

## Theory of Computer Science

Q1. (a) Explain different types of machines and state at least one application of each

10

Ans: Deterministic Finite State Automata (DFA)

One-way, infinite tape, broken into cells

One-way, read-only tape head.

Finite control, i.e., a program, containing the position of the read head, current symbol being scanned, and the current "state."

A string is placed on the tape, read head is positioned at the left end, and the DFA will read the string one symbol at a time until all symbols have been read. The DFA will then either accept or reject.

A DFA is a five-tuple:  $M = (Q, \Sigma, \delta, q_0, F)$

$Q$  A finite set of states

$\Sigma$  A finite input alphabet

$q_0$  The initial/starting state,  $q_0$  is in  $Q$

$F$  A set of final/accepting states, which is a subset of  $Q$

$\delta$  A transition function, which is a total function from  $Q \times \Sigma$  to  $Q$

$\delta: (Q \times \Sigma) \rightarrow Q$   $\delta$  is defined for any  $q$  in  $Q$  and  $s$  in  $\Sigma$ , and  $\delta(q,s) = q'$  is equal to another state  $q'$  in  $Q$ .

Intuitively,  $\delta(q,s)$  is the state entered by  $M$  after reading symbol  $s$  while in state  $q$ .

Nondeterministic Finite State Automata (NFA)

An NFA is a five-tuple:  $M = (Q, \Sigma, \delta, q_0, F)$

$Q$  A finite set of states

$\Sigma$  A finite input alphabet

$q_0$  The initial/starting state,  $q_0$  is in  $Q$

$F$  A set of final/accepting states, which is a subset of  $Q$

$\delta$  A transition function, which is a total function from  $Q \times \Sigma$  to  $2^Q$

$\delta: (Q \times \Sigma) \rightarrow 2^Q$   $2^Q$  is the power set of  $Q$ , the set of all subsets of  $Q$   $\delta(q,s)$ -The set of

all states  $p$  such that there is a transition labelled  $s$  from  $q$  to  $p$   $\delta(q,s)$  is a function from  $Q \times \Sigma$  to  $2^Q$

(but not to  $Q$ )

Formal Definition of a PDA

A pushdown automaton (PDA) is a seven-tuple:  $M = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$

$Q$  finite set of states

## Theory of Computer Science

$\Sigma$  finite input alphabet

finite stack alphabet

$\Gamma$

$q_0$  The initial/starting state,  $q_0$  is in  $Q$

$z_0$  A starting stack symbol, is in  $\Gamma$

$F$  A set of final/accepting states, which is a subset of  $Q$

$\delta$  A transition function, where

$\delta: Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma \rightarrow$  finite subsets of  $Q \times \Gamma^*$

Consider the various parts of  $\delta$ :

$Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma \rightarrow$  finite subsets of  $Q \times \Gamma^*$

$Q$  on the LHS means that at each step in a computation, a PDA must consider its' current state.

$\Gamma$  on the LHS means that at each step in a computation, a PDA must consider the symbol on top of its' stack.  $\Sigma \cup \{\epsilon\}$  on the LHS means that at each step in a computation, a PDA may or may not consider the current input symbol, i.e., it may have epsilon transitions. "Finite subsets" on the RHS means that at each step in a computation, a PDA will have several options.  $Q$  on the RHS means that each option specifies a new state.  $\Gamma^*$  on the RHS means that each option specifies zero or more stack symbols that will replace the top stack symbol.

Deterministic Turing Machine (DTM)

TMs model the computing capability of a general purpose computer, which informally can be described as: Effective procedure, Finitely describable, Well defined, discrete, "mechanical" steps, Always terminates. Computable function: A function computable by an effective procedure

Two-way, infinite tape, broken into cells, each containing one symbol. Two-way, read/write tape head. Finite control, i.e., a program, containing the position of the read head, current symbol being scanned, and the current state. An input string is placed on the tape, padded to the left and right infinitely with blanks, read/write head is positioned at the left end of input string. In one move, depending on the current state and the current symbol being scanned, the TM 1) changes state, 2) prints a symbol over the cell being scanned, and 3) moves its' tape head one cell left or right. Many modifications possible.

Formal Definition of a DTM

A DTM is a seven-tuple:  $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$

$Q$  A finite set of states

$\Gamma$  A finite tape alphabet

$B$  A distinguished blank symbol, which is in  $\Gamma$

$\Sigma$  A finite input alphabet, which is a subset of  $\Gamma - \{B\}$

$q_0$  The initial/starting state,  $q_0$  is in  $Q$

$F$  A set of final/accepting states, which is a subset of  $Q$

$\delta$  A next-move function, which is a *mapping* (i.e., may be undefined)

from  $Q \times \Gamma \rightarrow Q \times \Gamma \times \{L,R\}$

Theory of Computer Science

Intuitively,  $\delta(q,s)$  specifies the next state, symbol to be written, and the direction of tape head movement by M after reading symbol s while in state q.

