

Don't believe blindly on these most questions. It is just prediction based.

THEORY OF COMPUTATION

IV C.S.

(USE MY NOTES)

FINITE AUTOMATA AND REGULAR EXPRESSION

1

PREVIOUS YEARS QUESTIONS

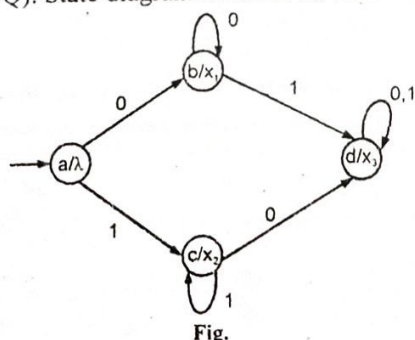
PART-A

Q.1 What is Moore Machine?

[R.T.U. 2019]

Ans. Moore Machine : Moore Machine is an FSM whose outputs depend on only present state. It can be described by a 6 tuple $(Q, \Sigma, O, \delta, X, g_0)$ where :

- Q is a finite set of states.
- Σ is a finite set of symbols called input alphabets.
- O is a finite set of symbols called output alphabets.
- δ is input transition function where $\delta : Q \times \Sigma \rightarrow Q$
- X is the output transition function where $X : Q \times \Sigma \rightarrow O$
- g_0 is initial state from where any input is processed ($g_0 \in Q$). State diagram is as shown below :



Q.2 What is N DFA?

[R.T.U. 2019]

Ans. Non-deterministic Finite Automaton (N DFA) : In N DFA, for a particular input symbol, the machine can move to any combination of the states in the machine. In other

words, the exact state to which the machine moves cannot be determined. Hence, it is called **Non-deterministic Automaton**. As it has finite number of states, the machine is called **Non-deterministic Finite Machine** or **Non-deterministic Finite Automaton**.

Formal Definition of an N DFA

An N DFA can be represented by a 5-tuple $(Q, \Sigma, \delta, q_0, F)$ where :

- Q is a finite set of states.
- Σ is a finite set of symbols called the alphabets.
- δ is the transition function where $\delta : Q \times \Sigma \rightarrow 2^Q$

(Here the power set of Q (2^Q) has been taken because in case of N DFA, from a state, transition can occur to any combination of Q states.)

- q_0 is the initial state from where any input is processed ($q_0 \in Q$).
- F is a set of final state/states of Q ($F \subseteq Q$).

Q.3 What is finite automata?

[R.T.U. 2019]

Ans. Finite Automaton : A finite automaton can be represented by a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

- Q is a finite non-empty set of states.
- Σ is a finite non-empty set of inputs called input alphabets.
- δ is a function which maps $Q \times \Sigma$ into Q and is usually called direct transition function. This is the function which describes the changes of states during the transition. This mapping is usually represented by a transition table or a transition diagram.
- $q_0 \in Q$ is the initial state.
- $F \subseteq Q$ is the set of final states. It is assumed here that there may be more than one final state.

TOC.2

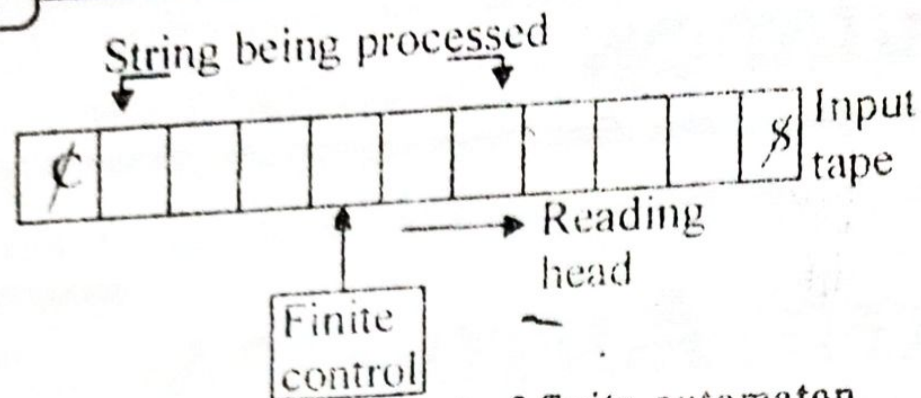


Fig. : Block diagram of finite automaton

- (vi) **Input Tape** : The input tape is divided into squares, each square containing a single symbol from the input alphabet Σ . The end square of the tape contain end-markers ϵ at the left end and $\$$ at the right end. Absence of end-markers indicates that the tape is of infinite length. The left-to-right sequence of symbol between the end markers is the input string to be processed.
- (vii) **Reading Head** : The head examines only one square at a time and can move one square either to the left or to the right. For further analysis, we restrict the movement of R-head only to the right side.
- (viii) **Finite Control** : The input to the finite control will be usually : symbol under the R-head, say a , or the present state of the machine, say q , to give the following outputs : (a) A motion of R-head along the tape to the next square (In some a null move *i.e.* R-head remaining to the same square is permitted); (b) The next state of a finite state machine given by $\delta(q, a)$.

Initial states are q_0 and q_1 .

Final state is q_3 .

For checking the string acceptability, we start the string from initial state. If we reach the final state after completing the string, then we say that this string is accepted by transition system or not.

Q.11 Give applications of pumping lemma.

Ans.(i) Assume L is regular: Let n be the number of state in the FA which accepts L .

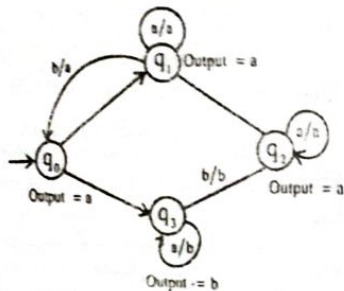
(ii) Choose a string w such that $|w| \geq n$. Using pumping lemma write $w = xyz$ with $|xy| \leq n$ and $|y| > 0$.

(iii) Find suitable integer i such that $xy^i z \notin L$. So it contradicts the assumption we have made in step (i) above. Hence L is not regular.

It is then clear that, the most important aspect of the method is to find i such that $xy^i z \notin L$. Some times we have to prove $xy^i z \in L$ by considering $|xy^i z|$ or some times we need to use the structure of strings in L .

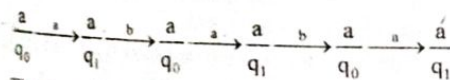
PART-B

Q.12 Consider the Moore Machine shown in figure. What is the output for the input ababa?



[R.T.U. 2019]

Ans. For the input = ababa



The output for the input string = ababa is aaaa

Q.13 Convert the following Moore Machine into Mealy Machine:

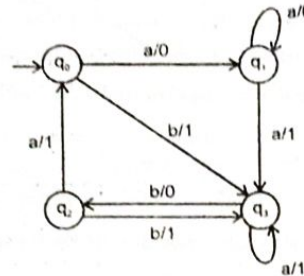
| State | Input | | Output |
|-------------------|-------|-------|--------|
| | a | b | |
| $\rightarrow q_0$ | q_1 | q_3 | 1 |
| q_1 | q_3 | q_1 | 0 |
| q_2 | q_0 | q_3 | 0 |
| q_3 | q_3 | q_2 | 1 |

[R.T.U. 2019]

Ans. Mealy Machine:

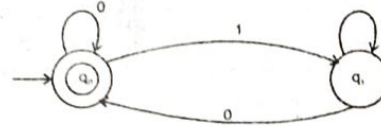
Next State

| P.S. | i = a | o/p | i/p = b | o/p |
|-------------------|-------|-----|---------|-----|
| $\rightarrow q_0$ | q_1 | 0 | q_3 | 1 |
| q_1 | q_3 | 1 | q_1 | 0 |
| q_2 | q_0 | 1 | q_3 | 1 |
| q_3 | q_3 | 1 | q_2 | 0 |



Q.14 Design a FA which checks whether the given binary number is even. [R.T.U. 2019]

Ans.



Q.15 Explain the difference between deterministic and non-deterministic finite automaton. [R.T.U. 2019] OR

State the difference between deterministic and non deterministic finite automata.

[R.T.U. 2015, 2014]

Ans. Deterministic and Non-deterministic Finite Acceptors

| S. No. | Deterministic Finite Acceptors | Non- deterministic Finite Acceptors |
|--------|---|--|
| 1. | For every symbol of the alphabet, there is only one state transition. | We do not need to specify how does the NFA react according to some symbol. |
| 2. | Cannot use empty string transition. | Can use empty string transition. |
| 3. | Can be understood as one machine. | Can be understood as multiple little machines computing at the same time. |
| 4. | It will reject the string if it end at other than accepting state. | If all of the branches of NFA dies or rejects the string, we can say that NFA reject the string. |

Theory of Computation

| S. No. | Deterministic Finite Acceptors | Non- deterministic Finite Acceptors |
|--------|---|--|
| 1. | For every symbol of the alphabet, there is only one state transition. | We do not need to specify how does the NFA react according to some symbol. |
| 2. | Cannot use empty string transition. | Can use empty string transition. |
| 3. | Can be understood as one machine. | Can be understood as multiple little machines computing at the same time. |
| 4. | It will reject the string if it end at other than accepting state. | If all of the branches of NFA dies or rejects the string, we can say that NFA reject the string. |
| 5. | The transition function is single valued. | The transition function is multi-valued. |
| 6. | Checking membership is easy. | Checking membership is difficult. |
| 7. | Construction is difficult. | Construction is easy. |
| 8. | Space required is more. | Space required is comparatively less. |
| 9. | Backtracking is allowed. | Not possible in every case. |
| 10. | Can be constructed for every input and output. | Cannot be constructed for every input and output. |
| 11. | There can be more than one final state. | There can only be one final state. |

Q.16 (a) Describe the block diagram of a finite automaton. Consider the transition system given below:

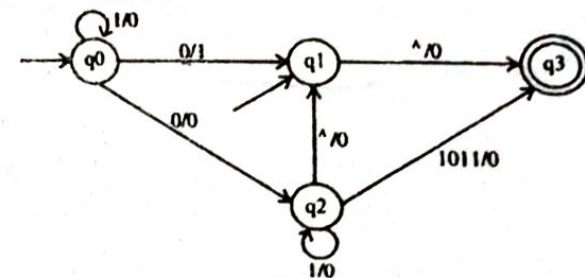


Fig.

Determine the initial states, the final state and the acceptability of 101011 and 111010.

(b) Prove that for any transition function δ and for any two input string x and y .
 $\delta(q, xy) = \delta(\delta(q, x), y)$

[R.T.U. 2016]

Ans.(a) Finite Automaton: Refer to Q.3.

For string 101011, the path value is $q_0 q_0 q_2 q_3$. Since q_3 is the final state so this string is accepted by above transition system.

For string 111010, there is no path value. So this string is not accepted by the above transition system.

Ans. (1)

(2) F

consu after t Exam any tv

Proof Basis

with where

length string

Q.17

Ans. Exam desc

Ans.

can origi

Mealy Machine:

Mealy Machine is an FSM whose output depends on present state as well as present input

It can be described by a six tuple $(Q, \Sigma, O, \delta, X, g_0)$ where :

- Q is finite set of states
- Σ is finite set of symbols called input alphabets
- O is finite set of symbols called output alphabets
- δ is input transition function where $\delta : Q \times \Sigma \rightarrow Q$
- X is output transition function where $X : Q \times \Sigma \rightarrow O$
- g_0 is initial state from where any input is processed ($g_0 \in Q$). State diagram of Mealy machine is shown below :

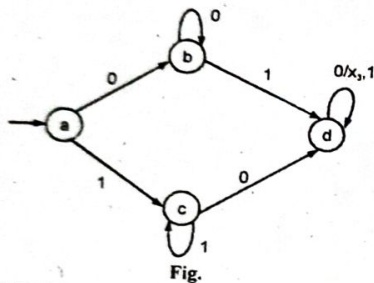


Fig.

Moore Machine: Refer to Q.1.

Difference between Mealy and Moore Machine: Refer to Q. 10.

Q.19 Minimize the following finite automata. Also write procedure for minimization.

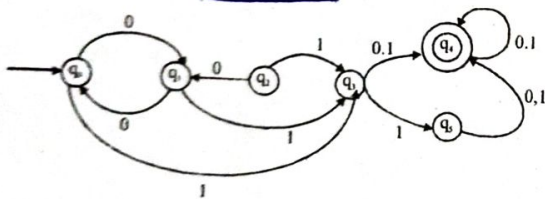


Fig.

[R.T.U. 2014]

Ans. Step 1 : Firstly we make state transition table for above given state transition diagram is given by as follows.

| State / Σ | Input | |
|------------------|-------|--------------|
| | 0 | 1 |
| q_0 | q_1 | q_1 |
| q_1 | q_2 | q_3 |
| q_2 | q_1 | q_3 |
| q_3 | q_4 | $[q_4, q_5]$ |
| q_4 | q_4 | q_4 |
| q_5 | q_4 | q_4 |

Theory of Computation

Step 2 : Now we obtain π_0 by use of grouping

$$\pi_0 = \{\{q_4\}, \{q_0, q_1, q_2, q_3, q_5\}\}$$

Step 3 : Partition set of non final states in such a way that equivalent states are grouped together in separate sets. This is π_1 equivalence criteria used here is:

Two states are said to be equivalent if their state transitions corresponding to same input belong to same set.

Step 4 : The step 3 is repeated for π_2, π_3 and so on until we get

$$\pi_k = \pi_{k+1}$$

where k is any integer and π_{k+1} is required solution.

