

B.E.

Fourth Semester Examination, Dec-2005

THEORY OF AUTOMATION AND COMPUTATION

Note: Attempt any five questions.

Q. 1. (a) Give the definitions of:

- (i) Deterministic Finite Automata
- (ii) Non Deterministic Finite Automata
- (iii) Regular expressions

Also prove NDFA = DFA

Ans. A deterministic automata is one in which each move is uniquely determined by the current configuration. If internal state, the input and the contents of temporary storage are known then future can be predicted easily and exactly. On the other hand, a non-deterministic automation may have several possible moves, therefore we can predicted a set of possible solutions.

Definition: A deterministic finite automata (DFA) is defined by Quintuple

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

where,

- 1. Q is a finite set of internal state.
- 2. Σ is a finite set of symbol (input alphabets).
- 3. $\delta: Q \times \Sigma \rightarrow Q$ is transition function.
- 4. $q_0 \in Q$ is the initial state.
- 5. $F \subseteq Q$ is the final state.

(ii) Non Deterministic Finite Automata.

Ans. Definition: Nondeterministic a means of choice of an automation rather than a unique move in each situation we allow a set of possible moves.

Nondeterministic automata $M = (Q, \Sigma, \delta, q_0, F)$ is a collection of following five things

- 1. Q is a finite set of states
- 2. It has an initial state or start state, i.e., $q_0 \in Q$.
- 3. It has some (may be none) final states, i.e., $F \subseteq Q$.
- 4. An alphabet Σ of possible input letter.
- 5. A transition function $\delta: Q \times \Sigma \rightarrow 2^Q$, 2^Q is the power set of Q .

(iii) Regular Expression :

Ans. Regular expressions are useful for representing certain sets of strings in an algebraic fashion. A language is regular if there exists a finite automata for it. Therefore, every regular language can be described by some DFA or some NDFA such a description can be very useful for example if we want to show the logic by which we decide if a given string is in a certain language. One way of describing regular language is via notation of regular expression. This notation involves a combination of strings of symbols from some alphabet Σ , parantheses and the operators $+$, $.$ and $*$.

Definition : Recursive definition of regular expression over Σ follows the following rules:

1. Every letter of Σ can be made into regular expression by writing it in bold face, \wedge itself is a regular expression.
2. If R_1 and R_2 are regular expression then
 - (i) (R_1) is a regular expression
 - (ii) $R_1 . R_2$ is also a regular expression
 - (iii) $R_1 + R_2$ is also a regular expression
 - (iv) R^* is also a regular expression.

Also Prove NDFA=DFA :

Proof : Let $M = (Q, \Sigma, \delta, q_0, F)$ be NDFA accepts L we can construct a DFA M' as $M' = (Q', \Sigma, \delta', q_0', F')$ where

- (i) $Q' = 2^Q$ any state in Q' is denoted by $[q_1, q_2, \dots, q_k]$ where $q_1, q_2, \dots, q_k \in Q$
- (ii) $q_0' = [q_0]$
- (iii) F' is the set of all subsets of Q containing an element of F .

Construction of Q , q_0' and F' : M is initially at q_0 . But by applying any input symbol say a , M can reach any of the states in $\delta(q_0, a)$. Just after the application of the input symbol a we require all the possible state that M can reach after the application of a . Hence M' has to remember all these possible states at any time. Hence the state of M' are defined as subsets of Q .

As M starts with start state q_0 , q_0' defined as $[q_0]$. A string w belongs to $T(M)$. If a final state is one of the possible state M reaches on processing w . So a final state in M' , i.e., an element of F'

$$(iv) \quad \delta'([q_1, q_2, \dots, q_k], a) = \delta(q_1, a) \cup \delta(q_2, a) \cup \dots \cup \delta(q_k, a)$$

$$\text{Then} \quad \delta'([q_1, q_2, \dots, q_k], a) = [p_1, \dots, p_k] \text{ If and only if}$$

$$\delta'([q_1, q_2, \dots, q_k], a) = \{p_1, p_2, \dots, p_k\}$$

$L = T(M')$ we prove an auxiliary result.

$$\delta^*(q_0, y) = \{q_1, \dots, q_k\}$$

If and only if $\delta(q_0, y) = \{q_1, \dots, q_k\}$ for all $x \in \Sigma^*$

Q. 1. (b) Prove or disprove whether following language is regular or not.

L → The set of all binary string that read backwards the same as forwards (pallindrome).