Moore Machine

Mealy Machine

Applications of each machine

(b) Let G be the grammar. Find the leftmost derivation, rightmost derivation and parse tree for the string 00110101

10

G:  $S \rightarrow 0B \mid 1A$

$A \rightarrow 0 \mid 0S \mid 1AA$

$B \rightarrow 1 \mid 1S \mid 0BB$

Ans: Leftmost derivation:

$S \Rightarrow 0B$

$\Rightarrow 00BB$

$\Rightarrow 001SB$

$\Rightarrow 0011AB$

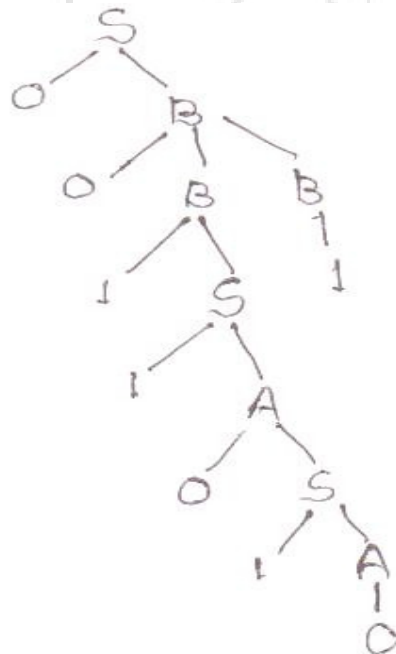
$\Rightarrow 00110SB$

$\Rightarrow 001101AB$

$\Rightarrow 0011010B$

$\Rightarrow 00110101$

Leftmost derivation tree

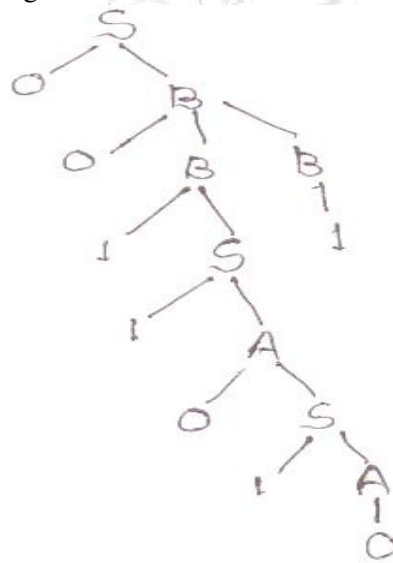


Theory of Computer Science

Rightmost derivation:

- S ⇒ 0B
- ⇒ 00BB
- ⇒ 00B1
- ⇒ 001S1
- ⇒ 0011A1
- ⇒ 00110S1
- ⇒ 001101A1
- ⇒ 00110101

Rightmost derivation tree

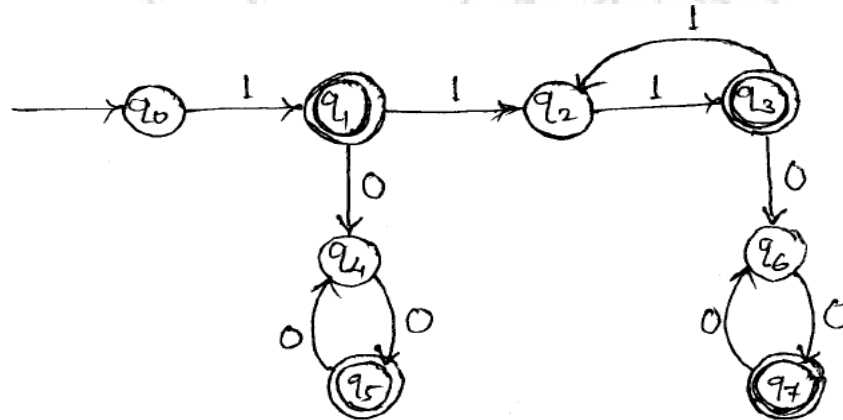


Q2. (a) Design a DFA to accept

(i) a set of all strings with odd number of ones followed by even number of zeros

05

Ans:

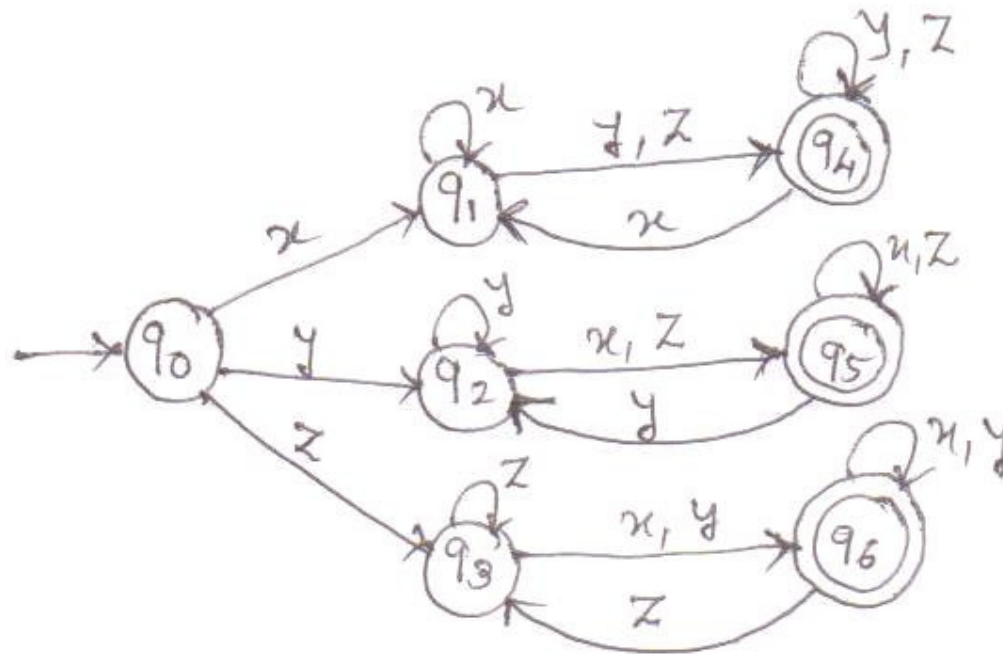


## Theory of Computer Science

(ii) a set of all strings with which begin and end with different letters  $\Sigma = \{x, y, z\}$ 

05

Ans:



(b) What is a regular expression? Give formal definition of a regular expression. Design a DFA corresponding to the regular expression  $(a+b)^*aba(a+b)^*$

10

Ans : **Regular Expression:** A regular expression is used to specify a language, and it does so precisely.

Regular expressions are very intuitive. Regular expressions are very useful in a variety of contexts.

Given a regular expression, an NFA- $\epsilon$  can be constructed from it automatically.

Thus, so can an NFA, a DFA, and a corresponding program, all automatically!

Definition of a Regular Expression

Let  $\Sigma$  be an alphabet. The regular expressions over  $\Sigma$  are:

$\emptyset$  Represents the empty set  $\{ \}$

$\epsilon$  Represents the set  $\{ \epsilon \}$

$a$  Represents the set  $\{ a \}$ , for any symbol  $a$  in  $\Sigma$

Let  $r$  and  $s$  be regular expressions that represent the sets  $R$  and  $S$ , respectively.

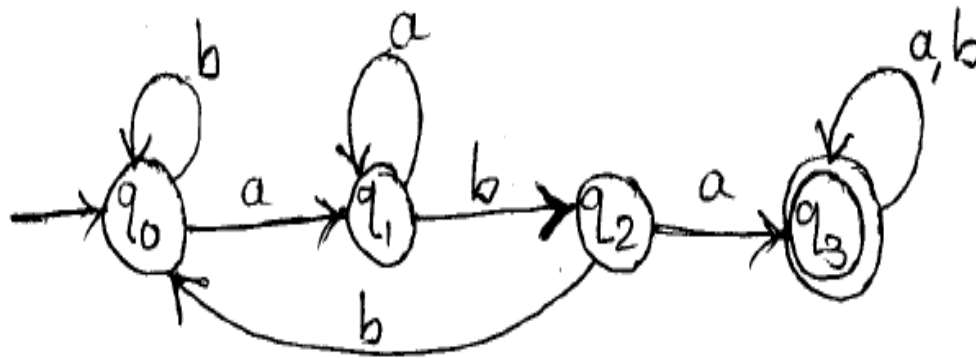
$r+s$  Represents the set  $R \cup S$  (precedence 3)

$rs$  Represents the set  $RS$  (precedence 2)

$r^*$  Represents the set  $R^*$  (highest precedence)

$(r)$  Represents the set  $R$  (not an op, provides precedence)

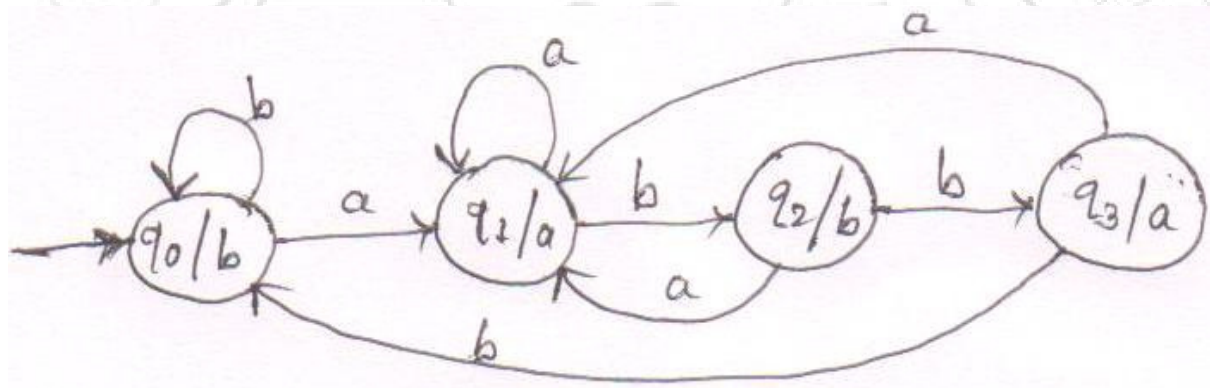
If  $r$  is a regular expression, then  $L(r)$  is used to denote the corresponding language.



