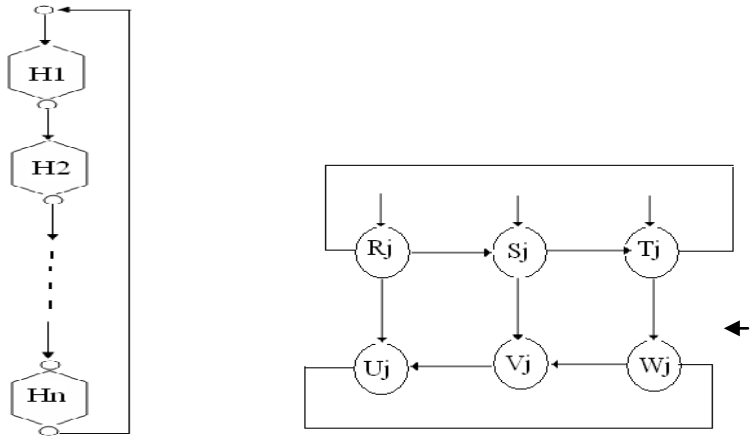
Proof:

To show Ldh in NP, guess a list of arcs and verify that the arcs form a cycle through all vertices. To show Ldh is NP- complete , reduce CNF Satisfiability problem to Ldh. Let F=F1 ^ F2 ^……^Fq be an expression .

Fi→ clause of form (αi1 V αi2 Vαi3)

α ij →literal

Let x1,x2…..xn be the variable of F we construct a graph composed of two types of subgraphs. For each variable xi' there is a subgraph of the form shown in figure.



Case (i)

The next 2 vertices on the circuit are Sj and Tj. Then the circuit must continue with Wj or Vj , Uj is inaccessible. Thus in this case it leaves at Uj.

Case(ii)

The next 2 vertices on the circuit Sj and Vj. If the circuit doesnot go to Uj then Vj will be inaccessible.

Case(iii)

The circuit goes directly to Uj. If it goes to Wj. The circuit cannot include TJ, because it is successor and it is already used, So again it must leave by Vj.

INTRGER LINEAR PROGRAMMING

Theorem

Statement

Integer linear programming in NP-complete.

Proof

We begin by guessing the sign's of xi's and adding n constraints xi≤ 0 depending on the sign guessed. Then guess a row I and a constant Ci in the range bi≤ci<bi+α such that in some solution xo , we have $a_i x_0 = C_i$. We have to recorder the guessed C1,C2……Ck such that

1) $B_i \le c_i < b_i + (\alpha_q)^{2q+1}$

2) $A_x \ge b$ has non negative integer

iff $a_i X = C_i$, $1 \le i \le k$,

$a_i X \ge bi$, $k \le i \le m$.

Let A_k→first rows of A.

C→vector (C1,C2,….Ck)

Case(i)

The rank of A_k<n an integer vector Z , Z ≠ 0 → magnitude greater than $(\alpha_q)^{2q}$ such that

$A_k Z = 0$ → position value.

Z→ integer matrix

A_{11}→value is 0

Therefore if $A_X X_0 = C$ it follows $A_k(Xo+d_z) = C$ for any integer d.

Case(ii)

The rank of A_k is in n, unique X satisfying $A_k X=C$. the non-deterministic process of guessing Ci's repeats almost n times, follows any number of times.

Best case→$\log_2 \alpha$

Ci→constants

di→series of integers

A→matrix

α →powers

Avg case→n log n

Worst case→n^2

6. Discuss on restricted satisfiability problem NOV'12 (or)Discuss Cook's problem APR'13

Cook has shown that 3-SAT, the Boolean satisfiability problem restricted to instances with exactly three variables per clause, is NP-complete. This is a tightest possible restriction on the number of variables in a clause because as Even et al. demonstrate, 2-SAT is in P. Horowitz and Sahni point up the importance of finding the strongest possible restrictions under which a problem remains NP complete.

First, this can help clarify the interesting boundary between problems known to be in P and those that are not. Second, it can make it easier to establish the NP-completeness of new problems by allowing easier transformations. To prove the Euclidean travelling salesman problem NP-hard, Papadimitriou first reduces 3-SAT to 3-SAT where each variable appears in at most five clauses.

The question arises, are any further reductions in this direction possible? In this note we show that 3-SAT remains NP-complete even when each variable appears at most four times.

Let r,s-SAT denote the class of instances with exactly r variables per clause and at most s occurrences per variable. We prove the 3,4-SAT result to be the strongest possible and show that 3,3-SAT is in fact trivial. In addition we show that the Boolean satisfiability problem is solvable in linear time if no variable appears more than twice, regardless of the number of variables per clause.