Ans.

Step 1 : Let L is regular and n be the number of states in corresponding FA.

Step 2 : Let $w = a^n b^n$. Then $|w| = 2n > n$ and

Let $w = xyz$ with $|xy| \leq n$ and $|y| \geq 1$

Step3 : To find $xy^i z \in L$, substring y can be in any of the following forms

Case (i) y has a's, i.e., $y = a^m$ for some $m \geq 1$

Case (ii) y has b's, i.e., $y = b^k$ for some $k \geq 1$

Case (iii) y has both a's and b's, i.e., $y = a^m b^k$

As in case (i) if $i = 0$, as $xyz = a^n b^n \Rightarrow xz = a^{n-m} b^n$ as $m \geq 1, n - m \neq n$, so $xy^i z \notin L$.

As in case (ii) if $i = 0$, as $xyz = a^n b^n \Rightarrow xz = a^n b^{n-k}$ as $k \geq 1, n - k \neq n$, then $xy^i z \notin L$.

As in case (iii) if $xyz = a^n b^n$, then $xyz = a^{n-m} a^m b^k b^{n-k} xy^2 z = a^{n-m} a^m b^k b^{n-k} xy^2 z$ is not of the form $a^n b^n$, so $xy^2 z \notin L$.

Q. 2. (a) Give type basic description of moore and mealy machines? Also state the reasons behind their emergence.

Ans. Mealy and Moor Machine: The finite automata which we considered in this section have binary output, i.e., they accept the string or not accept the string. The acceptability of any string decided on the basis of reachability of the final state by the initial state. If we remove this restriction and consider the model where output can be chosen from some other alphabet. The value of output function $X(t)$ is a function of present state $q(x)$ and the present input $x(t)$, i.e.,

$$X(t) = \lambda(q(t), x(t))$$

This model is called Mealy machine where λ is output function. If the output function depends on

present state and is independent of current input. The output function will be

$$X(t) = \lambda(q(t))$$

This model is called Moore machine.

Moore Machine :

The Moore machine is a six-tuple $(Q, \Sigma, \Delta, \delta, \lambda, q_0)$, in which

1. Q is a finite nonempty set of states
2. Σ is a set of input letters called input alphabets.
3. Δ is an alphabet of possible output.
4. δ is a transition function $\Sigma \times Q \rightarrow Q$, i.e., a transition table that shows for each state and each input letter, what state is reached next.
5. λ is the output function mapping Q into Δ .
6. q_0 is the initial state.

Mealy Machine :

Mealy machine is another variation of FA. A Mealy machine is like a Moore machine except that now output letter printing while we are travelling along the edge, not in state themselves. What we print output depends on edge we take. If there are two different edge from one state (i.e., one a-edge, one b-edge) to another state it is possible that they will have different printing instruction for us. We take no printing instruction from state itself.

Definition : A Mealy machine is a six tuple $(Q, \Sigma, \Delta, \delta, \lambda, q_0)$

1. Q is a finite nonempty set of states
2. Σ is a set of input letters called input alphabets.
3. Δ is an alphabet of possible output.
4. δ is a transition function $\Sigma \times Q \rightarrow Q$, i.e., a transition table that shows for each state and each input letter, what state is reached next.
5. λ is the output function mapping $Q \times \Sigma \rightarrow \Delta$
6. q_0 is the initial state.

Q.2. (b) Let $(Q, \Sigma, \Delta, \delta, \lambda, q_0)$ be a Mealy machine. Then there is a Moore machine M_2 equivalent to M_1 . Prove it.