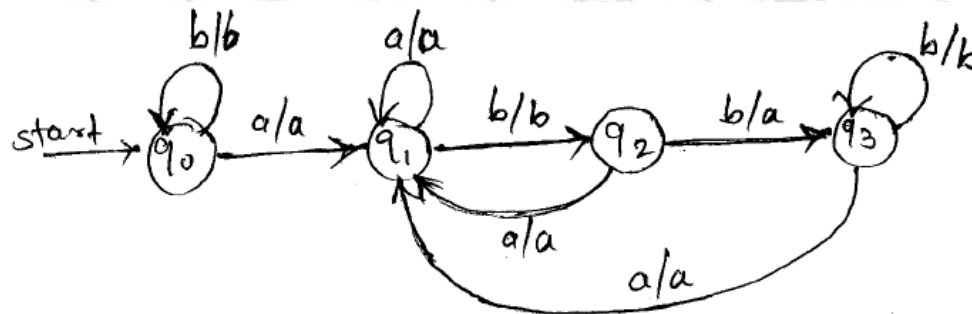
Q3. (a) Design a Moore and Mealy machine to convert each occurrence of a substring *abb* by *aba*

10

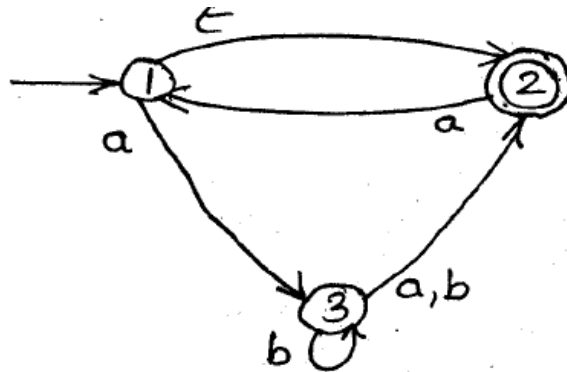
Ans: Moore Machine:



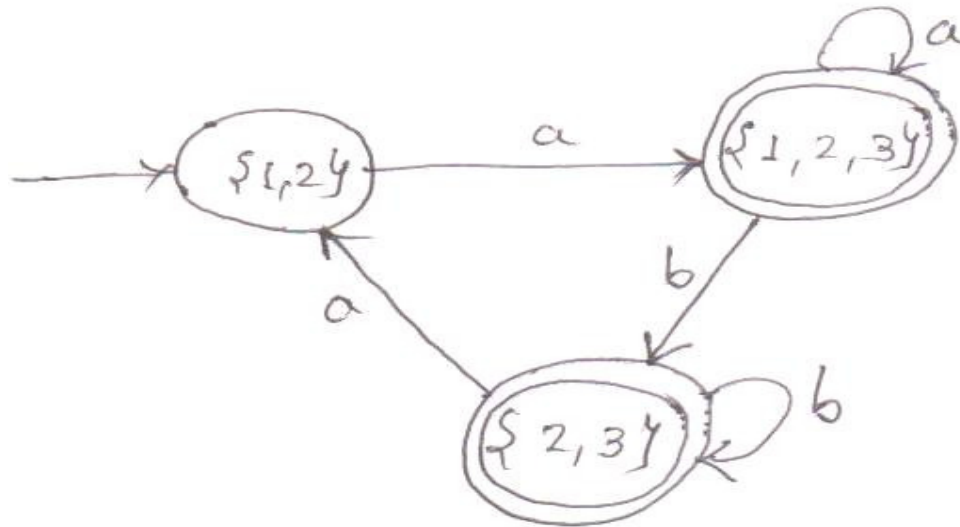
Mealy Machine:



(b) Convert the following NFA with epsilon moves to an NFA without epsilon moves and then to a DFA. 10



Ans:  $\epsilon$ -Closure(1) = {1,2}  
 $\delta((1,2), a) = \delta(1,a) \cup \delta(2,a) = \{1, 3\}$   
 $\epsilon$ -Closure(1,3) = {1,2,3}  
 $\delta((1,2), b) = \delta(1,b) \cup \delta(2,b) = \{\}$   
 $\delta((1,2,3), a) = \delta(1,a) \cup \delta(2,a) \cup \delta(3,a) = \{1, 2,3\}$   
 $\epsilon$ -Closure(1,2,3) = {1,2,3}  
 $\delta((1,2,3), b) = \delta(1,b) \cup \delta(2,b) \cup \delta(3,b) = \{2,3\}$   
 $\epsilon$ -Closure(2,3) = {2,3}  
 $\delta((2,3), a) = \delta(2,a) \cup \delta(3,a) = \{1, 2\}$   
 $\epsilon$ -Closure(1,2) = {1,2}  
 $\delta((2,3), b) = \delta(2,b) \cup \delta(3,b) = \{2,3\}$   
 $\epsilon$ -Closure(2,3) = {2,3}



Q4. (a) Using pumping lemma prove that the following languages are not regular

10

(i)  $L1 = \{ww \mid w \in \{0,1\}^*\}$

Ans.: -

Let  $n$  be constant of pumping lemma.

Select  $Z = 0^n 1 \mid 0^n 1$ , this ensures that  $Z$  is in  $L$ , and  $|Z| \geq n$ .