All Boolean expressions are taken to be in conjunctive normal form with no repeated variables in a clause.

2. The reduction

Start with any 3-SAT instance. For each variable x which appears in more than three clauses (by 'appears' we mean that it or its complement is in the clause)perform the following procedure: Suppose x appears in k clauses. Create k new variables x ,, ... xk and replace the ith occurrence of x with xi, i = 1, . . . , k. Append the clause {x,vx~+,} for i=l,...,k-1 and the clause {Xkvn,}.

In the new instance, the clause {XiVZ;+l} implies that if xi is false, xi+, must be false as well. The cyclic structure of the clauses therefore forces the xi to be either all true or all false, so the new instance is satisfiable if the original one is. Moreover the transformation requires polynomial time. We have proved:

Theorem 2.1. Boolean satisfiability is NP-complete when restricted to instances with 2 or 3 variables per clause and at most 3 occurrences per variable.

An amusing corollary is:

Corollary 2.2. For any sr3, either

(i) every Boolean expression with exactly 3 variables per clause and no more than s occurrences per variable, is satisfiable, or

(ii) 3,s-SAT is NP-complete.

Proof. Suppose (i) does not hold, so an unsatisfiable expression in the variables. x9 _Y, 2, . . . exists. Without loss of generality, the first clause of the expression includes an 'x', uncomplemented. Let B denote the rest of the expression; clearly we may assume that B is satisfiable. B now has the properties, that x appears at most s- 1 times, and that it can only be satisfied when x is false.

Now consider an arbitrary 3-SAT instance and perform the procedure in the construction from Theorem 2.1. For the ith clause, (aV6) containing two variables,append an ith copy of B using variables Xi, y. I7 Z, ,,"', and change the clause to (aV6vx;). So if (i) is false, then (ii) is true. Note that (i) and (ii) cannot both be true unless P=NP.

Theorem 2.3. 3,4-St\T is NP-complete.

Proof. The only thing lacking in the construction from Theorem 2.1 is that the clauses (XiVZi+,) contain only two variables. For each such clause, introduce a new variable _Yi, so that the clause becomes (xiVZj+, VJ~). Now note that we can force each yj to be true by means of the clauses below in which yi appears only three times. The construction is, we suspect, as small as possible. The first three clauses require y; to be true if any of the pairs aj, bj are both false; the other ten clauses force this to happen:

{YVajVbj}, i= 1, ... 3,

{djVDjV6j), {djVOjVbj), {djVajV6"}, j=1,...,3,

G%V&V41.

7.Discuss polynomial time and reduction NOV'12

Reductions Polynomial-Time Reductions Classify Problems According to Computational Requirements

Q. Which problems will we be able to solve in practice?

A working definition. [von Neumann 1953, Godel 1956, Cobham 1964, Edmonds 1965, Rabin 1966]

Those with polynomial-time algorithms.

Yes	Probablyno
Shortest path	Longest path
Min cut	Max cut
2-SAT	3-SAT
Matching	3D-matching
Primality testing	Factoring
Planar 4-color	Planar 3-color
Bipartite vertex cover	Vertex cover4

Classify Problems

Desiderata. Classify problems according to those that can be solved in polynomial-time and those that cannot.

Provably requires exponential-time.

- Given a Turing machine, does it halt in at most k steps?
- Given a board position in an n-by-n generalization of chess, can black guarantee a win?

Frustrating news. Huge number of fundamental problems have defied classification for decades.

Today and Wed. Show that these fundamental problems are "computationally equivalent" and appear to be different manifestations of one really hard problem.5 Polynomial-Time Reduction Reduction. Problem X polynomial-time reduces to problem Y if arbitrary instances of problem X can be solved using:

- Polynomial number of standard computational steps, plus
- Polynomial number of calls to oracle that solves problem Y.

Notation. X P Y.

Remarks.

- We pay for time to write down instances sent to black box
instances of Y must be of polynomial size.
- Note: Cook reducibility (vs. Karp reducibility)

Means we can solve X in polynomial time IF we can solve Y in polynomial time!6

Polynomial-Time Reduction Purpose. Classify problems according to relative difficulty.

Design algorithms. If X $\rightarrow$P Y and Y can be solved in polynomial-time,then X can also be solved in polynomial time.Establish intractability. If X $\rightarrow$P Y and X cannot be solved in

polynomial-time, then Y cannot be solved in polynomial time. Establish equivalence. If X → P Y and Y → P X, we use notation X → P Y.Reduction By Simple Equivalence

Basic reduction strategies.

- Reduction by simple equivalence.
- Reduction from special case to general case.
- Reduction by encoding with gadgets.8

Independent Set

INDEPENDENT SET: Given a graph G = (V, E) and an integer k, is there a subset of vertices S → V such that |S| ≥ k, and for each edge at most one of its endpoints is in S?