Step 5 : In this step state transition table is generated in such a way that the set obtained in π_{k+1} are considered to be as one state, the rest of state transitions in the state transition table goes according to original state transition table.

Step 6 : State transition diagram is generated from state transition table.

after applying step 3 we get π_1

$$\pi_1 = \{\{q_4\}, \{q_0, q_1, q_2\}, \{q_3\}, \{q_5\}\}$$

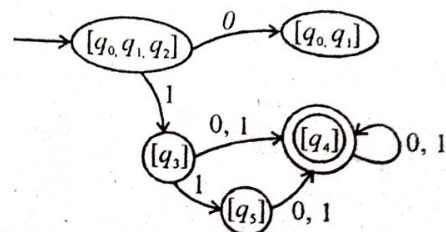
$$\pi_2 = \{\{q_4\}, \{q_0, q_1, q_2\}, \{q_3\}, \{q_5\}\}$$

we get $\pi_2 = \pi_1$ so we stop here.

Now we generate state transition table for minimizing finite automata. The transition table is as follows:

| State / Σ | Input | |
|-------------------|--------------|--------------|
| | 0 | 1 |
| $[q_0, q_1, q_2]$ | $[q_0, q_1]$ | $[q_3]$ |
| $[q_3]$ | $[q_4]$ | $[q_4, q_5]$ |
| $[q_4]$ | $[q_4]$ | $[q_4]$ |
| $[q_5]$ | $[q_4]$ | $[q_4]$ |

State transition diagram for above state transition table is given as follows :



Q.20 Constr to follow

Ans.

Given,

Step (1) : Fin

| State (Σ) |
|--------------------|
| $\rightarrow q_0$ |
| q_1 |
| q_2 |
| q_3 |
| q_4 |

Step (2) : No

- Start w and $[q_3]$
- Then w we do r
- Then w new st
- Then w new st

Step (3) : No NDFA using s

Table : S

| State (Σ) |
|--------------------|
| $[q_0]$ |
| $[q_1]$ |
| $[q_1, q_2]$ |
| $[q_3, q_2]$ |

Q.27 What do you understand by finite automata and regular expression.

Ans. Finite Automata : Refer to Q.3.

Regular Expression : An expression R is a regular expression if R is

1. a for some a in some alphabet Σ ,
2. ϵ ,
3. ϕ ,
4. $(R_1 \cup R_2)$ for some regular expressions R_1 and R_2 ,
5. $(R_1 \cdot R_2)$ for some regular expressions R_1 and R_2 ,
6. $(R_1)^*$ for some regular expression R_1 .

When the meaning is clear from the context, $()$ and can be removed from the expression.

The Language Represented by a Regular Expression

For a regular expression R, $L(R)$ denotes the language it expresses.

1. For each $a \in \Sigma$, $L(a) = \{a\}$.
2. $L(\epsilon) = \{\epsilon\}$
3. $L(\phi) = \phi$
4. $L(R_1 \cup R_2) = L(R_1) \cup L(R_2)$
5. $L(R_1 \cdot R_2) = L(R_1) \cdot L(R_2) = \{uv \mid u \in R_1 \text{ and } v \in R_2\}$
6. $L(R_1^*) = \{\epsilon\} \cup \{u_1 \dots u_k \mid u_1, \dots, u_k \in R_1\}$

Regular Expression Examples

- $L(a) = \{a\}$ (for a single-element regular expression, you may simply write the element)
- $L(abab \cup bc) = \{abab, bc\}$
- $L((abab \cup abc)^*) = \{\epsilon\} \cup \{w_1 \dots w_k \mid k \geq 1 \text{ and } w_1, \dots, w_k \in \{abab, abc\}\}$
- $L((abab \cup abc)^* \cup c^* \cup abc(abca)^*) = \{\epsilon\} \cup \{w_1 \dots w_k \mid k \geq 1 \text{ and } w_1, \dots, w_k \in \{abab, abc\}\} \cup \{w \mid w \text{ is a repetition of } c's\} \cup \{abc, abcabca, abcabcaabca, \dots\}$

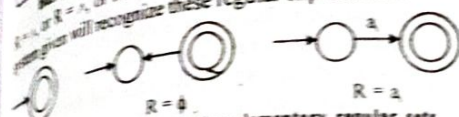
Finite Automata and Regular Expressions :

(1) Transition System and Regular Expression : The following theorem describes the relation between transition system and regular expression.

Theorem : Every regular expression R can be recognized by a transition system i.e. for every string w in the set R, there exists a path from the initial state to a final state with path value w.

Proof : The proof is by the principle of induction on the total number of characters in R. By "Character" we mean elements of $\Sigma, \cup, \phi, *$ and $+$, for example if $R = \wedge + 10^* 11$, the characters are $\wedge + 1, 0, *, 1, 1, *, 0$ and the number of character is 9.

Proof : Let the number of characters in R be 1. Then $R = a$, or $R = \epsilon$, or $R = \phi$, or $R = a, a \in \Sigma$. The transition system given will recognize these regular expressions.



Induction Step : Assume the theorem is true for regular expressions with n character or less. We must prove that it is also true for n + 1 characters. Let R have n + 1 characters.

$R = P + Q$ or $R = PQ$ or $R = P^*$ where P and Q are regular expressions each having n characters or less. By induction hypothesis, P and Q can be recognized by transition system G and H, respectively.

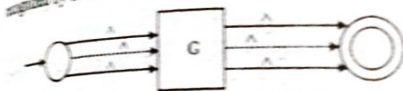


Fig. : Transition system recognizing P

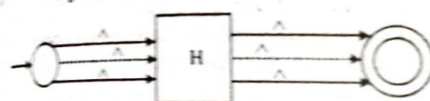


Fig. : Transition system recognizing Q

(2) Transition System Containing \wedge -Moves : The transition systems can be generalized by permitting \wedge -moves or \wedge -moves which are associated with a null input. These transition can occur when no input is applied. It is possible to convert a transition system with \wedge -moves into an equivalent transition system without \wedge -moves we shall give a simple method of doing it with the help of an example.

Suppose we want to replace a \wedge -move from vertex v_1 to vertex v_2 . Then we proceed as follows :
Step-1 : Find all the edges starting from v_1 .
Step-2 : Duplicate all these edges starting from v_1 without changing the edge labels.
Step-3 : If v_1 is an initial state, make v_2 also an initial state.
Example : Consider a finite automaton with \wedge -moves given in Fig. below. Obtain an equivalent automaton without \wedge -moves.

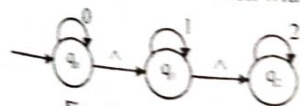
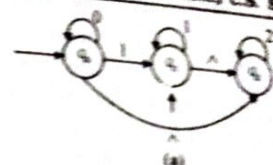


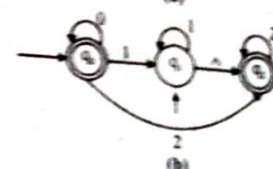
Fig. : FA of Example

We first eliminate the \wedge -moves from q_0 to q_1 to get Fig. (a). q_1 is made an initial state. Then we eliminate the \wedge -moves from q_1 to q_2 in Fig. (a) to get Fig. (b). As q_2 is a final state, q_0 is also made a final state. Finally, the \wedge -moves from q_1 to q_2 is eliminated in Fig. (c).

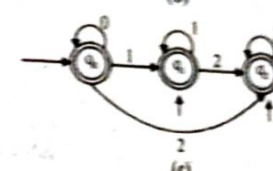
B.Tech. (IV Sem.) C.S. Solved Papers



(a)



(b)



(c)

Fig. : Transition system without \wedge -moves

Q.28 State and explain pumping lemma. Prove that the following language $L = \{a^n \mid n \text{ is a perfect square}\}$ is not regular. [R.T.U. 2012]

Ans. Pumping Lemma for Regular Sets : If we give a necessary condition for an input string to belong to a regular set, the result is called pumping lemma as it gives a method of pumping (generating) many input strings from a given string. As pumping lemma gives a necessary condition, it can be used to show that certain sets are not regular.

Theorem : (Pumping Lemma) : Let $M = (Q, \Sigma, \delta, q_0, F)$ be a finite automaton with n states. Let L be the regular set accepted by M. Let $w \in L$ and $|w| \geq m$. If $m \geq n$, then there exists x, y, z such that $w = xyz$, $y \neq \epsilon$ and $xy^iz \in L$ for each $i \geq 0$.

Proof : Let

$$w = a_1 a_2 \dots a_m, \quad m \geq n$$

$$\{\delta(q_0, a_1 a_2 \dots a_i)\} = q_i$$

$$\text{for } i = 1, 2, \dots, m, \quad Q_1 = \{q_0, q_1, \dots, q_m\}$$

That is, Q_1 is the sequence of states in the path with path value $w = a_1 a_2 \dots a_m$. As there are only n distinct states, at least two states in Q_1 must coincide. Among the various pairs of repeated states, we take the first pair. Let us take them as q_j and q_k ($q_j = q_k$). Then j and k satisfy the condition $0 \leq j < k \leq m$. The string w can be decomposed into three substrings $a_1 a_2 \dots a_j, a_{j+1} \dots a_k$ and $a_{k+1} \dots a_m$. Let x, y, z denote these strings $a_1 a_2 \dots a_j, a_{j+1} \dots a_k, a_{k+1} \dots a_m$ respectively. As

EXT FREE GRAMMARS 2

YEARS QUESTIONS

Q.3 What is context free grammar?
OR
Explain context free grammar in brief. [R.T.U. 2019]

Ans. Context free grammar is also called type 2 Grammar.

$G = \{V_N, \Sigma, P, S\}$ is called context free grammar if every production has form $A \rightarrow \alpha$ where $A \in V_N$ and $\alpha \in (V_N \cup \Sigma)^*$

Let $G = \{S, A, B, \{a, b\},$

$\{S \rightarrow a, S \rightarrow aS, S \rightarrow aAS, S \rightarrow aB, A \rightarrow ab\}, S\}$

then G is called context free grammar.

Q.4 Define degree of ambiguity of W .

Ans. Ambiguity in CFG

A CFG $G = (V, T, P, S)$ is ambiguous if there is at least one string W in $L(G)$ for which there are at least two different parse trees, each with its root labeled S and yielding W . Each parse tree corresponds to a left-most or a right-most derivation. The number of different parse trees of a string W is called the degree of ambiguity of W .

Q.5 Show the grammar

$S \rightarrow aB/ab$

$A \rightarrow aAB/a$

$B \rightarrow ABb/b$

is ambiguous.

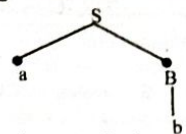
Ans. The left most derivation tree for grammar is

$S \Rightarrow aB$ (using $S \rightarrow aB$)

$S \Rightarrow ab$ (using $B \rightarrow b$)

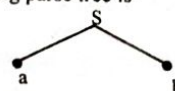
The parse tree can be given as

Theory of Computation



Left most derivation tree can also be given as:
 $S \Rightarrow ab$ (using $S \rightarrow ab$)

The corresponding parse tree is-



Hence we have two ways to represent this grammar. Thus the grammar is ambiguous.

Q.6 Consider the following production:

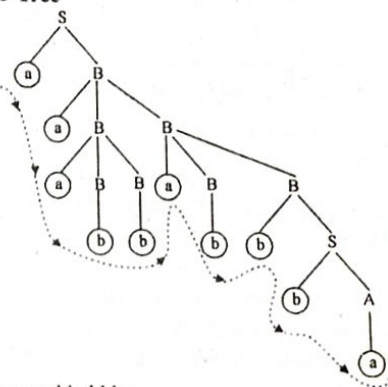
$S \rightarrow aB/bA$

$A \rightarrow as/bAA/a$

$B \rightarrow bs/aBB/b$

find the parse tree.

Ans. Parse Tree



String: aaabbabbba

Q.7 Reduce the grammar to CNF

$S \rightarrow aB/ab$

$A \rightarrow aAB/a$

$B \rightarrow ABb/b$

Ans. Production $A \rightarrow a$ and $B \rightarrow b$ are already in CNF. If introduce two more productions, say

$R_1 \rightarrow a$

$R_2 \rightarrow b$

above grammar can be written as;

$S \rightarrow R_1 B | R_1 R_2$

$A \rightarrow R_1 A B | a$

$B \rightarrow A B R_2 | b$

TOC.17

Still there are two production which are not in CNF, namely $A \rightarrow R_1 AB$ and $B \rightarrow ABR_2$. After further breakup and introducing new symbol R_3 we can write the given grammar in CNF as,

$S \rightarrow R_1 B | R_1 R_2$

$A \rightarrow R_1 R_3 | a$

$B \rightarrow R_3 R_2 | b$

$R_1 \rightarrow a$

$R_2 \rightarrow b$

$R_3 \rightarrow AB$

PART-B

Q.8 Consider the context free grammar -

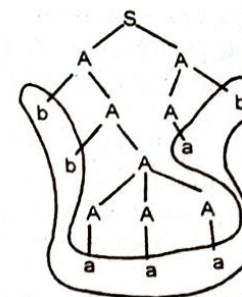
$S \rightarrow AA$

$A \rightarrow AAA|bA|Ab|a$

Find the parse tree for the string bbaaanaab

[R.T.U. 2019]

Ans. $S \rightarrow \underline{AA}$



$\Rightarrow b \underline{AA}$

$\Rightarrow bb \underline{AA}$

$\Rightarrow bb \underline{A} \underline{AAA}$

$\Rightarrow bba \underline{A} \underline{AA}$

$\Rightarrow bbaa \underline{A} \underline{A}$

$\Rightarrow bbaaa \underline{A}$

$\Rightarrow bbaaaa \underline{b}$

$\Rightarrow bbaaanaab$

Q.9 Reduce the following grammars in Chomsky normal form :

- (i) $S \rightarrow |A| 0B, A \rightarrow |AA| 0S | 0, B \rightarrow 0BB | 1S ||$
 (ii) $G = (\{S\}, \{a, b, c\}, \{S \rightarrow a | b | cSS\}, S)$
 (iii) $S \rightarrow a b S b | a | a A b, A \rightarrow b S | a A A b$

[R.T.U. 2016]

Ans. (i) As there are no null productions or unit productions, we can proceed to step 1.