Ans. Consider the following Mealy Machine :

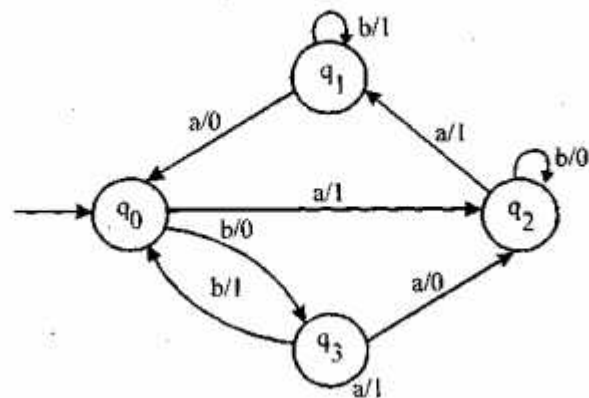


Fig. 1

We convert the transition diagram into the following transition table :

Present State	Next State			
	Input a		Input b	
	State	Output	State	Output
→ q ₀	q ₂	1	q ₃	0
q ₁	q ₀	0	q ₁	1
q ₂	q ₁	1	q ₂	0
q ₃	q ₂	0	q ₀	1

In the first step we develop the procedure so that both machines accept exactly the same set of input sequence.

We first look into the next state column for any state say q₁ and determine the number of different outputs associated with q₁ in that column.

We split the q₁ into several different states. The number of such states being equal to the number of different outputs associated with q₁.

Now we apply this method in given problem from the next state column.

q₀ is associated with two outputs 0 and 1. So we split it into two states q₀₀ associated with output 0 and q₀₁ associated with output 1.

We must select the initial state for new machine so let us arbitrarily select q₀₀.

q₁ is associated with output 1.

q_2 is associated with output 0 and 1 so we split it into two states q_{20} associated with output 0 and q_{21} associated with output 1.

q_3 is associated with output 0.

Now reconstructed table for new states.

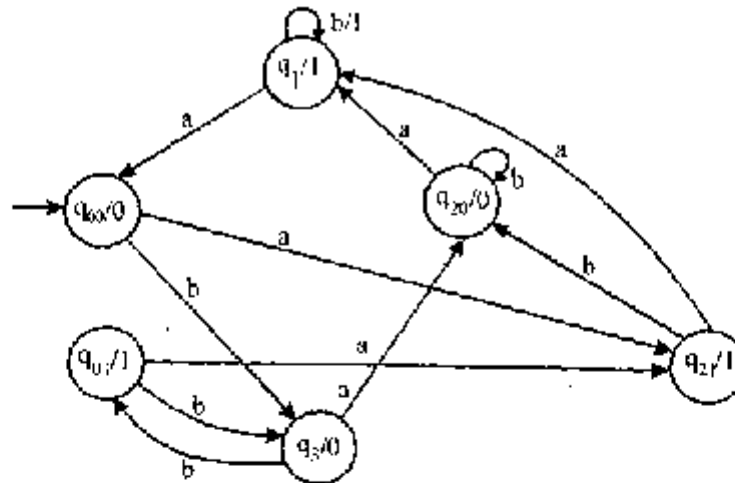
Present State	Next State			
	Input a		Input b	
	State	Output	State	Output
$\rightarrow q_{00}$	q_{21}	1	q_3	0
q_{01}	q_{21}	1	q_3	0
q_1	q_{00}	0	q_1	1
q_{20}	q_1	1	q_{20}	0
q_{21}	q_1	1	q_{20}	0
q_3	q_{20}	0	q_{01}	1

The pair of states and outputs in next state column can be rearranged as in the following table.

The Revised Table :

Present State	Next State		
	Input a	Input b	Output
$\rightarrow q_{00}$	q_{21}	q_3	0
q_{01}	q_{21}	q_3	1
q_1	q_{00}	q_1	1
q_{20}	q_1	q_{20}	0
q_{21}	q_1	q_{20}	1
q_3	q_{20}	q_{01}	0

Now the transition diagram for the required Moore machine is given in Fig. 2



Q. 3. (a) What is Pumping Lemma? Discuss the applications of the Pumping Lemma with examples.

Ans. The application of Pumping lemma are :

Pumping Lemma is used to prove that certain sets are not regular. The following steps are needed for proving that a given set is not regular.

Step 1 : Assume that given language is regular and n be the states in corresponding finite automata.

Step 2 : Choose a string w such that $|w| \geq n$ breaking the string into three substring x, y, z , i.e., $w = xyz$ with $|xy| \leq n, |y| \geq 1$.