Ex. Is there an independent set of size ≥ 6? Yes.

Ex. Is there an independent set of size ≥ 7? No.

independent set9

Vertex Cover

VERTEX COVER: Given a graph G = (V, E) and an integer k, is there a subset of vertices S ⊆ V such that |S| ≤ k, and for each edge, at least one of its endpoints is in S?

Ex. Is there a vertex cover of size ≤ 4? Yes.

Ex. Is there a vertex cover of size ≤ 3? No.

vertex cover10

Vertex Cover and Independent Set

Claim. VERTEX-COVER ≤P INDEPENDENT-SET.

Pf. We show S is an independent set iff V − S is a vertex cover.vertex cover independent set Reduction from Special Case to General Case Basic reduction strategies.

- Reduction by simple equivalence.
- Reduction from special case to general case.
- Reduction by encoding with gadgets.13

Set Cover

SET COVER: Given a set U of elements, a collection S1, S2, . . . , Sm of

subsets of U, and an integer k, does there exist a collection of ≤ k of these sets whose union is equal to U?

Sample application.

- m available pieces of software.
- Set U of n capabilities that we would like our system to have.
- The ith piece of software provides the set Si ⊆ U of capabilities.
- Goal: achieve all n capabilities using fewest pieces of software.

Ex:

U = { 1, 2, 3, 4, 5, 6, 7 }

k = 2

S1 = {3, 7} S4 = {2, 4}

S2 = {3, 4, 5, 6} S5 = {5}

S3 = {1} S6 = {1, 2, 6, 7}14

SET COVER

U = { 1, 2, 3, 4, 5, 6, 7 }

k = 2

Sa = {3, 7} Sb = {2, 4}

Sc = {3, 4, 5, 6} Sd = {5}

Se = {1} Sf= {1, 2, 6, 7}

8.**Describe Kruskal's algorithm APR' 2014**

Theorem: Kruskal's algorithm finds a minimum spanning tree.

Proof: Let G = (V, E) be a weighted, connected graph. Let T be the edge set that is grown in Kruskal's algorithm. The proof is by mathematical induction on the number of edges in T.

- We show that if T is promising at any stage of the algorithm, then it is still promising when a new edge is added to it in Kruskal's algorithm
- When the algorithm terminates, it will happen that T gives a solution to the problem and hence an MST.

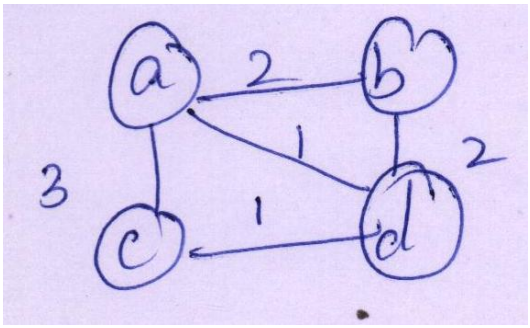
Basis: $T = \phi$ is promising since a weighted connected graph always has at least one MST.

Induction Step: Let T be promising just before adding a new edge $e = (u, v)$. The edges T divide the nodes of G into one or more connected components. u and v will be in two different components. Let U be the set of nodes in the component that includes u. Note that

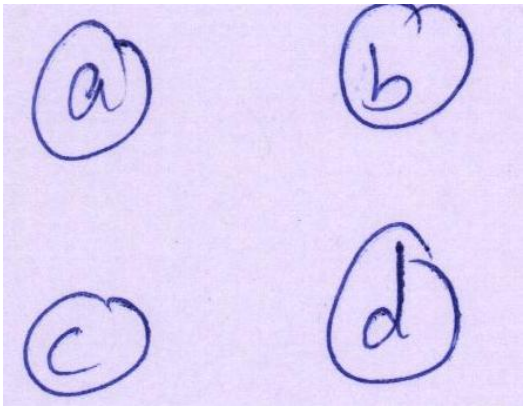
- U is a strict subset of V
- T is a promising set of edges such that no edge in T leaves U (since an edge T either has both ends in U or has neither end in U)

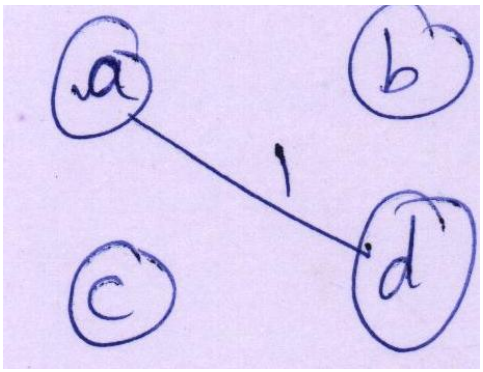
- e is a least cost edge that leaves U (since Kruskal's algorithm, being greedy, would have chosen eonly after examining edges shorter than e)
    - Kruskal algorithm ueses a greedy technique to compute a minimum spanning tree. This algorithm selects the edges in the order of smallest weight and accept an edge if it does not cause a cycle.The algorithm terminates if enough edges are accepted.
    -