Step 1 : Let $G_1 = (V'_1, \{0, 1\}, P_1, S)$, where P_1 and V'_1 are constructed as follows :

- (i) $A \rightarrow 0, B \rightarrow 1$ are included in P_1 .
 (ii) $S \rightarrow 1A, B \rightarrow 1S$ give rise to $S \rightarrow C_1A, B \rightarrow C_1S$ and $C_1 \rightarrow 1$
 (iii) $S \rightarrow 0B, A \rightarrow 0S$ give rise to $S \rightarrow C_0B, A \rightarrow C_0S$ and $C_0 \rightarrow 0$
 (iv) $A \rightarrow 1AA, B \rightarrow 0BB$ give rise to $A \rightarrow C_1AA$ and $B \rightarrow C_0BB$.

$$V'_1 = \{S, A, B, C_0, C_1\}$$

Step 2 : $G_2 = (V'_2, \{0, 1\}, P_2, S)$, where P_2 and V'_2 are constructed as follows :

- (i) $A \rightarrow 0, B \rightarrow 1, S \rightarrow C_1A, B \rightarrow C_1S$
 $C_1 \rightarrow 1, S \rightarrow C_0B, A \rightarrow C_0S, C_0 \rightarrow 0$ are included in P_2
 (ii) $A \rightarrow C_1AA$ and $B \rightarrow C_0BB$ are replaced by $A \rightarrow C_1D_1, D_1 \rightarrow AA, B \rightarrow C_0D_2, D_2 \rightarrow BB$.

Thus $G_2 = \{S, A, B, C_0, C_1, D_1, D_2\}, \{0, 1\}, \{P_2, S\}$ is in CNF and equivalent to the given grammar where P_2 consists of

$$S \rightarrow C_1A | C_0B, A \rightarrow 0 | C_0S | C_1D_1,$$

$$B \rightarrow 1 | C_1S | C_0D_2, C_1 \rightarrow 1, C_0 \rightarrow 0, D_1 \rightarrow AA$$

and $D_2 \rightarrow BB$.

(ii) Chomsky Normal Form

In the chomsky normal form, we have restrictions on the length of R.H.S. and the nature of symbols in the R.H.S. of productions.

Definition : A context-free grammar G is in chomsky normal form if every production is of the form $A \rightarrow a$, or $A \rightarrow BC$, and $S \rightarrow \Lambda$ is in G if $\Lambda \in L(G)$. When Λ is in $L(G)$, we assume that S does not appear on the R.H.S. of any production.

For example, consider G whose productions are $S \rightarrow AB | \Lambda, A \rightarrow a, B \rightarrow b$. Then G is in Chomsky normal form.

Remark : For a grammar in CNF, the derivation tree has the following property :

Every node has atmost two descendants - either two internal vertices or a single leaf.

When a grammar is in CNF, some of the proofs and constructions are simpler.

The techniques applied in this example are used in the following theorem.

Theorem : (Reduction to Chomsky Normal Form)
 For every context-free grammar, there is an equivalent grammar G_2 in chomsky normal form.

$$G = (\{s\}, \{a, b, c\}, \{s \rightarrow a/b/css\}, s)$$

Given production of G is

$$s \rightarrow a/b/css \quad \dots (A)$$

In a given grammar (A) following productions are in CNF

$$s \rightarrow a$$

$$s \rightarrow b$$

Also, in the given grammar (A), following productions is not in CNF

$$s \rightarrow css$$

So, Consider the production

$$s \rightarrow css$$

We write this production as

$$s \rightarrow v_1ss \quad \dots (1)$$

$$v_1 \rightarrow c \quad \dots (2)$$

Where v_1 is now variable.

So, from (1) and (2), the resultant grammar become

$$\left. \begin{array}{l} s \rightarrow a/b/v_1ss \\ v_1 \rightarrow c \end{array} \right\} \quad \dots (B)$$

Now in the resultant grammar, following production is not in CNF

$$s \rightarrow v_1ss$$

Thus, Consider the production

$$s \rightarrow v_1ss$$

We write this production as

$$s \rightarrow v_1v_2 \quad \dots (3)$$

$$v_2 \rightarrow ss \quad \dots (4)$$

v_2 also a new variable.

So, from (B), (4), the resultant grammar become

$$\left. \begin{array}{l} s \rightarrow a/b/v_1v_2 \\ v_1 \rightarrow c \\ v_2 \rightarrow ss \end{array} \right\} \quad \dots (c)$$

Thus, the resultant grammar (c) is in CNF.

Ans. (iii) Step (1) : Since S appears in R.H.S., we add a new state. S_0 and $S_0 \rightarrow S$ is added to the production set and it becomes :

$$S_0 \rightarrow S$$

$$S \rightarrow abSb | a | aAb$$

$$A \rightarrow bS | aAAb$$

Step 2 : There is no null production to remove in the given production. So move towards next step.

Step 3 : Now we will remove the unit productions. But, here in this question there is only one unit production which is

$$S_0 \rightarrow S$$

After removing $S_0 \rightarrow S$, production becomes :

$$S_0 \rightarrow abSb | a | aAb$$

$$S \rightarrow abSb | a | aAb$$

$$A \rightarrow bS | aAAb$$

Step 4 : Now make right hand side not contain more than 2 non-terminals or 1 terminal

$$S_0 \rightarrow XYSY | a | XAY$$

$$X \rightarrow a$$

$$S \rightarrow XYSY | a | XAY$$

$$Y \rightarrow b$$

$$A \rightarrow YS | XAAY$$

Step 5 : Now transform the production into Chomsky normal form.

$$S_0 \rightarrow XP | a | XQ$$

$$S \rightarrow XP | a | XQ$$

$$A \rightarrow YS | XR$$

$$X \rightarrow a$$

$$Y \rightarrow b$$

$$P \rightarrow YSY$$

$$Q \rightarrow AY$$

$$R \rightarrow AAY$$

Now production of S_0, S, A, X, Y and Q are in Chomsky Normal form but P and R is not so make them.

$$P \rightarrow YM$$

$$R \rightarrow AN$$

$$M \rightarrow SY$$

$$N \rightarrow AY$$

Now the final production in Chomsky normal form is

$$S_0 \rightarrow XP | a | XQ$$

$$S \rightarrow XP | a | XQ$$

$$A \rightarrow YS | XR$$

$$X \rightarrow a$$

$$Y \rightarrow b$$

$$P \rightarrow YM$$

$$R \rightarrow AN$$

$$Q \rightarrow AY$$

$$M \rightarrow SY$$

$$N \rightarrow AY$$

B]

| | |
|-------------|-----------|
| →baaB | [S → aB] |
| →baaaBB | [B → aBB] |
| →baaabB | [B → b] |
| →baaabBS | [B → bS] |
| →baaabbaB | [S → aB] |
| →baaabbaBS | [B → bS] |
| →baaabbaBSA | [S → bA] |
| →baaabbaBSA | [A → a] |

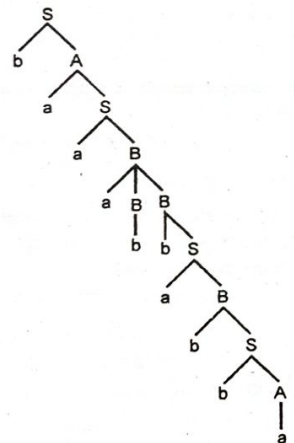
Yield =baaabbaBSA

Right Most Derivation :

| | |
|--------------|-----------|
| S → bA | |
| →baS | [A → aS] |
| →baaB | [S → aB] |
| →baaaBB | [B → aBB] |
| →baaaBbs | [B → bS] |
| →baaaBbaB | [S → aB] |
| →baaaBbabs | [B → bS] |
| →baaaBbabbaA | [S → bA] |
| →baaaBbabbaA | [A → a] |
| →baaabbaBSA | [B → b] |

Yield =baaabbaBSA

Parse Tree

left most
parse tree.
T.U. 2015]

e this :

Q.12 Explain Greibach normal form in detail.

[R.T.U. 2014]

Ans. In computer science and formal language theory, context-free grammar is in Greibach normal form (GNF) if the right-hand sides of all production rules start with a terminal

A → aS]

Theory of Computation

symbol, optionally followed by some variables. A non-strict form allows one exception to this format restriction for allowing the empty word (epsilon, ϵ) to be a member of the described language. The normal form bears the name of Sheila Greibach.

More precisely, a context-free grammar is in Greibach normal form, if all production rules are of the form:

$$A \rightarrow aA_1A_2 \dots A_n$$

or $S \rightarrow \epsilon$

where A is a nonterminal symbol, a is a terminal symbol, $A_1A_2 \dots A_n$ is a (possibly empty) sequence of nonterminal symbols not including the start symbol, S is the start symbol, and ϵ is the empty word.

Observe that the grammar does not have left recursions.

Every context-free grammar can be transformed into an equivalent grammar in Greibach normal form.

Some definitions do not consider the second form of rule to be permitted, in which case a context free grammar that can generate the empty word cannot be so transformed. In particular, there is a construction ensuring that the resulting normal form grammar is size at most $O(n^4)$, where n is the size of the original grammar. This conversion can be used to prove that every context free language can be accepted by a non-deterministic pushdown automaton.

Given a grammar in GNF and a derivable string in the grammar with length n , any top-down parser will halt at depth n .

Q.13 The production of any grammar ϵ is given by

$$S \rightarrow 0B/1A \quad A \rightarrow 0/0S/1AA$$

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

For the string 00110101, find leftmost derivation, rightmost derivation and derivation tree.

[R.T.U. 2013]

Ans.(i) Left most derivation

$$S \rightarrow 0B$$

$$\rightarrow 00BB$$

$$\rightarrow 001B$$

$$\rightarrow 0011S$$

$$\rightarrow 00110B$$

$$\rightarrow 001101S$$

$$\rightarrow 0011010B$$

$$\rightarrow 00110101$$

(ii) Right most derivation

$$S \rightarrow 0B$$

$$\rightarrow 00BB$$

$$\rightarrow 00B1S$$

$$\rightarrow 00B10B$$

$$\rightarrow 00B101S$$

$$\rightarrow 00B1010B$$

$$\rightarrow 00B10101$$

$$\rightarrow 00110101$$

$$S \Rightarrow 0B \Rightarrow 00BB \Rightarrow 001B \Rightarrow 0011S \Rightarrow 0110B \Rightarrow 001101S \Rightarrow 00110101$$

$$S \Rightarrow 0B \Rightarrow 00BB \Rightarrow 00B1S \Rightarrow 00B10B \Rightarrow 00B101S \Rightarrow 00B1010B \Rightarrow 00B10101 \Rightarrow 00110101$$

(iii) Deriv

0

Fig. : D

Q.14 Convert th

$$S \rightarrow AI$$

$$A \rightarrow BS$$

$$B \rightarrow S$$

Ans. $S \rightarrow AI$

$$A \rightarrow BS$$

$$B \rightarrow S$$

Step 1 : Re
productions are

$$A_1 \rightarrow$$

$$A_2 \rightarrow$$

$$A_3 \rightarrow$$

Step 2 : (a)

$$A_1 \rightarrow$$

(b) A_2 proc(c) $A_3 \rightarrow a$ (d) $A_3 \rightarrow$ (replacing $A_1 \rightarrow$

$$A_3 \rightarrow$$

Step 3 : A_3

$$A_3 \rightarrow$$

$$A_3 \rightarrow$$

Introducing

The resulti

$$A_3 \rightarrow$$

$$A_3 \rightarrow$$

$$Z \rightarrow$$

Q.17 Find a grammar in Chomsky Normal Form equivalent to $S \rightarrow aAbB$, $A \rightarrow aA/a$, $B \rightarrow bB/b$.

[R.T.U. 2010, Raj. Univ. 2005, 2004, 2002]

Ans. As there are no unit productions or null productions we need not carry out of elimination of unit productions or null production. We proceed to next steps.

Let $G_1 = (V'_N \{a, b\}, P_1, S)$, where P_1 and V'_N are constructed as follows:

(i) $A \rightarrow a$, $B \rightarrow b$ are added to P_1

(ii) $S \rightarrow aAbB$, $A \rightarrow aA$, $B \rightarrow bB$ yield $S \rightarrow C_a A C_b B$,
 $A \rightarrow C_a A$, $B \rightarrow C_b B$, $C_a \rightarrow a$, $C_b \rightarrow b$.
 $V'_N = \{S, A, B, C_a, C_b\}$.

P_1 consists of

$S \rightarrow C_a A C_b B$, $A \rightarrow C_a A$, $B \rightarrow C_b B$, $C_a \rightarrow a$,
 $C_b \rightarrow b$, $A \rightarrow a$, $B \rightarrow b$.