If we write  $Z = uvw$ , then the possible choices of  $v$  satisfying the conditions:

$1 \leq |v| \leq n$  and  $|uv| \leq n$  are:

$v = 0^p$ , where  $1 \leq p \leq n$ , i.e.,  $v$  can be chosen to be string of minimum one zero and maximum  $n$  0's.

For this choice of  $v$ ,  $uv^0w$  will be  $0^{n-p} 10^n 1$ , and since  $1 \leq p \leq n$ ,  $uv^0w$  will not be of the form  $ww$ , hence it cannot belong to  $L$ .

$v$  cannot contain 1's, because if  $v$  is chosen to contain 1's, then  $|uv|$  will not be less than or equal to  $n$ . Thereby proving that  $L$  is not regular.

Therefore, we get contradiction to pumping lemma. Thereby proving that  $L$  is not regular.

(ii)  $L2 = \{0^i 1^i \mid i \geq 1\}$

Ans. : Assume that  $L = \{0^i 1^i \mid i \geq 1\}$  is regular.

Let,  $w = 0^n 1^n$  such that  $|w| = 2n$ . By pumping lemma we can write

$w = xyz$  such that  $|xy| \leq n$  and  $|y| \neq 0$ .

Now if  $xy^i z \in L$  then the language  $L$  is said to be regular.

There are many cases - i)  $y$  has only 0's ii)  $y$  has only 1's iii)  $y$  has both 0's and 1's.

i) If  $y$  has only 0's then the string

$w = 0^{n-k} 1^n = xz$  since  $y = 0^k$  and  $i = 0$

Surely  $n - k \neq n$ . Hence  $xz \notin L$ .

Hence our assumption of being  $L$  regular is wrong.

ii) If  $y$  has only 1's then, for  $i = 0$  and  $y = 1^k$

$\therefore w = xz = 0^n 1^{n-k}$

As  $n \neq n - k$ ,  $xz = w \notin L$

Again  $L$  is not regular.

iii) If  $y$  has 0's and 1's then

$w = 0^{n-k} 0^k 1^j 1^{n-i} = xy^i z$

If  $i = 2$  then

$w = 0^{n-k} 0^{2k} 1^{2j} 1^{n-j} \notin L$

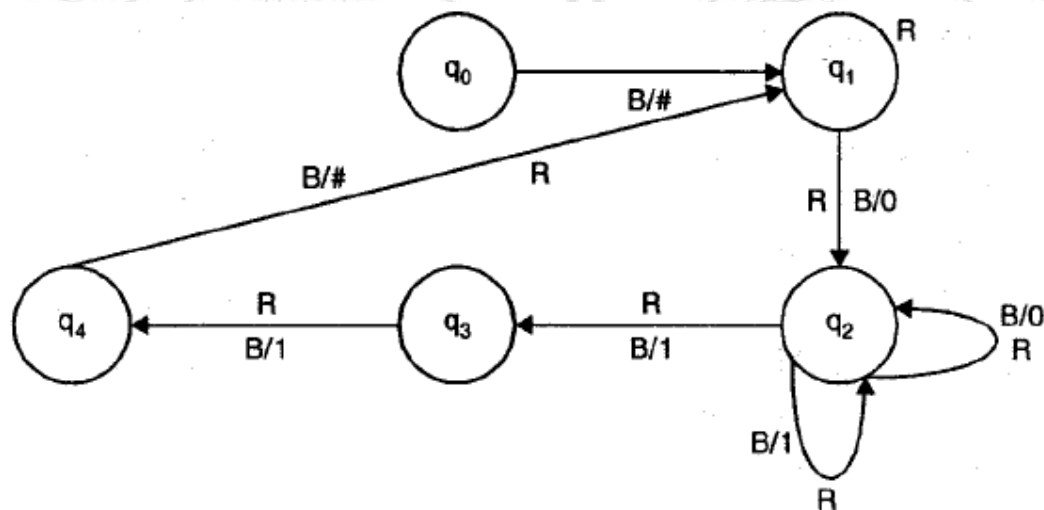
Hence from all these 3 cases it is clear that language  $L$  is not regular.

(b) Design a Turing machine to generate the language given by a regular expression  $0(0+1)^*11$  10

Ans:

1. It starts in the initial with blank tape, and replaces the blank symbol scanned by tape head with #, and enter into a state  $q_1$ , and moves tape head right.
2. In  $q_1$  it replaces the next blank by 0, enters into state  $q_2$ , and moves the tape head right.
3. In state  $q_2$ , it either replaces the next blank by 1 or 0, and remains in state  $q_2$ , or replaces next blank by 1, and enters into state  $q_3$ , in both cases the tape head is moved right.
4. In state  $q_3$ , it replaces the next blank by 1, and enters into state  $q_4$ , moves tape head right.
5. In state  $q_4$ , it replaces the next blank by #, and enters into state  $q_1$ .

The above Turing machine, therefore, prints those strings that starts with 0, and ends with 11. Hence, the Turing machine generates the language given by the regular expression :  $0(0|1)^*11$ .