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

For example, we want to prove that $L = \{a^n b^n : n \geq 1\}$ is regular or not. Then we can prove it using Pumping Lemma as shown below :

Step 1 : Let L is regular and n be the number of states in corresponding FA.

Step 2 : Let $w = a^n b^n$. Then $|w| = 2n > n$ and

Let $w = xyz$ with $|xy| \leq n$ and $|y| \geq 1$.

Step 3 : To find $xy^iz \notin L$, substring y can be in any of the following forms :

Case (i) y has a 's, i.e., $y = a^m$ for some $m \geq 1$.

Case (ii) y has b 's, i.e., $y = b^k$ for some $k \geq 1$.

Case (iii) y has both a 's and b 's, i.e., $y = a^m b^k$

As in case (i) if $i = 0$, as $xyz = a^n b^n \Rightarrow xy = a^{n-m} b^n$ as $m \geq 1, n-m \neq n$, so $xy^i z \notin L$.

As in case (ii) if $i = 0$, as $xyz = a^n b^n \Rightarrow xz = a^n b^{n-k}$ as $k \geq 1, n-k \neq n$, then $xy^i z \notin L$.

As in case (iii) if $xyz = a^n b^n$, then $xyz = a^{n-m} a^m b^k b^{n-k}$.

$$xy^2 z = a^{n-m} a^m b^k a^m b^k b^{n-k}.$$

$xy^2 z$ is not of the form $a^n b^n$, so $xy^2 z \notin L$.

Q. 3. (b) State the Myhill-Mecode Theorem and its applications.

Ans. My Hill-Nirode Theorem :

For any language L , we use equivalence relation R_L i.e., $xR_L y$ if and only if for each string z , either both xy, yz or neither of xz and yz is in L . The index, i.e., number of equivalence classes is always finite if L is regular set.

Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA for string x and y in Σ^* . Let $xR_M y$ if and only if $\delta(q_0, x) = \delta(q_0, y)$. R_M divides the set Σ^* into equivalence classes one for each state that is reachable from q_0 .

If $xR_M y$, then $xzR_M yz$ for all z in Σ^* since

$$\delta(q_0, xy) = \delta(\delta(q_0, x), z) = \delta(\delta(q_0, y), z) = \delta(q_0, yz)$$

An equivalence-relation R such that $xR_y \Rightarrow xzRyz$ is said to be right invariant with respect to concatenation.

Theorem : The following three statements are equivalent :

1. The set $L \subseteq \Sigma^*$ is accepted by some finite automation.
2. L is the union of some of the equivalence classes of a right invariant equivalence relation of finite index.
3. Let equivalence relation R_L be defined by : $xR_L y$ if and only if for all z in Σ^* , xz is in L exactly

when yz is in L . Then R_L is of finite index.

My Hill-NIRODE theorem is used for the immunization of automata and in construction of minimum automata.

Q. 3. (c) Prove that the regular sets are closed under union and concatenation.

Ans. If L_1 and L_2 are regular, there are regular expression r_1 and r_2 that define these languages L_1 and L_2 . Then $(r_1 + r_2)$ is also a regular expression that defines the language $L_1 + L_2$.

The language $L_1 L_2$ can be defined by regular expression by regular expression $r_1 r_2$.

The language L^* can be defined by the regular expression $(r_1)^*$. Therefore all these sets of words are definable by regular expression so $L_1 + L_2$, $L_1 L_2$ and L_1^* are regular languages.

Q. 4. (a) Describe context free and context sensitive grammar with examples.

Ans. Context sensitive grammar :

Suppose we place the restriction on production $\alpha \rightarrow \beta$ of a phrase structure grammar that β be at least as long as α . Then we can call the resulting grammar context-sensitive and the language is context-sensitive language. The term 'context-sensitive' comes from a normal form for those grammars where each production is of the form $\alpha_1 A \alpha_2 \rightarrow \alpha_1 \beta \alpha_2$ with $\beta \neq \epsilon$.