$S \rightarrow C_a A C_b B$ is replaced by $S \rightarrow (C_a C_1, C_1 \rightarrow AC_2, C_2 \rightarrow C_b B)$. The remaining productions in P_1 are added to P_2 .
 Let

$G_2 = (\{S, A, B, C_a, C_b, C_1, C_2\}, \{a, b\}, P_2, S)$,
 where P_2 consists of $S \rightarrow C_a C_1$, $C_1 \rightarrow AC_2$, $C_2 \rightarrow C_b B$,
 $A \rightarrow C_a A$, $B \rightarrow C_b B$, $C_a \rightarrow a$, $C_b \rightarrow b$, $A \rightarrow a$, and $B \rightarrow b$.
 G_2 is in CNF and equivalent to the given grammar.

Q.18 Find a grammar in GNF equivalent to the grammar

$E \rightarrow E+T \mid T, T \rightarrow T * F \mid F, F \rightarrow (E) \mid a$

[R.T.U. 2009]

Ans. $E \rightarrow E+T \mid T$

$T \rightarrow T * F \mid F$

$F \rightarrow (E) \mid a$

GNF (Greibach Normal Form)

GNF is defined as the normal form of a grammar whose production are in the format

$A \rightarrow a\alpha$ where $a \in \Sigma$

$\alpha \in (V_N \cup \Sigma)^*$

α may be Λ

In the given question,

Terminal symbols $\Sigma = \{a\}$

Non-terminals $V_N = \{E, T, F, +, *, (,), \}$
 $A_1, A_2, A_3, A_4, A_5, A_6, A_7$

We convert all V_N into A_1, A_2, A_3

Now, the production rules are

$A_1 \rightarrow A_1 A_4 A_2 \mid A_2$ (1)

$A_2 \rightarrow A_2 A_3 A_3 \mid A_3$ (2)

$A_3 \rightarrow A_6 A_1 A_7 \mid a$ (3)

Since $A_3 \rightarrow a$ (Required form)

$A_2 \rightarrow A_3 \Rightarrow A_2 \rightarrow a$ (Required form)

and $A_1 \rightarrow A_2 \Rightarrow A_1 \rightarrow a$ (Required form)

Now, from (1)

$A_1 \rightarrow A_1 A_4 A_2$

From (iii) $A_1 \rightarrow a$

$A_1 \rightarrow a A_4 A_2$ (Required form)

Similarly (2)

$A_2 \rightarrow A_2 A_3 A_3$

$A_2 \rightarrow a A_3 A_3$ (Required form)

Now

$A_3 \rightarrow A_6 A_1 A_7$

$A_3 \rightarrow A_6 a A_7$

Hence, the converted grammar in GNF is

$A_1 \rightarrow a A_4 A_2 \mid a$

$A_2 \rightarrow a A_3 A_3 \mid a$

$A_3 \rightarrow a$

In the original symbols

$E \rightarrow a + T \mid a$

$T \rightarrow a * F \mid a$

$F \rightarrow (a) \mid a$

Q.19 Find a grammar in CNF equivalent to the grammar

$S \rightarrow \sim S \mid [S \supset S] \mid p \mid q$ (S being the only variable).

[R.T.U. 2009]

Ans. $S \rightarrow \sim S \mid [S \supset S] \mid p \mid q$

$V_N = \{S\}$

$\Sigma = \{\sim, [, \supset,], p, q\}$

$S \rightarrow$ Start symbol production

$S \rightarrow \sim S$

$S \rightarrow [S \supset S]$

$S \rightarrow p$

$S \rightarrow q$

Given context free grammar is free from null variable, unit production and useless symbol.

Step 1 : Eliminate null production.

Result : In the grammar no null variable so ignore this step.

Step 2 : Eliminate unit production.

Result : There is no unit production so ignore this step also.

PUSHDOWN AUTOMATON

PREVIOUS YEARS QUESTIONS

PART-A

Q.1 Write three properties of CFL.

Ans. Properties of Context Free Languages

- (i) The reverse of a context-free language is context-free, but the complement need not be.
- (ii) Every regular language is context-free because it can be described by a regular grammar.
- (iii) The intersection of a context-free language and a regular language is always context-free.

Q.2 Give definition of PDA.

Ans. In the theory of computation, a branch of theoretical computer science, a pushdown automaton (PDA) is a type of automation that employs a stack. Pushdown automata are used in theories about what can be computed by machines. They are more capable than finite-state machines but less capable than Turing machines.

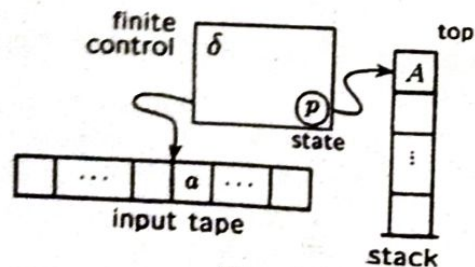
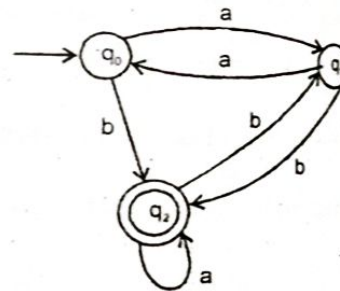


Fig.

PART-B

Q.3 Check whether the strings abb , aba and aab are accepted by transition graph or not.

Ans. Let us assume the transition graph as following



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

$$I.S. = q_0$$

$$\text{Final state} = q_2$$

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

Take our first string abb ;

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

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

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

and q_1 is not final state so this string is not accepted

Now take the second string for the given transition system aba ;

TOC-28

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

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

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

So q_2 is final state, this string is accepted. Now take the third string for the given

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

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

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

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

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

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

Since q_2 is not final state, so this string is not accepted by this transition graph.

Q.4 Define pushdown automaton and illustrate the move relation in a

Ans. Pushdown Automaton



Fig. Model of Pushdown Automaton

Definition : A pushdown automaton is a system consisting of

- (i) A finite nonempty set of states Q .
- (ii) A finite nonempty set of input symbols Σ .
- (iii) A finite nonempty set of pushdown symbols Γ .
- (iv) A special pushdown symbol ϵ called the pushdown state denoted by ϵ .
- (v) The set of final states, a subset of Q .
- (vi) The transition function δ from $Q \times (\Sigma \cup \epsilon) \times \Gamma$ to a set of finite subsets of $Q \times \Gamma^*$.

Symbolically, a PDA is a 7-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F, \epsilon)$.

Let us assume eq. (4) when the number of moves is n. Assume

$$(q, uv, S) \vdash^n (q, v, \alpha) \dots (5)$$

The last move in eq. (5) is obtained either from $(q, \lambda, A) \vdash (q, \lambda, \alpha')$ or $(q, \alpha, \alpha') \vdash (q, \lambda, \lambda)$. In the first case (5) can be split as

$$(q, uv, S) \vdash^n (q, v, \alpha^2 A) \vdash (q, v, \alpha_2 \alpha_1) = (q, v, \alpha)$$

By induction hypothesis, $S \Rightarrow uA\alpha_2$, and the last move is induced by $A \rightarrow \alpha_1$.

Thus

$S \Rightarrow uA\alpha_2$ implies $\alpha_1 \alpha_2 = \alpha$. So $S \Rightarrow uA\alpha_2 \Rightarrow u\alpha_1 \alpha_2 = u\alpha$.

In the second case (5) can be split as

$$(q, uv, S) \vdash^n (q, av, \alpha a) \vdash (q, v, \alpha)$$

Also $u = u'a$ for some $u' \in \Sigma$.

So $(q, u'av, S) \vdash^n (q, av, \alpha a)$ implies (by induction hypothesis) $S \Rightarrow u'a\alpha = u\alpha$. Thus in both cases we have shown that $S \Rightarrow u\alpha$. By the principle of induction eq. (4) is true.

Now we can prove that if $w \in N(M)$ then

$w \in L(G)$. As $w \in N(A)$,

$(q, w, S) \vdash^n (q, \lambda, \lambda)$. By taking $u = w, v = \lambda, \alpha = \lambda$ and

applying (4), we get $S \Rightarrow w\lambda = w$

i.e. $w \in L(G)$.

Thus $L(G) = N(M)$

Q.6 Explain the steps involving in conversion from context free grammar to pushdown automata with example. [R.T.U. 2013]

Ans. Suppose we have to construct a PDA A equivalent to the following CFG

$$\delta \rightarrow 0BB$$

$$B \rightarrow 0S/1S/0$$

Now, we can define PDA A as follows:

$$A = (\{q\}, \{0, 1\}, \{S, B, 0, 1\}, \delta, q, s, \phi)$$

Where δ is defined by the following rules:

$$R_1 : \delta(q, \wedge, S) = \{(q, 0BB)\}$$

$$R_2 : \delta(q, \wedge, B) = \{(q, 0S), (q, 1S), (q, 0)\}$$

$$R_3 : \delta(q, 0, 0) = \{(q, \wedge)\}$$

$$R_4 : \delta(q, 1, 1) = \{(q, \wedge)\}$$

Note : Rules R_1 to R_4 generated according to given CFG, for example

We generated rule 2 (R_2) for

$$B \rightarrow 0S/1S/0$$

Now if we want to test whether a string is $N(A)$ or not, we can test as follows :

For example,

We want to test whether 010000 is in $N(A)$.

| | |
|------------------------|----------|
| (q, 010000, S) | by R_1 |
| (q, 010000, 0BB) | by R_3 |
| (q, 10000, BB) | by R_2 |
| (q, 10000, 1SB) | by R_4 |
| (q, 0000, SB) | by R_1 |
| (q, 0000, 0BBB) | by R_3 |
| (q, 000, BBB) | by R_2 |
| (q, 000, 000) | by R_3 |
| (q, \wedge, \wedge) | |

Thus

$$010000 \in N(A)$$

Q.7 Convert the given PDA to CFG

$$A = (\{q_0, q_1\}, \{a, b\}, \{z_0, z_1\}, S, q_0, z_0, \phi)$$

S Ps given by

$$S(q_0, b, z_0) = (q_0, zz_0)$$

$$S(q_0, n, z_0) = (q_0, n)$$

$$S(q_0, b, z_1) = (q_0, zz_1)$$

$$S(q_0, a, z_1) = (q_1, z_1)$$

$$S(q_0, b, z_1) = (q_1, n)$$

$$S(q_1, a, z_0) = (q_0, z_0)$$

[R.T.U. 2012]

Ans. $G = (V_N, \{a, b\}, P, S)$

Where V_N consists of $S, [q_0, z_0, q_0]$

$$[q_0, z_0, q_1], [q_0, z, q_0], [q_0, z_1, q_1], [q_1, z_0, q_0],$$

$$[q_1, z_0, q_1], [q_1, z, q_0], [q_1, z_1, q_1]$$

The productions are

$$P_1 : S \rightarrow [q_0, z_0, q_0]$$

$$P_2 : S \rightarrow [q_0, z_0, q_1]$$

$\delta(q_0, b, z_0) = \{(q_0, z z_0)\}$ yields

$$P_3 : [q_0, z_0, q_0] \rightarrow b[q_0, z, q_0][q_0, z_0, q_0]$$

$$P_4 : [q_0, z_0, q_0] \rightarrow b[q_0, z, q_1][q_1, z_0, q_0]$$

$$P_5 : [q_0, z_0, q_1] \rightarrow b[q_0, z, q_0][q_0, z_0, q_1]$$

$$P_6 : [q_0, z_0, q_1] \rightarrow b[q_0, z, q_1][q_1, z_0, q_1]$$

$\delta(q_0, \wedge, z_0) = \{(q_0, \wedge)\}$ gives

$$P_7 : [q_0, z_0, q_0] \rightarrow n$$

$\delta(q_0, b, z_1) = \{(q_0, zz_1)\}$ gives

$$P_8 : [q_0, z, q_0] \rightarrow b[q_0, z, q_0][q_0, z, q_0]$$

$$P_9 : [q_0, z, q_0] \rightarrow b[q_0, z, q_1][q_1, z, q_0]$$

$$P_{10} : [q_0, z, q_1] \rightarrow b[q_0, z, q_0][q_0, z, q_1]$$

$$P_{11} : [q_0, z, q_1] \rightarrow b[q_0, z, q_0][q_0, z, q_1]$$

$\delta(q_0, a, z_1) = \{(q_1, z_1)\}$ yields

$$P_{12} : [q_0, z, q_0] \rightarrow a[q_1, z, q_0]$$

$$P_{13} : [q_0, z, q_1] \rightarrow a[q_1, z, q_1]$$

$\delta(q_1, b, z) = \{(q_1, \wedge)\}$ gives

$$P_{14} : [q_1, z, q_2] \rightarrow b$$

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

$$P_{15} : [q_1, z_0, q_0] \rightarrow a[q_1, z_0, q_0]$$

$$P_{16} : [q_1, z_0, q_1] \rightarrow a[q_0, z_0, q_1]$$

$P_1 - P_{16}$ gives the productions in P.

Q.8 What is PDA, Explain. Construct PDA equivalent to $L = \{a^n b^{n+m} a^m / n, m \geq 0\}$. [R.T.U. 2012]

Ans. PDA : Refer to Q.4.