Q5. (a) List and explain decision properties of regular languages. Explain the test for checking emptiness of a regular language 10

Ans: **Decision Properties:**

A *decision property* for a class of languages is an algorithm that takes a formal description of a language (e.g., a DFA) and tells whether or not some property holds.

Example: Is language L empty?

You might imagine that the language is described informally, so if my description is “the empty language” then yes, otherwise no.

But the representation is a DFA (or a RE that you will convert to a DFA).

Can you tell if  $L(A) = \Phi$  for DFA A?

When we talked about protocols represented as DFA's, we noted that important properties of a good protocol were related to the language of the DFA.

Example : “Does the protocol terminate?” = “Is the language finite?”

## Theory of Computer Science

Example : “Can the protocol fail?” = “Is the language nonempty?”

We might want a “smallest” representation for a language, e.g., a minimum-state DFA or a shortest RE.

If you can’t decide “Are these two languages the same?”

I.e., do two DFA’s define the same language?

You can’t find a “smallest.”

Membership:

Our first decision property is the question: “is string  $w$  in regular language  $L$ ?”

Assume  $L$  is represented by a DFA  $A$ .

Simulate the action of  $A$  on the sequence of input symbols forming  $w$ .

Emptiness:

Given a regular language, does the language contain any string at all.

Assume representation is DFA.

Construct the transition graph.

Compute the set of states reachable from the start state.

If any final state is reachable, then yes, else no.

Infiniteness:

Is a given regular language infinite?

Start with a DFA for the language.

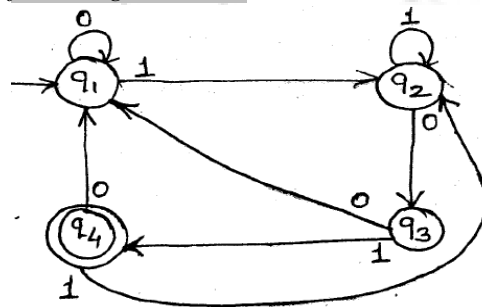
Key idea: if the DFA has  $n$  states, and the language contains any string of length  $n$  or more, then the language is infinite.

Otherwise, the language is surely finite.

Limited to strings of length  $n$  or less.

(b) State Arden’s theorem and use it to construct a regular expression corresponding to the following automata

10



Ans:  $(0+1(1+011)^*(00+010))^*1(1+011)^*01$

Q6 (a) (i) Convert the following CFG to CNF

05

$S \rightarrow bA \mid aB$

## Theory of Computer Science

$$A \rightarrow bAA \mid aS \mid a$$

$$B \rightarrow aBB \mid bS \mid b$$

Ans:

$$S \rightarrow BA \mid AB$$

$$A \rightarrow BC \mid AS \mid a$$

$$B \rightarrow AD \mid BS \mid b$$

$$C \rightarrow AA$$

$$D \rightarrow BB$$

(ii) Construct a PDA accepting the following language  $L = \{a^n b^m a^n \mid m, n \geq 1\}$

05

Ans:

$$\delta(q_0, a, Z) = (q_0, aZ)$$

$$\delta(q_0, a, a) = (q_0, aa)$$

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

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

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

$$\delta(q_2, a, a) = (q_2, \epsilon)$$

$$\delta(q_2, \epsilon, Z) = (q_3, \epsilon)$$

(b) Explain the rules for simplification of a context free grammar

10

Ans: **Simplification of CFGs**

As we have seen various languages can effectively be represented by context tree. The grammars are not always optimized. That means grammar may consist of some extra symbols (non terminals).

Having extra symbols unnecessary

Simplification of grammar means reduction of grammar by removing useless symbols. The properties of reduced grammar are given below.

1. Each variable (i.e. non terminal) and each terminal of  $C$  appears in the derivation of some word in  $L$

2. There should not be any production as  $X \rightarrow Y$  where  $X$  and  $Y$  are non terminals.

3. if  $\epsilon$  is not in the language  $L$  then there need not be the production  $X \rightarrow \epsilon$

**Removal of Useless Symbols**

Any symbol is useful when it appears on the right hand side, in the product rule and generates some terminal string, if no such derivation exists then it supposed to be an useless symbol.

For example,  $G = (V, T, P, S)$  where  $V = (S, T, X)$ ,  $T = \{0, 1\}$

$$S \rightarrow 0T11T1X1011 \text{ rule 1}$$

$$T \rightarrow 00 \text{ rule 2}$$

Now, in the above CFG, the non terminals are  $S, T$  and  $X$ .

To derive some string we have to start from start symbol  $S$ .

## Theory of Computer Science

$0T S \rightarrow 0T$

$000 T \rightarrow 00$

Thus we can reach to certain string after following these rules.

But if  $S \rightarrow X$  then there is no further rule as a definition to  $X$ . That means there is no point in the rule  $S \rightarrow X$ . Hence we can declare that  $X$  is a useless symbol. And we can remove this so after removal of this useless symbol CFG becomes

$G=(V, T, P, S)$  where  $V=(S, T)$

$T=\{0,1\}$  and  $P=\{S \rightarrow 0T11T1011$

$T \rightarrow 00\}$

$S$  is a start symbol.