Definition (1) : A grammar $G = (V, \Gamma, \delta, P)$ is said to be context-sensitive if all productions of the form

$$x \rightarrow y$$

where $x, y \in (V \cup T)^+$ and $|x| \leq |y|$

Definition (2) : A language L is said to be context-sensitive if there exists a context-sensitive grammar G such that $L = L(G)$ or $L = L(G) \cup \{\lambda\}$.

Context Free Grammar :

It is collection of three things :

1. An alphabet Σ of letters called terminal from which we are going to make strings that will be the word of a language.
2. A set of symbols called non terminals. One of which is the symbol S , called start symbol.
3. A finite set of productions of the form

One non terminal \rightarrow Finite strings of terminals / or non terminal.

E.g. : The grammar $G = (\{S\}, \{a, b\}, P, S)$ with productions

$$S \rightarrow aSa;$$

$$S \rightarrow bSb;$$

$$S \rightarrow \epsilon$$

Also, The grammar with productions

$$S \rightarrow abY$$

$$X \rightarrow aaYb$$

$$Y \rightarrow bbXa$$

$$X \rightarrow \epsilon$$

is also a context free grammar.

Q. 4. (b) Convert the grammar into Chomsky Normal Form :

$$S \rightarrow bA / aB$$

$$A \rightarrow bAA / aS / a$$

$$B \rightarrow aBB / bS / b$$

Ans.

Step 1 : The given grammar is without unit and null productions.

Step 2 : Let $G_1 = \{V_N, \{a, b\}, P, S\}$

(i) $A \rightarrow a, B \rightarrow b$ are in Chomsky normal form, add in P'

(ii) $S \rightarrow bA$ gives $S \rightarrow YA$ and $Y \rightarrow b$

$$S \rightarrow aB \text{ gives } S \rightarrow XB \text{ and } X \rightarrow a$$

$$A \rightarrow bAA \text{ gives } A \rightarrow YAA$$

$$A \rightarrow aS \text{ gives } A \rightarrow XS$$

$$B \rightarrow aBB \text{ gives } B \rightarrow XBB$$

$$B \rightarrow bS \text{ gives } B \rightarrow YS$$

Now $V_N = \{S, A, B, X, Y\}$

P' consists of

$$S \rightarrow YA$$

$$S \rightarrow XB$$

$$A \rightarrow YAA$$

$$A \rightarrow XS$$

$$A \rightarrow a$$

$$B \rightarrow XBB$$

$$B \rightarrow YS$$

$$B \rightarrow b$$

$$X \rightarrow a$$

$$Y \rightarrow b$$

Step 3: In P' productions $A \rightarrow YAA$ and $B \rightarrow XBB$ are not in Chomsky normal form. We need simply only two productions.

$$A \rightarrow YAA \text{ gives } A \rightarrow YR_1$$

$$R_1 \text{ becomes } R_1 \rightarrow AA$$

$$B \rightarrow XBB \text{ gives } B \rightarrow XR_2$$

$$R_2 \text{ becomes } R_2 \rightarrow BB$$

Now P'' is

$$S \rightarrow YA|XB$$

$$A \rightarrow YR_1|XS|a$$

$$B \rightarrow XR_2|YS|b$$

$$X \rightarrow a$$

$$Y \rightarrow b$$

$$R_1 \rightarrow AA$$

$$R_2 \rightarrow BB$$

$$V''_N = \{S, A, B, X, Y, R_1, R_2\}$$

$$G_2 = (\{S, A, B, X, Y, R_1, R_2\}, \{a, b\}, P'', S)$$

All productions in G_2 are in Chomsky normal form.

Q. 5. (a) Give the formal definition of PDA, also state applications of PDA.

Ans. A push down automata is defined by seven tuple

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

where,

- (i) Q is a finite set of internal states
- (ii) Σ is the input alphabet
- (iii) Γ is finite set of pushdown symbols
- (iv) the transition function δ from $Q \times (\Sigma \cup \{\wedge\}) \times \Gamma^*$ to the set of finite subsets of $Q \times \Gamma^*$
- (v) $q_0 \in Q$ is the special pushdown symbol called the initial symbol on the pushdown store.
- (vii) $F \subseteq Q$ is the set of final states.