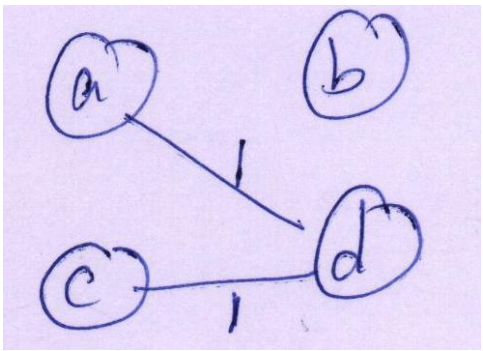
ALGORITHM

- Void Kruskal(graph G)
- {
- int EdgesAccepted=0;
- while(EdgesAccepted<Numvertx -1)
- {
- USet=Find(U,S);
- VSet=Find(V,S);
- if(USet!=VSet)
- {
- EdgesAccepted++;
- SetUnion(S,Uset,VSet);
- }
- }
- }
- ***EXAMPLE:***
- 

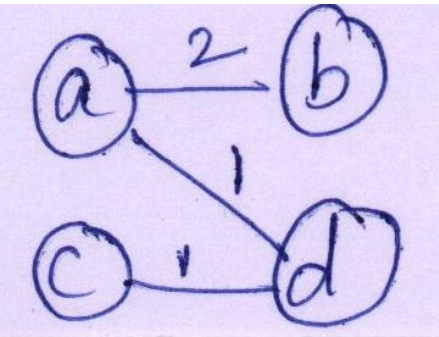
- **Step 1:** Initially each vertex is in its own set

- 

- **Step 2:** Now select the edge with minimum weight
- 

- Edge(a,b) is added to minimum spanning tree
- **Step 3:** Select the next edge , with minimum weight and check whether it forms a cycle. If it forms a cycle then that edge is rejected. else add the edge to minimum spanning tree.
- 

- (c,d) is next minimum weighted edge.
- **Step 4:** Repeat stpe 3 until all vertices included in minimum spanning tree.

- 

- The minimum cost of spanning tree is 4 (2 + 1 + 1 = 4)
-

Edge	Weight	Action
(a,d)	1	Accepted
(c,d)	1	Accepted
(a,b)	2	Accepted
(b,d)	2	Rejected
(a,c)	3	Rejected

-

Pondicherry University Questions

2 Marks

1. Define time complexity of Turing Machines **Nov'11,May '2012, Dec 2013 (Q.No.6)**
2. Define NP Hardness. **Nov'11 (Q.No.20)**
3. Specify any two complexity classes **May '2012 (Q.No.21)**
4. What are P classes **APR'13 (Q.No.23)**
5. State Travelling Salesperson problem? **APR'13. (Q.No.24)**
6. What is NP completeness? **Nov'13,MAY'15 (Q.No.22)**
7. What is DFA With NFA? **APR 2014 (Q.No.25)**
8. When will you call an expression is satisfiable? **Dec 2013(Q.No.27)**
9. What are NP classes? **APR'2014,NOV'15 (Q.No.26)**
10. State satisfiability problem?**APR'2014 (Q.No.10)**
11. What is meant by Space complexity? **(NOV'14) (Q.No.1)**
12. Differentiate between space and time complexity? **(NOV'15) (Q.No.7)**

11 Marks

1. Define and Explain Complexity classes **APR'11 (Q.No.10)**

2. Discuss about NP complete problem. **APR'11, APR'13, APR' 2014 ,MAY'15,NOV'15,APR'16 (Q.No.4)**
3. Discuss on restricted satisfiability problem**NOV'12 (Q.No.7)**
4. Discuss polynomial time and reduction **NOV'12 (Q.No.8)**
5. Explain briefly about time complexity and space complexity **NOV'2011,APR'16(Q.No.1)**
6. Neat diagram about vertex cover in NP Completeness problem **NOV'2011 ,NOV'14(Q.No.4)**
7. Briefly explain about NP hard problem.**NOV 2013 ,NOV'14 ,MAY'15(Q.No.3)**
8. Describe Kruskal's algorithm **APR' 2014 (Q.No.10)**
9. Discuss the Complexity classes of Turning machines? **(NOV'15) (Q.No.2)**