#### Elimination of $\epsilon$ Productions from Grammar

As we have seen in finite automata and regular expression that  $\epsilon$  or a null string indicates a string with no value. We have also seen that even though some NFA contains  $\epsilon$  moves we can convert that NFA without  $\epsilon$  moves. Even in context free grammar, if at all there is  $\epsilon$  production we can remove it, without changing the meaning of the grammar. Thus  $\epsilon$  productions are not necessary in a grammar.

For example,

If  $S \rightarrow 0S1S\epsilon$

Then we can remove  $\epsilon$  production. But we have to take a care of meaning of CFG. i.e. Meaning of CFG should not get changed if we place  $S \rightarrow \epsilon$  in other rules we get  $S \rightarrow 0$  when  $S \rightarrow 0S$  and  $S \rightarrow \epsilon$  as well as  $S \rightarrow 1$  when  $S \rightarrow 1S$  and  $S \rightarrow \epsilon$

Hence we can rewrite the rules as

$S \rightarrow 0S1S1011$

Thus  $\epsilon$  production is removed.

#### Removing Unit Productions

The unit productions are the productions in *which* one non terminal gives another non terminal.

For example if

$X \rightarrow Y$

$Y \rightarrow Z$

$Z \rightarrow X$

Then  $X$ ,  $Y$  and  $Z$  are unit productions. To optimize the grammar we need to remove the unit productions.

If  $A \rightarrow B$  is a unit production and  $B \rightarrow X_1 X_2 X_3 \dots X_n$  then while removing  $A \rightarrow B$  Production we should add a rule  $A \rightarrow X_1 X_2 X_3 \dots X_n$ .

Q7. Write short notes on (any three)

20

(a) Variants of a Turing Machine

## Theory of Computer Science

Ans: Other (Extended) TM Models:

One-way infinite tapes

Multiple tapes and tape heads

Non-Deterministic TMs

Multi-Dimensional TMs (n-dimensional tape)

Multi-Heads

Multiple tracks

All of these extensions are equivalent to the basic TM model

(b) **Post Correspondence Problem**

Ans:

In this section, we will discuss the undecidability of strings and not of Turing machines. The undecidability of strings is determined with the help of Post's Correspondence Problem (PCP). Let us define the PCP.

"The post's correspondence problem consists of two lists of strings that are of equal length over the input  $\Sigma$ . The two lists are  $A = w_1, w_2, w_3, \dots, w_n$  and  $B = x_1, x_2, x_3, \dots, x_n$  then there exists a non empty set of integers  $i_1, i_2, i_3, \dots, i_n$  such that  $w_{i_1} w_{i_2} w_{i_3} \dots w_{i_n} = x_{i_1} x_{i_2} x_{i_3} \dots x_{i_n}$ "

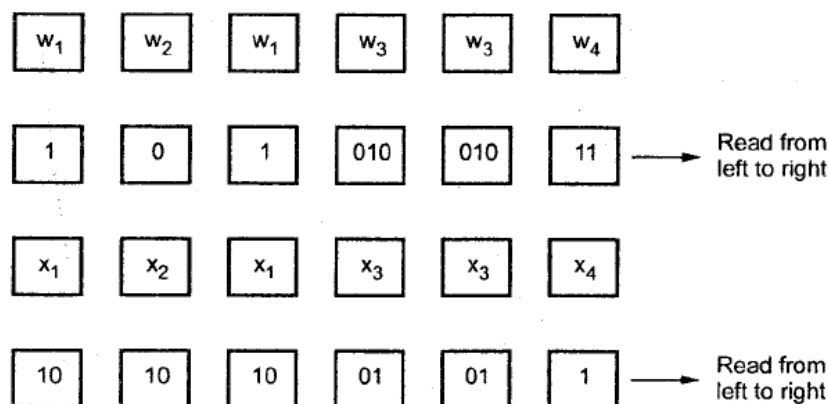
To solve the post correspondence problem we try all the combinations of  $i_1, i_2, i_3, \dots, i_n$  to find the  $w_i = x_i$  then we say that PCP has a solution.

Consider the correspondence system as given below -

$A = (1; 0; 010; 11)$  and  $B = (10; 10; 01; 1)$ . The input set is  $\Sigma = \{0, 1\}$ . Find the solution.

**Solution :** A solution is 1; 2; 1; 3; 3; 4. That means  $w_1 w_2 w_1 w_3 w_3 w_4 = x_1 x_2 x_1 x_3 x_3 x_4$ .

The constructed string from both lists is 10101001011.