The argument of δ are current state of the control unit, the current input symbol and current symbol on top of the stack. The result is a set of pairs (q, x) where q is the next state and x is a string which put on top of the stack in place of the single symbol there before. Note that the second argument of δ may be \wedge indicating that a move that does not consume an input symbol is possible. We will call such a move a \wedge -transition. δ is defined so that it needs a stack symbol no move is possible if stack is empty. Finally the requirement that the range of δ be finite subset is necessary because $Q \times \Gamma^*$ is an infinite set and therefore has infinite subsets.

Example : Suppose the set of transition rules in PDA contains

$$\delta(q_1, a, b) = \{(q_2, a)\} \quad \dots(1)$$

$$\delta(q_1, b, a) = \{(q_1, \wedge)\} \quad \dots(2)$$

according to rule (1).

If at any time the PDA is in state q_1 , the input symbol read is a , the symbol on top of the stack is b , then the PDA goes into state q_2 and the a replace b on top of the stack. According to rule (2), if the PDA is in state q_1 , the input symbol read is b , the symbol on top of the stack is a then PDA goes into state q_1 and the symbol a is removed from the stack.

Q. 5. (b) Construct a PDA equivalent to the following grammar.

$$S \rightarrow aAA, A \rightarrow aS / bS / a.$$

Ans. Since the grammar is already in Grieback Normal form, So We define PDA as follows.

$$M = (\{q\}, \{a, b\}, \{S, A, a, b\}, \delta, q, S, \phi)$$

S is defined by the following rules :

$$R_1: \delta(q, \wedge, S) = \{(q, aAA)\}$$

$$R_2: \delta(q, \wedge, A) = \{(q, aS)\}$$

$$R_3: \delta(q, \wedge, A) = \{(q, bS)\}$$

$$R_4: \delta(q, \wedge, A) = \{(q, a)\}$$

$$R_5: \delta(q, a, a) = \{(q, \wedge)\}$$

$$R_6: \delta(q, b, b) = \{(q, \wedge)\}$$

$$(q, aba^4, S) \vdash (q, aba^4, aAA) \vdash$$

$$(q, ba^4, AA) \vdash (a, ba^4, bSA) \vdash (q, a^4, SA)$$

$$\vdash (q, aaaa, aAAA) \vdash (q, aaa, AAA)$$

$$\vdash (q, aaa, aAA) \vdash (q, aa, AA)$$

$$\vdash (q, aa, aA) \vdash (q, a, A) \vdash (q, a, a) \vdash (q, \wedge, \wedge)$$

Thus $aba^4 \in N(A)$.

Q. 6. (a) Define a TM mathematically. Also differentiate the deterministic and non-deterministic Turing Machine.

Ans. A Turing machine is defined by

$$M = (Q, \Sigma, \Gamma, \delta, q_0, b, F)$$

where

1. Q is the set of states.
2. Σ is a set of input symbols and $b \notin \Sigma$.
3. Γ is a set of tape symbols called tape alphabet.
4. δ is the transition function such that δ is

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

Current state of the machine and current tape symbol being read. The result is a new state of the machine, a new tape symbol, which replace old one and a move symbol L or R.

5. $q_0 \in Q$ is the initial state.
6. $b \in \Gamma$ is a special symbol called blank.
7. $F \subseteq Q$ is the set of final states.

Non-deterministic Turing Machine :

A non-deterministic Turing Machine (NTM) differ from the deterministic variety we have been studying by having a transition function δ such that for each state q and tape symbol X , $\delta(q, X)$ is a set of triples $\{(q_1, Y_1, D_1), (q_2, Y_2, D_2), \dots, (q_k, Y_k, D_k)\}$ for any finite integer k .

The NTM is a device with a finite control and a single one-way infinite tape. The machine has a finite number of choices for the next move, for a given state and tape symbol, scanned by the head. Each choice consists of a new state, a tape symbol to print, and a direction of head motion. The existence of other choices that do not lead to an accepting state is irrelevant as it is for the NDFA to PDA. As with the finite automation NTM's accept no language not accepted by a deterministic Turing machine. The combination of non-determinism with any of the extension presented such as two-way infinite or multitape Turing Machines does not add extra power.