The PDA A accepting $\{a^n b^{n+m} a^m / n, m \geq 0\}$ is defined as follows :

$$A = (\{q_0, q_1\}, \{a, b\}, \{a, z_0\}, \delta, q_0, z_0, \phi)$$

Where δ is defined by

$$R_1 : \delta(q_0, a, z_0) = \{(q_0, az_0)\}$$

$$R_2 : \delta(q_0, a, a) = \{(q_0, aa)\}$$

$$R_3 : \delta(q_0, b, a) = \{(q_1, a)\}$$

$$R_4 : \delta(q_1, b, a) = \{(q_1, a)\}$$

$$R_5 : \delta(q_1, a, a) = \{(q_1, \wedge)\}$$

$$R_6 : \delta(q_1, \wedge, z_0) = \{(q_1, \wedge)\}$$

We start storing a's until a b occurs. When the current input symbol is b, the state changes, but no change in PDS occurs. Once all the b's in the input string are exhausted, the remaining a's are erased. Using R_6 , z_0 is erased. So

$$(q_0, a^n b^{n+m} a^m / n, z_0) \xrightarrow{*} \dots (q_1, \wedge, z_0) \xrightarrow{*} \dots (q_1, \wedge, \wedge)$$

This means that $a^n b^{n+m} a^m \in N(A)$. We can show that

$$N(A) = \{a^n b^{n+m} a^m / n \geq 0\}$$

By using Rule

Define $G = (V_N, \{a, b\}, P, S)$ where V_N consisting of $[q_0, z_0, q_0], [q_1, z_0, q_0], [q_0, a, q_0], [q_1, a, q_0], [q_0, z_0, q_1], [q_1, z_0, q_1], [q_0, a, q_1], [q_1, a, q_1]$

The production in P are constructed as follows :

The S - production are

$$P_1 : S \rightarrow [q_0, z_0, q_0]$$

$$P_2 : S \rightarrow [q_0, z_0, q_1]$$

$$\delta(q_0, a, z_0) = \{(q_0, az_0)\} \text{ induces}$$

Q.15 Write short note on Pumping Lemma for CFG
[Raj. Univ. 2007]

Lemma : Let G be a context-free grammar in CNF and T be a derivation tree in G . If the length of the longest path in T is less than or equal to k , then the yield of T is of length less than or equal to 2^{k-1} .

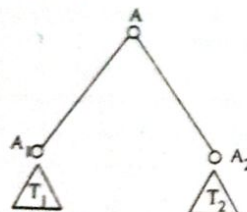


Fig. Tree T with subtrees T_1 and T_2 .

Assume the result for $k-1$ ($k > 1$). Let T be an A -tree with a longest path of length less than or equal to k . As $k > 1$, the root of T has exactly two sons with labels A_1 and A_2 . The two subtrees with the two sons as roots have the longest paths of length less than or equal to $k-1$.

If w_1 and w_2 are their yields, then by induction hypothesis, $|w_1| \leq 2^{k-2}$, $|w_2| \leq 2^{k-2}$.

A) So the yield of $T = w_1 w_2$, $|w_1 w_2| \leq 2^{k-2} + 2^{k-2} = 2^{k-1}$. By the principle of induction, the result is true for all A-trees, and hence for all derivation trees.

- (i) Every $z \in L$ with $|z| \geq n$ can be written as $uvwxy$ for some strings u, v, w, x, y .
- (ii) $|vx| \geq 1$
- (iii) $|vwx| \leq n$
- (iv) $uv^kwx^ky \in L$ for all $k \geq 0$

Proof: When $\Lambda \in L$; we consider $L - \{\Lambda\}$ and construct a grammar $G = (V_N, \Sigma, P, S)$ CNF generating $L - \{\Lambda\}$ (when $\Lambda \notin L$, we construct G in CNF generating L).

Let $|V_N| = m$ and $n = 2^m$. To prove that n is the required number, we start with $z \in L$, $|z| \geq 2^m$, and construct a derivation tree T (parse tree) of z . If the length of a longest path in T is at most m , by lemma, $|z| \leq 2^{m-1}$ (since z is the yield of T). But $|z| \geq 2^m > 2^{m-1}$. So T has a path, say Γ of length greater than or equal to $m+1$. Γ has at least $m+2$ vertices and only the last vertex is a leaf. Thus in Γ all the labels except the last one are variables. As $|V_N| = m$ some label is repeated.

We choose a repeated label as follows: We start with the leaf of Γ and travel along Γ upwards. We stop when some label, say B , is repeated (Among several repeated labels, B is the first). Let v_1 and v_2 be the vertices with label B , v_1 being nearer the root. In Γ , the portion of the path from v_1 to the leaf has only one label, namely B , which is repeated, and so its length is at most $m+1$.

Let T_1 and T_2 be the subtrees with v_1, v_2 as roots and z_1, w as yields, respectively. As Γ is a longest path in T , the portion of Γ from v_1 to the leaf is a longest path in T_1 and of length at most $m+1$. By lemma $|z_1| \leq 2^m |z_1| \leq 2^m$ (since z_1 is the yield of T_1).

For better understanding, we illustrate the construction for the grammar whose productions are

$S \rightarrow AB, A \rightarrow aB \mid a, B \rightarrow bA \mid b$, as in Fig.

In the figure,

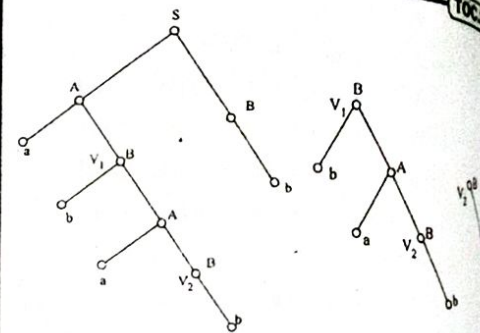
$$\Gamma = S \rightarrow A \rightarrow B \rightarrow A \rightarrow B \rightarrow b$$
$$z = ababb, z_1 = bab, w = b$$
$$v = ba, x = \Lambda, u = a, y = b$$


Fig. : Tree T and its subtrees T_1 and T_2

As z and z_1 are the yields of T and a proper subtree of T , we can write $z = uz_1y$. As z_1 and w are the yields of T_1 and a proper subtree T_2 of T_1 , we can write $z_1 = wz_2$. Also $|vwx| > |w|$. So $|vx| \geq 1$. Thus, we have $z = uvwx^i y$ with $|vwx| \leq n$ and $|vx| \geq 1$. This proves the points (i)-(iii) of the theorem.

As T is an S-tree and T_1, T_2 are B-trees, we get

$$S \xrightarrow{*} uBy, B \xrightarrow{*} vBx \text{ and}$$
$$B \xrightarrow{*} w, AsS \xrightarrow{*} uBy \Rightarrow wvy, uv^0wx^0y \in L$$

For $k \geq 1, S \Rightarrow uBy \Rightarrow uv^k Bx^k y \Rightarrow uv^k wx^k y \in L$.
proves the point (iv) of the theorem.

Corollary : Let L be a context-free language and n be the natural number obtained by using the pumping lemma. Then (i) $L \neq \phi$ if and only if there exists $w \in L$ with $n \leq |w| < 2n$ and (ii) L is infinite if and only if there exists $x \in L$ such that

Proof : (i) We have to prove the 'only if' part. If $z \in L$ with $|z| \geq n$, we apply the pumping lemma to write $z = uvw$ where $1 \leq |u| \leq n$. Also $uwv \in L$ and $|uwv| < |z|$. Applying pumping lemma repeatedly, we can get $z' \in L$ such that $|z'| < n$. Thus (i) is proved.

(ii) If $z \in L$ such that $n \leq |z| < 2n$, by pumping lemma we can write $z = uvwxy$. Also, $uv^iwx^iy \in L$ for all $i \geq 0$. Thus we get an infinite number of elements in L . Conversely, if L is infinite, we can find $z \in L$ with $|z| \geq n$. If $|z| < 2n$, there is nothing to prove. Otherwise, we can apply pumping lemma to write $z = uvwxy$ and get $uvw^iwx^iy \in L$.

TOC.30
time we appl
and the decre

So, we ultim
moves (

Note :

Note :
the length of
well (refer to

The cor
a given cont
algorithms a

We use
not a conte
free. By app

The pr
steps :

Step 1
number ob

Step 2
using the p

Step :

a contradic



Q.16 Ho

Ans. As
store or 1

So
T

store, i.e.

L
V

Gramm

TURING MACHINES

4

PREVIOUS YEARS QUESTIONS

This is why we introduce the notion of a universal turing machine (UTM), which along with the input on the tape, take in the description of a machine M. The UTM can go on then to simulate M on the rest of the contents of the input tape. A universal turing machine can thus simulate any other machine

machine? [R.T.U. 2019]

Q.2 What is Turing Machine?

[R.T.U. 2015]

OR

Explain turing machine in brief.

Ans. Turing Machine : The Turing Machine (TM) is a simple mathematical model of a general purpose computer. In other words, Turing machine model is the computing power of computer i.e. the Turing machine is capable of performing any calculation which can be performed by any computing machine.

The Turing machine can be thought of as a finite state automaton connected to R/W (read / write) head. It has an input tape which is divided into a number of cells.

Q.3 Explain the properties of context – sensitive language

[R.T.U. 2015]

Ans. Context Sensitive Language

- Context sensitive languages are also called type 1 grammar.
- Context sensitive languages have both left and right context.
- The production is of the form :

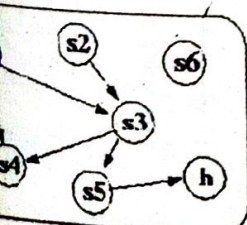
$$\phi\alpha\psi \rightarrow \phi\beta\psi$$

Here

$$\phi \rightarrow \text{left context}$$

$$\psi \rightarrow \text{right context}$$

mathematical tool equivalent to the Turing machine. It was introduced by the mathematician Alan Turing. It is then the most widely used model of computation. The input output relation that is computed is given in binary form on the input tape. It consists of the contents of the input tape. The contents of the tape change as the machine runs. M, also called a finite state machine. The FSM is determined by the transitions between states.



The input tape is divided into cells. The character read from the tape and the character written on the tape (possibly the same character) are used to determine the next state. The character read from the tape (possibly the same character) is used to determine the next state.

bcd -
In this $S \rightarrow \Lambda$ is the right hand side (RHS) of the production.

The Turing machine model for context sensitive language :

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

Here

Q = Finite non empty set of states

Σ = Finite non empty set of input symbols

Γ = External symbols

δ → Transition function

q_0 = Initial state

Δ or b or $B = \Lambda$

F = Final state.

$\Gamma \rightarrow$ Maps $\Sigma \times Q$ to $\Gamma \times Q \times \Delta$

Q.4 Prove the transitive closure of a context sensitive language.

Ans. Transpose : Transpose of a sequence of symbols is the sequence of symbols in reverse order.

$$\text{Let } s = s_1 s_2 \dots s_n \\ s^T = s_n s_{n-1} \dots s_1$$

It should be noted that the transpose of a sequence of symbols is the sequence of symbols in reverse order.

$$\text{Suppose } L = \{ s_1 s_2 \dots s_n \}$$

$$L^T = \{ s_n s_{n-1} \dots s_1 \}$$

Here $W = 12$ is the length of the string.

Transpose (W^T) is the reverse of W .

Q.5 Define indexed language

Ans. Indexed language

They are described by a grammar where the non-terminals are recognized by nested strings.

Indexed languages are a class of languages. They qualify for the pumping lemma and hence satisfy many properties. They are not closed under intersection.

Q.6 Define union in context sensitive languages

strictly weaker device.

Q.8 Design a Turing machine over $\{1, b\}$ which can compute a concatenation function over $\Sigma = \{1\}$.

If a pair of words (w_1, w_2) is the input, the output has to be $w_1 w_2$.

[R.T.U. 2016]

Ans. Suppose w_1 is given as a string $w_1 = 11$ and $w_2 = 111$ and the two words are written in the input tape separated by a blank as

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| B | 1 | 1 | B | 1 | 1 | 1 | B |
|---|---|---|---|---|---|---|---|

Input tape

The output tape should be

| | | | | | | |
|---|---|---|---|---|---|---|
| B | 1 | 1 | 1 | 1 | 1 | B |
|---|---|---|---|---|---|---|

Output tape

When the blank symbol is found, it must be replaced by 1 and when the rightmost 1 is found, it is replaced by B. The tape head return to the starting position.

The machine M can be defined as :

$$M = (\{q_0, q_1, q_2, q_3, q_f\}, \{1\}, \{1, B\}, \delta, q_0, B, \{q_f\})$$

The moves of the Turing machine are shown below in the form of transition table :

TOC.42

Table : Transition Table

| Present State | 1 | B |
|-------------------|---------------|---------------|
| $\rightarrow q_0$ | $(q_0, 1, R)$ | $(q_1, 1, R)$ |
| q_1 | $(q_1, 1, R)$ | (q_2, B, L) |
| q_2 | (q_3, B, L) | — |
| q_3 | $(q_3, 1, L)$ | (q_f, B, R) |
| q_f | — | — |

The transition diagram corresponding to the above transition table is shown in following figure.