## Theory of Computer Science

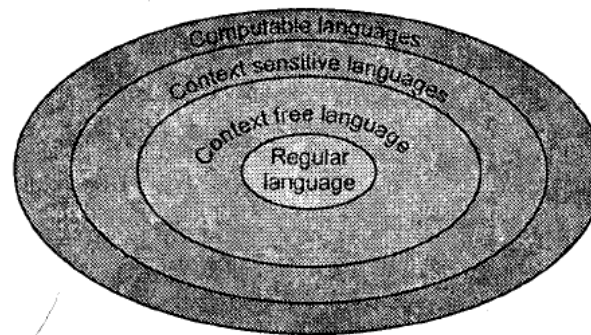
## (C) Chomsky Hierarchy

Ans:

The Chomsky's Hierarchy represents the class of languages that are accepted by different machine. The category of languages in Chomsky's Hierarchy is as given below -

Language class	Language	Grammar	Machine	One example
Type 3	Regular	Regular grammar	FSM i.e. NFA or DFA	$a^*b^*$
Type 2	Context free	Context free grammar	PDA	$a^n b^n$
Type 1	Decidable languages	Context sensitive grammar	Linear bounded automata	$a^n b^n c^n$
Type 0	Computable languages	Unrestricted grammar	Turing machine	$n!$

This is a hierarchy therefore every language of type 3 is also of type 2, 1 and 0. Similarly every language of type 2 is also of type 1 and 0 etc.



## (d) Intractable Problems

Ans:

**Introduction**

So far now we have discussed, whether the problems are solvable or unsolvable. The class of solvable problems is known as the decidable problems, and unsolvable problems are known as undecidable problems. In this chapter we will mainly focus decidable problems. But now we will discuss them from their efficiency point of view. That means there are some decidable problems that could be solved in some measurable amount of time 'but there are other problems which can not be solved in some fixed amount of time; rather only small instances of that problem are solvable within tolerable amount of time. Hence the term intractability has come up. Intractability is a technique of showing whether the problem could be solved within polynomial time. This has lead to the two classes of solving the problem. These are P and NP class problem. Throughout this chapter we will discuss about P and NP class problems, examples and their properties.

## Theory of Computer Science

### The Classes P and NP

There are two groups in which a problem can be classified. The first group consists of the problems that can be solved in polynomial time. For example: searching of an element from the list  $O(\log n)$ , sorting of elements  $O(\log n)$ .

The second group consists of problems that can be solved in non-deterministic polynomial time. For example: Knapsack problem  $O(2^{n^2})$  and Travelling Salesperson problem ( $O(n^{2.2})$ ).

### Intractable Problems

Any problem for which answer is either yes or no is called decision problem. The algorithm for decision problem is called decision algorithm.

- Any problem that involves the identification of optimal cost (minimum or maximum) is called optimization problem. The algorithm for optimization problem is called optimization algorithm.

- Definition of P - Problems that can be solved in polynomial time. ("P" stands for polynomial).

Examples - Searching of key element, Sorting of elements, All pair shortest path.

- Definition of NP - It stands for "non-deterministic polynomial time". Note that NP does not stand for "non-polynomial".

Examples - Travelling Salesperson problem, Graph coloring problem, Knapsack problem, Hamiltonian circuit problems.

- The NP class problems can be further categorized into NP-complete and NP hard problems.

- A problem D is called NP-complete if -

- i) It belongs to class NP

- ii) Every problem in NP can also be solved in polynomial time.

- If an NP-hard problem can be solved in polynomial time then all NP-complete problems can also be solved in polynomial time.

- All NP-complete problems are NP-hard but all NP-hard problems can not be NP-complete.

- The NP class problems are the decision problems that can be solved by non-deterministic polynomial algorithms.

(e) *Recursive and recursively enumerable languages*

Ans:

**Recursively enumerable languages :** A language L is called recursively enumerable if it is accepted by some Turing machine M that accepts every word in L and either rejects or loops for every word in the complement of L..

**Recursive language :** A language is said to be recursive if there exists a Turing machine that accepts every string of the language and every string is rejected if it is not belonging to that language.

- i) The universal language is recursively enumerable. Similarly union of two recursively enumerable languages is recursively enumerable.

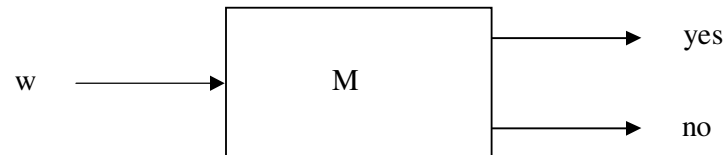
- ii) The Language. i.e. diagonalization language is not recursively enumerable. .

- iii) The union of two recursive languages is a recursive language. The complement of recursive language is recursive.

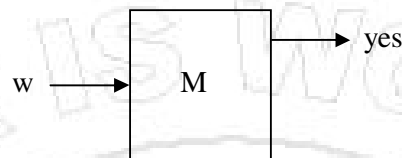
TM Block Diagrams:

If L is a recursive language, then a TM M that accepts L and always halts can be pictorially represented by a "chip" that has one input and two outputs.

## Theory of Computer Science



If  $L$  is a recursively enumerable language, then a TM  $M$  that accepts  $L$  can be pictorially represented by a “chip” that has one output.



Conceivably,  $M$  could be provided with an output for “no,” but this output cannot be counted on. Consequently, we simply ignore it.

**References:**

- 1) John E. Hopcroft, Rajeev Motwani, Jeffrey D. Ullman, “Introduction to Automata Theory, Languages and Computations”.
- 2) J. C. Martin, “Introduction to Languages and the Theory of Computation”
- 3) Michael Sipser, “Theory of Computation”.