Q. 6. (b) Write short notes on following :

- (i) Post correspondence problem.
- (ii) Halting problem of T. M.

Ans. (i) Post correspondence problem : The Post Correspondence Problem (PCP) was first introduced by Emil Post in 1946. Here, we begin reducing undecidable questions about Turing machine to undecidable questions about 'reel;' thing that is common matters that have nothing to do with the abstraction of the Turing machine. The Post correspondence problem involves strings rather than Turing machines.

The Post correspondence problem can be stated as follows :

$$A = w_1, w_2, \dots, w_n$$

and
$$B = v_1, v_2, \dots, v_n.$$

Then, we say that there exists a Post correspondence solution (PC-solution) for pair (A, B) if there is a non-empty sequence of integer i, j, k, \dots , such that

$$w_i w_j \dots w_k = v_i v_j \dots v_k$$

(ii) Halting Problem of T.M.

Halting Problem :

For a given configuration Turing Machine, there are following two cases :

- (a) The machine starting at this configuration will halt after a finite number of steps.
- (b) The machine starting at this configuration never reaches a halt, no matter how long it runs.

We are asking for a decision procedure, consisting of a simple set of instructions given once and for all, that will enable us to solve the halting problem for every pair. The answer is that the halting problem is unsolvable suppose that there exists a Turing machine T_1 which decides whether or not any computation by a Turing machine T will ever halt, given the description of T and tape t of T then for every input (t, d_T) to T_1 if T halts for the input, t , T_1 reads on accept halt. If T does not halt for the input t , then T_1 reached a reject halt. Now, we can construct another Turing machine T_2 which takes d_T as the input and proceed as follows. First it copies the input d_T and duplicates d_T on its tape and then takes this duplicate information tape as input to T_1 with one modification.

Q. 7. (a) Define Chomsky hierarchy of languages.

Ans. Noam chomsky a founder of formal language theory provide an initial classification in to four language type 0 to type 3 in terms of production.

Type 0 : A type 0 grammar is any phrase structure grammar without any restriction. Type 0 includes all productions of the form

any string \rightarrow any string

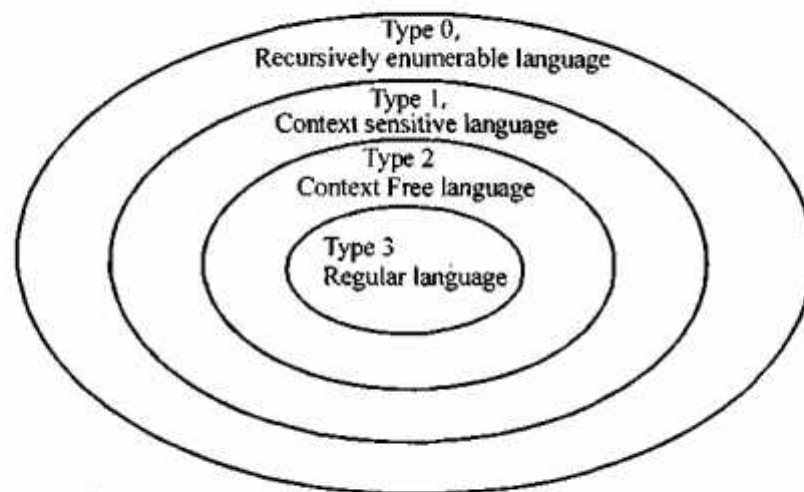
i.e. $\alpha \rightarrow \beta$

where α is in $(VNU\Sigma)^+$ and β is in $(VNU\Sigma)^*$.

Type 1 : A grammar is said to be type 1 grammar in which left side of each production is not larger than the right side is called context sensitive grammar denoted as CSG or Type 1. e.g. A grammar G is said to be Type 1 grammar if every production $\alpha \rightarrow \beta$ has the property $|\alpha| \leq |\beta|$ where $|\alpha|$ and $|\beta|$ are the length of left side and length of right side respectively.

Type 2 : A grammar is type 2 grammar if every production is of form $A \rightarrow \beta$, i.e. the left side is a variable and right side is a word in one or more symbols or $A \in VN$ and $\beta \in (VNU\Sigma)^+$.