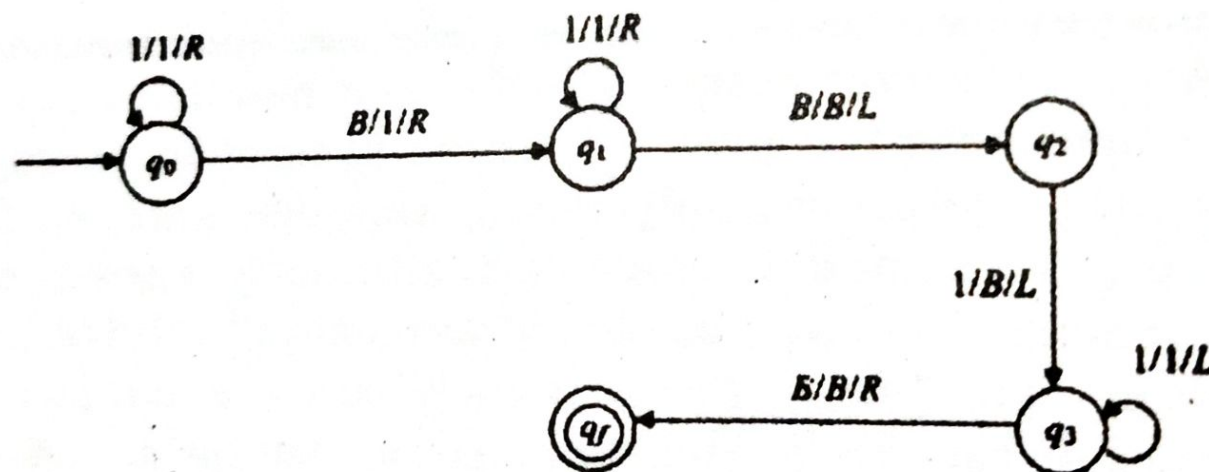


Fig. : Representation of Turing Machine of Transition Table

└ bb0

└ b q:

└ bbt

└ bbt

└ bbt

q₅ 000

Q.10

Ans.

the gi

a total

string

sieve

follow

then r

Step 3 : On scanning the left most 3, the R/W head replaces 3 by and moves to the right. M enters q_4 .
Step 4 : After scanning the rightmost 3, the R/W head moves to the left until it finds the left most 1. As a result, the left most 1, 2, 3 are replaced by b.
Step 5 : Steps 1 - 4 are repeated until all 1's, 2's and 3's are replaced by blanks.

Thus we can construct as-

| Present state | Input tape symbol | | | |
|-------------------|-------------------|-----------------|-----------------|-----------------|
| | 1 | 2 | 3 | \square |
| $\rightarrow q_1$ | $\square q_2 R$ | - | - | $\square q_1 R$ |
| q_2 | $1 q_2 R$ | $\square q_3 R$ | - | $\square q_2 R$ |
| q_3 | - | $2 q_3 R$ | $\square q_4 R$ | $\square q_3 R$ |
| q_4 | - | 0 | $3 q_5 L$ | $\square q_4 L$ |
| q_5 | $1 q_6 L$ | $2 q_5 L$ | - | $\square q_5 L$ |
| q_6 | $1 q_6 L$ | - | - | $\square q_1 R$ |
| q_7 | - | - | - | - |

This is the required solution.

Q.14 Write short note on recursive and recursively enumerable language.

[R.T.U. 2013, 2012, 2011, Raj. Univ. 2007, 06, 04, 03]

Ans. Recursive and Recursively Enumerable Language : In mathematics, logic and computer science, a recursively enumerable language is a type of formal language which is also called partially decidable or Turing-acceptable. It is known as a type-0 language in the Chomsky hierarchy of formal languages. The class of all recursively enumerable language is called RE.

Definitions

There exist three equivalent major definitions for the concept of a recursively enumerable language.

1. A recursively enumerable formal language is a recursively enumerable subset in the set of all possible words over the alphabet of the language.
2. A recursively enumerable language is a formal language for which there exists a Turing machine (or other computable function) which will enumerate all valid strings of the language. Note that, if the language is infinite, the enumerating algorithm provided can be chosen so that it avoids repetitions, since we can test whether the string produced for number n is "already" produced for a number which is less than n . If it is already produced, use the output for input $n+1$ instead (recursively), but again, test whether it is "new".

A recursively enumerable language is a formal language for which there exists a Turing machine (or other computable function) that will halt and accept when presented with any string in the language as input but may either halt and reject or loop forever when presented with a string not in the

Theory of Computation

language. Contrast this to recursive language, which require that the Turing machine halts in all cases.

Closure Properties

Recursively enumerable languages are closed under the following operations. That is, if L and P are two recursively enumerable language, then the following languages are recursively enumerable as well :

- The Kleene star L^* of L .
- The concatenation of L and P .
- The union.
- The intersection.

Note that recursively enumerable languages are not closed under set difference or complementation. The set difference $L-P$ may or may not be recursively enumerable. If L is recursively enumerable, then the complement of L is recursively enumerable if and only if L is also recursive. There are three possible outcomes of executing a Turing machine over a given input. The Turing machine may

- Halt and accept the input
- Halt and reject the input or
- Never halt.

A language is recursive if there exists a Turing machine that accepts every string of the language and rejects every string (over the same alphabet) that is not in the language.

Q.15 (a) Write short note on :

- (i) Halting problem
 - (ii) Multitape and Multi dimensional turing machine [R.T.U. 2012, 2011, Raj. Univ. 2007]
- (b) Design turing machine M that recognize the language $\{a^n b^n c^n / n \geq 1\}$ [R.T.U. 2012, 2011]

Ans.(a) (i) Halting Problem : For a given configuration of a TM, two cases arise :

- (a) The machine starting at this configuration will halt after a finite number of steps.
- (b) The machine starting at this configuration never halts no matter how long it runs. Given any TM, problem of determining whether it halts ever or not is called halting problem.

To solve the halting problem, we should have some mechanism to which given any functional matrix, input data type and initial configuration of the TM for which we want to detect, determine whether the process will ever halt or not. In reality one cannot solve the halting problem. The halting problem is unsolvable. That means, there exist no TM which can determine whether given TM 'T' will ever halt or not.

Reduction Technique is used to prove the undecidability of halting problem of Turing machine. We say that problem A is reducible to problem B if a solution to problem B can be used to solve problem A.

Example : If A is the problem of finding some root of $x^4 - 3x^2 + 2 = 0$ and B is the problem of finding some root of $x^2 - 2 = 0$, then A is reducible to B. As $x^2 - 2$ is a factor of $x^4 - 3x^2 + 2$, a root of $x^2 - 2 = 0$ is also a root of $x^4 - 3x^2 + 2 = 0$.

Note : If A is reducible to B and B is decidable then A is decidable. If A is reducible to B and A is undecidable, then B is undecidable.

Theorem : $HALT_{TM} = \{(M, w)\}$ The Turing machine M halts on input $\{w\}$ is undecidable.

Proof : We assume that $HALT_{TM}$ is decidable and get a contradiction. Let M_1 be the TM such that $T(M_1) = HALT_{TM}$ and let M_1 halt eventually on all (M, w) . We construct a TM M_2 as follows:

1. For $M_2, (M, w)$ is an input.
2. The TM M_1 acts on (M, w) .
3. If M_1 rejects (M, w) then M_2 rejects (M, w) .
4. If M_1 accepts (M, w) , simulate the TM, M on the input string w until M halts.
5. If M has accepted w , M_2 accepts (M, w) ; otherwise M_2 rejects (M, w) .

When M_1 accepts (M, w) (in step 4), the Turing machine M halts on w . In this case either an accepting state q or a state q' such that $\delta(q', a)$ is undefined till some symbol a in w is reached. In the first case (the first alternative of step 5) M_2 accepts (M, w) . In the second case (the second alternative of step 5) M_2 rejects (M, w) .

It follows from the definition of M_2 that M_2 halts eventually.

Also $T(M_2) = \{(M, w)\}$ The Turing machine accepts $w\} = A_{TM}$

This is a contradiction since A_{TM} is undecidable.

Ans. As the elements of $T(M)$ are given by path values of paths from q_0 to itself or from q_0 to q_1 (note that we have two final states q_0 and q_1), we can construct $T(M)$ by inspection.

As shown in figure, arrows do not come into q_0 , the paths from q_0 to itself are self-loops repeated any number of times. The corresponding path values are $0^i, i \geq 1$. As no arrow comes from q_2 to q_0 or q_1 , the paths from q_0 to q_1 are of the form $q_0 \dots \rightarrow q_0 \dots q_1 \rightarrow q_1$. The corresponding path values are $0^i 1^j$, where $i \geq 0$ and $j \geq 1$. As the initial state q_0 is also a final state, $\Lambda \in T(M)$. Thus,

$$T(M) = \{0^i 1^j \mid i, j \geq 0\}$$

$$\text{Hence, } T(M)^T = \{1^j 0^i \mid i, j \geq 0\}$$

The transition system M' is concerned as follows :

- (i) The initial states of M' are q_0 and q_1 .
- (ii) The (only) final state of M' is q_0 .
- (iii) The direction of the directed edges is reversed. M' is given in figure.

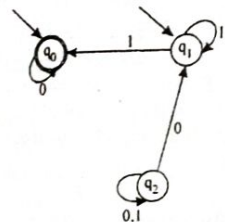


Fig. : Finite Automaton of $T(M)^T$
From (i) - (iii) it follows that

$$T(M') = T(M)^T$$

Hence, $T(M)^T$ is regular.

In above case, we can see by inspection that $T(M') = \{1^j 0^i \mid i, j \geq 0\}$. The strings of $T(M')$ are obtained as path values of paths from q_0 to itself or from q_1 to q_0 .

PART-C

Q.21 Explain Turing Machine with its various way of representation. [R.T.U. 2019]

OR

Explain turing machine with its various ways of representation. Draw diagram wherever required. [R.T.U. 2012]

Ans. (i) **Standard Turing Machine** : A turing machine M is defined by

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

Where

Q = Set of internal states

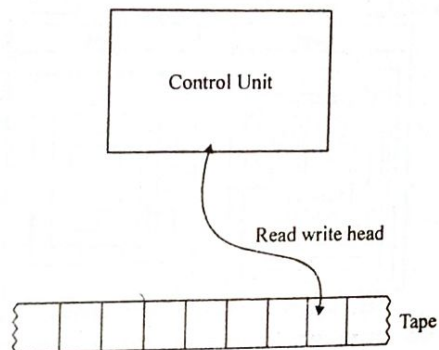
Σ = Input alphabet

Γ = Finite set of symbols

$\square \in \Gamma$ a special symbol called the blank

$q_0 \in Q$ is the initial state

$F \subseteq Q$ is the set of final state



(ii) **Turing Machine as Language Acceptor** : Turing machine can be viewed as acceptors in following sense. A string w is written on the tape, with blanks filling out the unused portion. The machine is started in the initial state q_0 with the read write head positioned on the leftmost symbol of w . If, after a sequence of moves, the Turing Machine enters a final state and halts, then w is considered to be accepted.

Let $M = (Q, \Sigma, \Gamma, \delta, q_0, \square, F)$ be a Turing machine. Then the language accepted by M is

$$L(M) = \{w \in \Sigma^* : q_0 w 1^* x_1 q_f x_2 \text{ for some } q_f \in F, x_1, x_2 \in \Gamma^*\}$$

(iii) **Turing Machine as Transducer** : Turing Machine are not only interesting as language acceptors, they provide us with a simple abstract model for digital computers in general. Since the primary purpose of a computer is to transform input into output, it acts as a transducer. If we want to model computer using Turing machine, we have to look at this aspect more closely.

We can view a Turing Machine transducer m as an implementation of a function f defined by

$$w = f(w)$$

provided that

$$q_0 w 1^* = q_f w$$

for some final state q_f

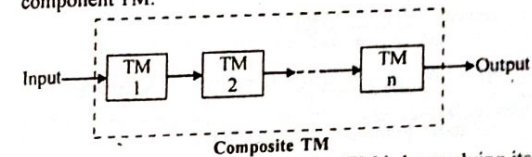
Theory of Computation

Definition : A function f with domain D is said to be Turing-computable or just computable if there exist some Turing machine $M = (Q, \Sigma, \Gamma, \delta, q_0, \square, F)$ such that

$$q_0 w \Gamma^* \vdash^* q_f f(w) \quad q_f \in F$$

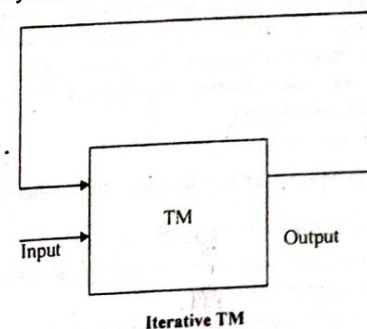
for also $w \in D$

(iv) **Composite Turing Machine (TM) and Iterated Turing Machine** : Two or more Turing Machine can be combined to solve a collection of simpler problems, so that the output of one Turing Machine forms the input to the next Turing Machine and so on. This is called as **Composition**. For realizing a compsite TM, the functional matrices of the component TMs are combined by increasing and relabelling I and suitably branching to an appropriate state rather than the halt state at the completion of the performance of each component TM.



Another way of having a combination TM is by applying its own output as input respectively. This is called as iteration or recursion.

The idea of composite TM gives rise to the concept of breaking the complicated job into number of jobs implementing each separately and then combining them together to get answer for the job required to be done. Therefore, we can divide a problem into simple jobs and design different TM, for each. Then we can take the composition of all TMs to get work done what we want initially. Modular programming is definitely influenced by CTM (Composite TM).



(v) **Universal TM (or UTM)** : Refer to Q.1.

(vi) **Multistack TM** : This symbols to the left of the head of the TM can be stored on one stack, while the symbols on the right of the head can be placed on the other stack. On each

said to be Turing
exist some Turing
at

closer to the top of the stack. This type is called as Multistack
Turing Machine.