Type 3 : A grammar is type 3, if all the productions are of the form $A \rightarrow a$ or $A \rightarrow aB$, e.g. the left side is a single variable and right side is either a single terminal or a terminal followed by a variable. $A \rightarrow \Lambda$ is allowed in type 3 grammar.



Relation between languages

Q. 7. (b) Prove that : There is a recursive language that is not context sensitive.

Ans. Let M_1, M_2, \dots be an enumeration of some set of Turing machines that halt on all inputs. Then there is some recursive language that is not $L(M_i)$ for every i .

Theorem 2 : For any non empty Σ , there exists language that are not regressively enumerable.

It remains to show that we can enumerate halting Turing machines for the context-sensitive languages over alphabet $\{0, 1\}$. Let the 4-tuples representation for CSG's with terminal alphabet $\{0, 1\}$ be given some binary coding. Let be the Turing machine implementing the algorithm of Theorem-2 that recognize the language of the CSG with binary code. Clearly, $L(M_1)$ always halts whether the input is accepted or not.

Q. 8. (a) Define the following functions known as initial function over N :

- (i) Zero function
- (ii) Successor function
- (iii) Projection function.

Ans. (i) Zero function : The zero function Z is defined as $Z(a_1) = 0 \forall a_1 \geq 0$ therefore the initial tape expression can be taken as $X = 1^{a_1} x_1 b y$. As we require the computed value $Z(a_1)$, i.e., 0 to appear to the left of y , we require the machine to halt without changing the input.

Therefore, we define a Turing machine by taking $Q = \{q_0, q_1\}$, $\Gamma = \{b, 1, x_1, y\}$, $X = 1^{a_1} x_1 b y$. Also, P consists of $q_0 b R q_0$, $q_0 1 R q_0$, $q_0 x_1 x_1 q_1$, $q_0 b R q_0$ and $q_1 1 R q_0$ are used to move to the right until x_1 is encountered. $q_0 x_1 x_1 q_1$ enables the Turing machine to enter state q_1 . M enters q_1 without altering the tape symbol.

(ii) Successor function : The successor function S is defined by $S(a_1) = a_1 + 1 \forall a_1 \geq 0$. Therefore, the initial tape expression can be taken as $X = 1^{a_1} x_1 b y$. At the end of the computation, we require 1^{a_1+1} to appear to the left of y . Hence, we define a Turing machine by taking

$$Q = \{q_0, \dots, q_9\}, \Gamma = \{b, 1, x_1, y\}, X = 1^{a_1} x_1 b y$$

Here, P consists of

- (i) $q_0 b R q_0$, $q_0 1 b q_1$, $q_0 x_1 R q_0$
- (ii) $q_1 b R q_1$, $q_1 1 R q_1$, $q_1 x_1 R q_1$, $q_1 y 1 q_2$
- (iii) $q_1 1 R q_2$, $q_2 b y q_3$
- (iv) $q_3 b L q_3$, $q_3 1 L q_3$, $q_3 y L q_3$, $q_3 x_4 L q_4$
- (v) $q_4 1 L q_4$, $a_4 b 1 q_5$

$$(vi) \quad q_5 I R q_0$$

$$(vii) \quad q_6 b R q_6, q_6 I R q_6, q_6 x I R q_6, q_6 y L q_7$$

$$(viii) \quad q_7 II q_7, q_7 b I q_8$$

$$(ix) \quad q_8 b I q_8, q_8 I L q_8, q_8 y L q_8, q_8 x I x I q_9$$

(iii) **Projection function** : A function $f(x)$ is said to be projection function. If function $F(x)$ is such that

$$F_k(x_1 x_2) = x_k, \quad k = 1, 2, \dots,$$

Q. 8. (b) Define $n!$ by recursion.

Ans. Define $n!$ by recursion. Function $f(x)$ over N is defined by recursion if there exist a constant $K \in N$ and a function $h(x, y)$ such that

$$f(0) = k,$$

$$f(n+1) = h(n, f(n))$$

By induction on n , we can define $f(n)$ for all n . As $f(0) = K \in N$, there is basis for induction. Once $F(n)$ is known, $f(n+1)$ can be evaluated by using above equation.