Q.22 Explain Chomsky classification of language with the help of suitable example. [R.T.U. 2019, 13, 12]

OR

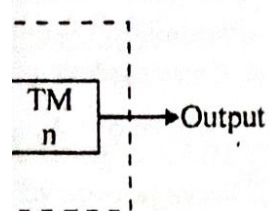
Write short note on Chomsky Hierarchy of languages in detail.

[R.T.U. 2016, 2014, 2011, Raj. Univ. 2007, 2006, 2004, 2003]

OR

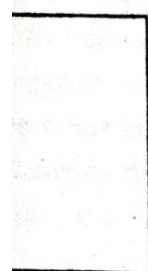
Discuss chomsky hierarchy in detail. [R.T.U. 2015]

M) and Iterated
g Machine can be
r problems, so that
he input to the next
l as **Composition**.
onal matrices of the
ing and relabelling
te state rather than
rformance of each



A is by applying its
alled as iteration or

e to the concept of
f jobs implementing
em together to get
Therefore, we can
gn different TM, for
of all TMs to get
lar programming is
(TM).



Output

to Q.1.

he left of the head of
the symbols on the
ther stack. On each

Ans. According to Chomsky (Name of Scientist) there are four types of grammar:

- (1) Type-3 grammar or regular grammar
- (2) Type-2 grammar or context free grammar
- (3) Type-1 grammar or context sensitive grammar
- (4) Type-0 grammar or unrestricted grammar

(1) Type-3 Grammar or Regular Grammar :

These types of grammar follows the following rule of production:

$m \rightarrow n$ is a production rule for regular grammar where,
 $m \in V_N$ and $n \in \{\lambda, a, b, aA, bB, bA, aB\}$
 for $V_N = \{A, B\}$
 $\Sigma = \{a, b\}$

and A is starting non terminal (start symbol).

Example 1 : Production rules for regular grammar

- | | | |
|-----------------------------|---------------------------|-------------------------|
| (i) $A \rightarrow \lambda$ | (ii) $A \rightarrow a$ | (iii) $A \rightarrow b$ |
| (iv) $A \rightarrow aA$ | (v) $A \rightarrow bB$ | (vi) $B \rightarrow b$ |
| (vii) $B \rightarrow aA$ | (viii) $B \rightarrow aB$ | |

Solution : All of the (i) to (viii) are regular grammar.

In regular grammar left side of production will always be only one variable and in right side there will be single terminal or one non terminal (variable) followed by terminal or λ only.

Example 2 :

Given $V_N = \{A, B, C\}$

$\Sigma = \{0, 1\}$, A is start symbol. Productions rules are following:

- | | | |
|-----------------------------|------------------------|------------------------|
| (a) $C \rightarrow \lambda$ | (b) $A \rightarrow BC$ | (c) $A \rightarrow 0C$ |
| (d) $AB \rightarrow 0B$ | (e) $AC \rightarrow 0$ | (f) $A \rightarrow 01$ |

Solution : (a), (c) are production for regular grammar.

(b) $A \rightarrow BC$ are not in RG (Regular Grammar) because right side

(d) $AB \rightarrow 0B$ are not in RG because left side must contain only one non terminal (variable)

(e) $AC \rightarrow 0$ (same as (d)) (not in RG)

(f) $A \rightarrow 01$ are not in RG (Same logic for b).

(2) Type-2 Grammar or Context Free Grammar:

These types of grammar follows the following rule of production:

$m \rightarrow n$ is a production rule for context free grammar (CFG) where
 $m \in V_N$

and $n \in (V_N \cup \Sigma)^*$

Example 3 : Following production are under the context free grammar (CFG) for $V_N = \{S, A, B\}$ and $\Sigma = \{0, 1\}$.
 Production rules are following-

- (a) $S \rightarrow 0$ (b) $S \rightarrow 0A$ (c) $A \rightarrow \wedge$
 (d) $S \rightarrow 0SA$ (e) $S \rightarrow 0AB$ (f) $S \rightarrow B$

Solution : Here from (a) to (f) left side of production are single variable and right side of production are any combination of terminal and variable means $(V_N \cup \Sigma)^*$ or $(S, A, B, 0, 1)^*$.

In the above example

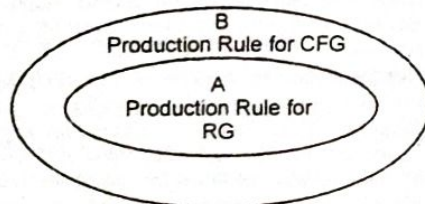
- (a) $S \rightarrow 0$ (b) $S \rightarrow 0A$ (c) $A \rightarrow \wedge$

are also follow the production rules for regular grammar or type-3 grammar. So we can say that production rules for regular grammar is subset of production rule for context free grammar

= Production rule for RG \subseteq Production rule for CFG

Diagrammatically we can understand this.

So, any grammar that will be regular will also be context free grammar but the converse is not always true i.e., some production which follows the CFG property ($A \rightarrow aBB$) do not follow the regular grammar property.



Example 4 :

Given $V_N = \{A, B, C\}$
 $\Sigma = \{0, 1\}$, A is start symbol

- (a) $A \rightarrow 0AB$ (b) $A \rightarrow AB0$ (c) $A \rightarrow A0C$
 (d) $A \rightarrow 0CB$ (e) $A \rightarrow 0$ (f) $AB \rightarrow 0C$
 (g) $BC \rightarrow 1A$ (h) $AI \rightarrow 1B$

Solution : From (a) to (e) are production rule for context free grammar because left hand side are only one variable and right side have element of $(V_N \cup \Sigma)^*$.

(f), (g), (h) are not production rule for context free grammar because left hand side are not only one variable or non terminal.

(3) Type-1 Grammar or Context Sensitive Grammar: This types of grammar follows the following rule of production:

$\phi A \psi \rightarrow \phi \alpha \psi$ if $\alpha \neq \lambda$ and erasing of A is not permitted.

$$\text{Example 5 : } \frac{a A b c D}{\phi A \psi} \rightarrow \frac{a b c b c D}{\phi \alpha \psi}$$

In the above grammar if $A = A$ and $\psi = b c D$, $\phi = a$, $\alpha = b c$ then it follows:

$$\phi A \psi \rightarrow \phi \alpha \psi \text{ where } \alpha \neq \lambda$$

So it is context sensitive grammar

$$\text{Example 6 : } \frac{A B}{\phi A} \rightarrow \frac{A b B c}{\phi \alpha}$$

When we add λ in the above grammar on both sides then there will be no difference between added ϕ grammar and original grammar so we can write above grammar as:

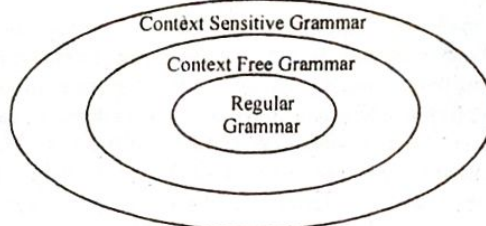
$$\frac{A B \lambda}{\phi A \psi} \rightarrow \frac{A b B c \lambda}{\phi \alpha \psi}$$

where $\phi = A$, $\psi = \lambda$, $A = B$ and $\alpha = b B c$ which is not null.

So above grammar is context sensitive.

Production set for RG \subseteq Production set for CFG

Production set for CFG \subseteq Production set for CSG



(4) Type - 0 Grammar or Unrestricted Grammar:

Every production which follow the production rule for grammar are called unrestricted grammar. So, unrestricted grammar definition is same as definition of grammar which we have discussed in above section.

$$\Rightarrow m \rightarrow n$$

where $m \in (V_N \cup \Sigma)^*$ which contain at least one variable

and $n \in (V_N \cup \Sigma)^*$ So, $ab \rightarrow cd$

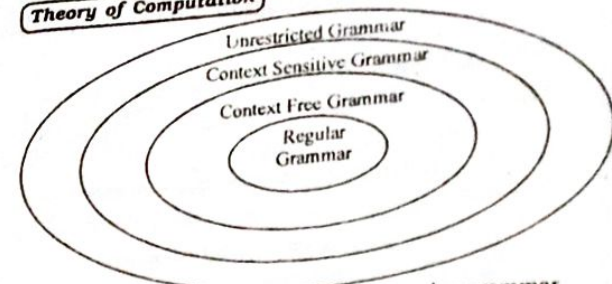
are not Grammar on unrestricted grammar because left side does not contain variables.

* It is called unrestricted grammar because this grammar does not follow any particular types.

So we can say

Production set for RG \subseteq Production set for CFG \subseteq Production set for CSG \subseteq Production set for unrestricted grammar.

Diagrammatically:



Note : type 3 grammar is also regular grammar.

The following fig describes the relation b/w the four types of languages and automata:

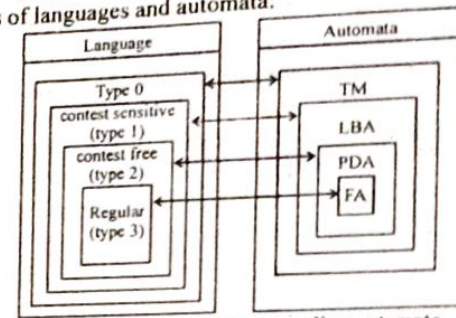


Fig. : Language and corresponding automata

Q.23 Make the comment on the following statement finite state machine with two stack is as powerful as Turing Machine. [R.T.U. 2010, Raj. Univ. 2007]

Ans. Pushdown automata is differ from finite state machines in two ways

1. They can use the top of the stack to decide which transition to take.
2. They can manipulate the stack as part of performing a transition.

Pushdown automata choose a transition by indexing a table by input signal, current state and the symbol at the top of the stack. This means that those three parameters completely determine the transition path that is chosen. Finite state machines just look at the input signal and the current state : they have no stack to work with. Pushdown automata add the stack as a parameter for choice.

Pushdown automata can also manipulate the stack, as part of performing a transition. Finite state machines choose a new state, the result of following the transition. The manipulation can be to push a particular symbol to the top of the stack, or to pop off the top of the stack. The automaton can alternatively ignore the stack and leave it as it is. The choice of manipulation (or no manipulation) is determined by the transition table.

and s
anoth

instea
equi
auton
pushd

gramm
pushd
the gr
autom
harder
a conti
the au
the gra
Defini

- A f
- A fi
- A fi
- A s
- A s
- the
- The
- The
- set
- Syn
- F).

the inte
with as
to q, po
denotes
relation
letter fr
In many
(equival

Q.6 Explain the Cook's theorem with suitable example. [R.T.U. 2016]

OR

What is the use of Cook's theorem? Prove it with an example. [R.T.U. 2012, 2011, 2010, Raj. Univ. 2003]

OR

What is Cook's theorem? Explain. [R.T.U. 2013, 2009]

Ans. Cook's theorem states that satisfiability is in P if and only if $P = NP$. If $P = NP$, then satisfiability is in P. It remains to be shown that if satisfiability is in P, then $P = NP$.

Proof : To do this, we show how to obtain from any polynomial time nondeterministic decision algorithm A and input I a formula Q(A, I) such that Q is satisfiable if A has a successful termination with input I.

If the length of I is n and the time complexity of A is $p(n)$ for some polynomial $p()$, then the length of Q is $O(p^3(n) \log n) = O(p^4(n))$. The time needed to construct Q is also $O(p^3(n) \log n)$. A deterministic algorithm Z to determine the outcome of A on any input I can be easily obtained.

Algorithm Z simply computes Q and then uses a deterministic algorithm for the satisfiability problem to determine whether Q is satisfiable.

If $O(p(m))$ is the time needed to determine whether a formula of length m is satisfiable, then the complexity of Z is $O(p^3(n) \log n + q(p^3(n) \log n))$. If satisfiability is in P, then $q(m)$ is a polynomial function of m and the complexity of Z becomes $O(r(n))$ for some polynomial $r()$. Hence, if satisfiability is in P, then for every nondeterministic algorithm A in NP we can obtain a deterministic Z in P. So, the above construction shows that if satisfiability is in P, then $P = NP$.

Before going into the construction of Q from A and I, we make some simplifying assumptions on our nondeterministic machine model and on the form of A. These assumptions do not in any way alter the class of decision problems in NP or P. The simplifying assumptions are as follows :

1. The machine on which A is to be executed is word oriented. Each word is w bits long. Multiplication, addition, subtraction, and so on, between numbers one word long take one unit of time. If numbers are longer than a word, then the corresponding operations take at least as many units as the number of words making up the longest number.

2. A simple expression is an expression that contains at most one operator and all operands are simple variables (i.e.,

no array variables are used). Some simple expressions are $B + C$, D or E , and F . We assume that all assignments are in one of the following forms:

- (a) (simple variable) := (simple expression)
- (b) (array variable) := (simple expression)
- (c) (simple variable) := (simple variable)
- (d) (simple variable) := (array variable)
- (e) (simple variable) := Choice(S)

where S is a finite set $\{S_1, S_2, \dots, S_k\}$ or $1, u$. In the last case the function chooses an integer in the range $\{1, \dots, u\}$.

Indexing within an array is done using a simple expression. All index values are positive. Only one-dimensional arrays are allowed. Clearly, all assignment statements falling into one of the above categories can be replaced by a set of statements of these types. Hence, this restriction does not alter the class NP.

3. All variables in A are of type integer or boolean.

4. Algorithm A contains no read or write statements. The only input to A is via its parameters. At the time A is invoked all variables (other than the parameters) have value zero or false if boolean.

5. Algorithm A contains no constants. Clearly, all constants in any algorithm can be replaced by new variables. The new variables can be added to the parameter list of A and the constants associated with them can be a part of the input.

6. In addition to simple assignment statements, A is allowed to contain only the following types of statements:

- (a) The statement goto k, where k is an instruction number.
- (b) Success(), Failure().

(c) Algorithm A may contain type declaration and dimension statements. These are not used during execution of A and so need not be translated into Prob. The dimension information is used to allocate array space. It is assumed that successive elements in an array are assigned to consecutive words in memory. It is assumed that the instructions in A are numbered sequentially from 1 to l (if A has l instructions).

Every statement in A has a number. The goto instructions in (a) and (b) use this numbering scheme to effect a branch. It should be easy to see how to rewrite repeat-until, for and so on statements in terms of goto and if b then goto k statements. Also, note that the goto k statement can be replaced by the statement if true then goto k. So, this may also be eliminated.

7. Let $p(n)$ be a polynomial such that A takes no more than $p(n)$ time units on any input of length n. Because of the complexity assumption of 1, A cannot change or use more than $p(n)$ words of memory. We assume that A uses some subset of the words indexed $1, 2, 3, \dots, p(n)$. This assumption does not restrict the class of decision problems in NP. Therefore, this, let $f(1), f(2), \dots, f(k), 1 \leq k \leq p(n)$, be the distinct words

used by A while working on input I. We can construct another polynomial time nondeterministic algorithm A' that uses $2p(n)$ words indexed $1, 2, \dots, 2p(n)$ and solves the same decision problem as A does. A' simulates the behaviour of A. However, A' maps the addresses $f(1), f(2), \dots, f(k)$ onto the set $\{1, 2, \dots, k\}$. The mapping function used is determined dynamically and is stored as a table in words $p(n) + 1$ through $2p(n)$.

If the entry at word $p(n) + i$ is j, then A' uses word i to hold the same value that A stored in word j. The simulation of A proceeds as follows: Let k be the number of distinct words referenced by A upto this time. Let j be a word referenced by A in the current step. A' searches its table to find word $p(n) + i, 1 \leq i \leq k$, such that the contents of this word is j. If no such i exists, then A' sets $k := k + 1; i := k$; and word $p(n) + k$ gives the value j. A' makes use of the word i to do whatever A would have done with word j. Clearly, A' and A solve the same decision problem.

The complexity of A'; is $O(p^2(n))$ as it takes $A'p(n)$ time to search its table and simulate a step of A. Since $p^2(n)$ is also a polynomial in n, restricting our algorithms to use only consecutive words that does not alter the classes P and NP.



Q.7 Explain NP and Hard NP Complete with example. [R.T.U. 2016]

OR

Explain the terms P, NP, NP-Hard, NP-complete with suitable example. Also give relationship between them. [R.T.U. 2014]

Ans.(i) P : P is the set of decision problems with a yes-no answer that is polynomial bound.

A problem is said to be polynomial-bound if there exists a polynomial bound algorithm for it. It is also to be noted that not for all the problems the class Phas "acceptably efficient" algorithm. Also, if a problem does not belong to class P then it is intractable.

Note : An algorithm is said to be polynomial bounded if its worst-case complexity is bound by a polynomial function P of input size n. That means, for each input of size n the algorithm terminates after atmost $P(n)$ steps; For instance, $n^3 + 24n^2 + 65$

Decision Problems : The problems under this class have the single bit output which shows 0 or 1 i.e., the answer for the problem is either zero or one.

For instance, some decision problems are :

- Given two sets of strings S_1 and S_2 , does S_2 a substring of S_1 ?

• Given two sets of elements contain same number of

Any problem that involves (either minimum or maximum is known as optimization problem, an optimization algorithm optimization problem is as follows

• Given a weighted graph a minimum spanning tree of

• Given S, does there exist in the knapsack, and has total

We can say that a given 'S' only when A produces the set of string is referred to a language L. Also be viewed as a set L of strings. Thus, an algorithm A accepts output 'yes' on input 'S' if it produces output 'no'.

It is to be noted that the decision problems (or languages) the worst-case running time $S \in L$, in polynomial time $p(n)$ and produces output 'yes'. The definition does not say anything to this situation as a complete 'yes' for a given set of binary strings L.

We can also create a complement of L if given language L in polynomial time, showing such decision problem.

(ii) NP : The complexity class P but allows for the possibility of non-deterministic. But, in the case of NP problem, operation:

Select : This problem deterministic way and also takes the advantage of S say that A is non-deterministic calls to Select operation, to acceptance if there exists that this operation's work choices.

The complexity class (or languages L) that can be solved in the polynomial time. That is, for an input S, there exists a string A so that it produces output 'yes' where n is the input size.

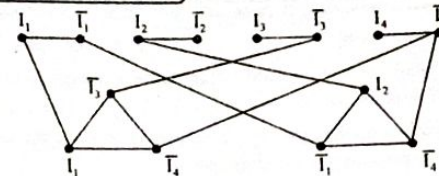


Fig.

The above graph will have a vertex cover of size $n + 2C$ if and only if the expression is satisfiable. Every cover must have at least $n + 2C$ vertices. For showing that our reduction is correct, we have to show the following :

For every satisfying truth assignment there exists a cover : For this select the n vertices that correspond to the true literals to be in the cover. As it is a satisfying truth assignment, atleast one of the three cross edges associated with each clause must already be covered. Now, select the other two vertices to complete the cover.

There exists a satisfying truth assignment for every vertex cover : For this, every vertex cover must contain n first level vertices and $2C$ second level vertices. Let the truth assignment be defined by the first level vertices. To get the cover at last, one cross-edge must be covered, so that the truth assignment satisfies.

It can be noticed that for a cover to have $n + 2C$ vertices, all the cross edges must be incident on a selected vertex. Let us consider that the n selected vertices from the first level corresponds to true literals. If there exists a satisfying truth assignment, then that means atleast one of the three cross edges from each triangle is incident on a true literal vertex. It is to be noted that by adding the other two vertices to the cover, we cover all the edges associated with the clause.

Vertex-cover problem is to find a vertex cover of minimum size. Using approximation algorithm we have to find a sub-optimal solution to the problem. As a result of this algorithm, we will get a vertex-cover with size no more than twice the size of an optimal vertex cover.

Algorithm Approx_vertex_cover

input to the algorithm is the graph G .

Step 1. Initialize the vertex-cover D to be null.

$C \leftarrow \phi$

Step 2. The set of edges in G is E .

Step 3. Repeat steps 4 to 6 till the set of edges E is empty.

Step 4. Choose an arbitrary edge (u, v) of E .

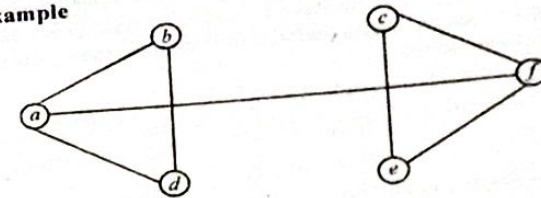
Step 5. Add the endpoints u, v to vertex cover C .

Step 6. Remove every edge incident on either u or v from the set of edges E .

Step 7. return C and Exit.

The running time of this algorithm is $O(V + E)$.

Example



$C = \phi$

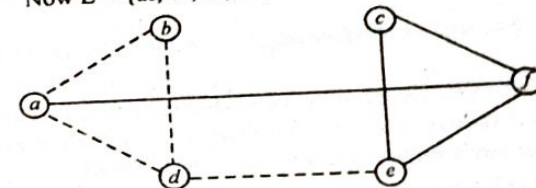
$E = \{ab, ad, bd, de, af, cf, ef, ce\}$

pick bd arbitrarily

$C = \{b, d\}$

Remove the edges associated with b or d , that is ab, bd, ad and de .

Now $E = \{af, cf, ef, ce\}$



Pick cf at random

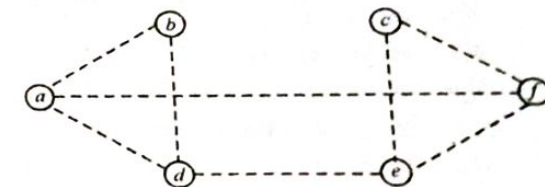
$C = \{b, d, c, f\}$

Remove edges associated with c or f , that is af, cf, ef and ce

Now $E = \phi$

So, stop

The cover is C



Q.11 Solve the Travelling Salesman Problem (TSP) for the following graph by using the branch and bound algorithm, the tour must be start from vertex 1 and generate only tour in which 2 is visited before 3.

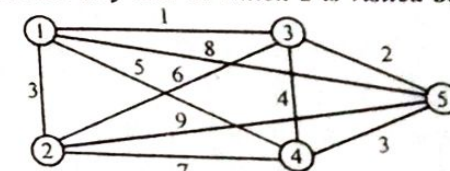


Fig.

PART-C

Q.13 Explain Hamiltonian path problem. [R.T.U. 2019]

OR

Prove that Hamilton cycle problem in NP complete.

[R.T.U. 2016]

OR

Show that the Hamilton Cycle problem is NP-complete.

[R.T.U. 2014]

Ans. Let us define a nondeterministic algorithm A that takes, as input, a graph G encoded as an adjacency list in binary notation, with the vertices numbered 1 to N. We define A to first iteratively call the choose method to determine a sequence S of N + 1 numbers from 1 to N. Then, we have A check that each number from 1 to N appears exactly once in S (for example, by sorting S), except for the first and last numbers in S, which should be the same. Then, we verify that the sequence S defines a cycle of vertices and edges in G. A binary encoding of the sequence S is clearly of size at most n, where n is the size of the input. Moreover, both of the checks made on the sequence S can be done in polynomial time in n.

Observe that if there is a cycle in G that visits each vertex of G exactly once, returning to its starting vertex, then there is a sequence S for which A will output "yes." Likewise, if A outputs "yes," then it has found a cycle in G that visits each vertex of G exactly once, returning to its starting point. That is, A non-deterministically accepts the language HAMILTONIAN-CYCLE. In other words, Hamiltonian-Cycle is in NP.

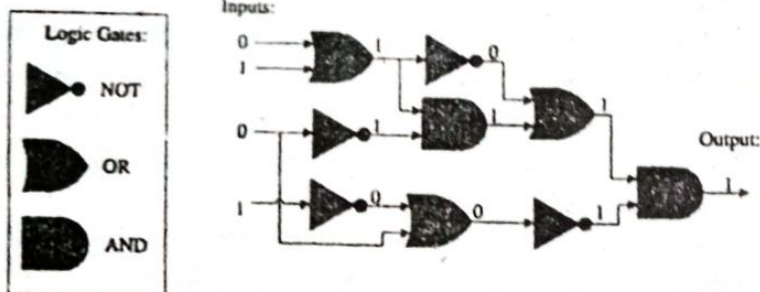


Fig. : An example of Boolean circuit.

Our next example is a problem related to circuit design testing. A Boolean circuit is a directed graph where each node, called a logic gate, corresponds to a simple boolean function, AND, OR, or NOT. The incoming edges for a logic gate correspond to inputs for its boolean function and the outgoing

edges correspond to outputs, which will all be the same value of course, for that gate. (see fig.) Vertices with no incoming edges are input nodes and a vertex with no outgoing edges is an output node.

Circuit-Sat is the problem that takes as input a boolean circuit with a single output node, and asks whether there is an assignment of values to the circuit's inputs so that its output value is "1." Such an assignment of values is called a satisfying assignment.

Q.14 Suggest an approximation algorithm for traveling sales person problems using minimum spanning tree algorithm. Assume that the cost function satisfies the triangle inequality. [R.T.U. 2014]

Ans. If for the set of vertices $a, b, c \in V$, it is true that $t(a, b) \leq t(a, c) + t(b, c)$ where t is the cost function, we say that t satisfies the triangle inequality.

First create a minimum spanning tree the weight of which is a lower bound on the cost of an optimal traveling salesman tour. Using this minimum spanning tree let us construct a tour the cost of which is at most 2 times the weight of the spanning tree.

Approximation – Traveling Sales Person Problem

Input: A complete graph $G(V, E)$

Output: A Hamiltonian cycle

1. Select a "root" vertex $r \in V[G]$.
2. Use MST-Prim (G, c, r) to compute a minimum spanning tree from r .
3. Assume L to be the sequence of vertices visited in preorder tree walk of T .
4. Return the Hamiltonian cycle H that visits the vertices in the order L .

The next set of figures show the working of the proposed algorithm.

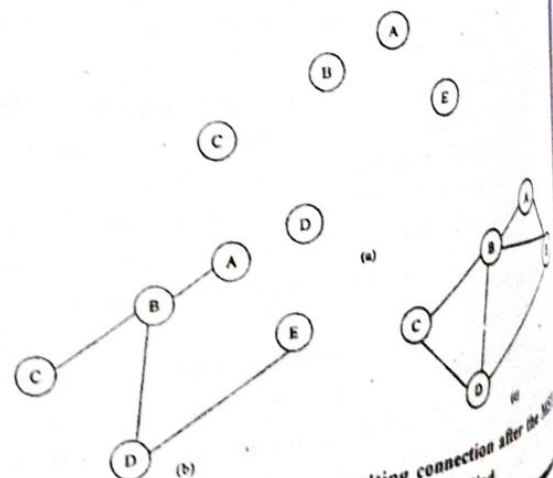


Fig. : A set of cities and the resulting connection after the Prim algorithm has been applied