Printed pages: 03

Sub Code: EIT-072 **Roll No** | | | | | | | | | | |

Paper Id: 113714

B.Tech (CSE/IT/CS&IT)
(SEM - 7) THEORY EXAMINATION 2018-19
THEORY OF AUTOMATA AND FORMAL LANGUAGES

Time: 3 Hours ***Total Marks: 100***

Note: 1. Attempt all Sections. If require any missing data; then choose suitably.
2. Any special paper specific instruction.

Solution

SECTION A

1. Attempt *all* questions in brief. 2 x10 = 20

a. Prove that union of two CFL's is CFL.

Solution:

Proof. Let L_1 and L_2 be two context-free languages generated by the following context free grammars respectively.

$$G_1 = (V_{n_1}, V_{t_1}, P_1, S_1)$$

and
$$G_2 = (V_{n_2}, V_{t_2}, P_2, S_2)$$

Union. Consider the language $L(G)$, generated by the following grammar

$$G = (V_n, V_t, P, S)$$

where
$$V_n = V_{n_1} \cup V_{n_2} \cup \{S\}$$
$$V_t = V_{t_1} \cup V_{t_2}$$

S is the start symbol, and productions P is defined as follows:

$$P = \{P_1 \cup P_2 \cup \{S \to S_1/S_2\}\}$$

Now let us choose a string $W \in (V_{t_1} \cup V_{t_2})^*$

It $S_1 \overset{*}{\Rightarrow} W$ or $S_2 \overset{*}{\Rightarrow} W$, and in our grammar

$S \to S_1/S_2$, hence S will lead to W.

Hence G is a context-free grammar, so that $L(G)$ is a context free languages that is

$$L(G) = L_1 \cup L_2 \text{ is a CFL.}$$

b. What is the role of finite automata for searching a keyword in documents?

Solution:

Keyword Searching is an important problem in computer science. The finite automats can be deigned to verify the particular word which is under search. For example the key words of C language can be verified by the designing the finite automat for particular key word.

c. State Myhill Nerode Theorm.

Solution:

The following statements are equivalent:

1. *L is a regular language.*
2. *L is the union of some of the equivalence classes of a right invariant relation of finite index.*
3. *L induces a relation =L of finite index, where =L is defined by: x =L y iff ∀x, xz and yz are either both in L or both not in L.*

d. Write difference between Mealy and Moore Machine.

Solution:

Moore Machine –

1. Output depends only upon present state.
2. If input changes, output does not change.
3. More number of states are required.
4. They react slower to inputs(One clock cycle later)
5. Synchronous output and state generation.
6. Output is placed on states.
7. Easy to design.

Mealy Machine –

1. Output depends on present state as well as present input.
2. If input changes, output also changes.
3. Less number of states are required.
4. They react faster to inputs.
5. Asynchronous output generation.
6. Output is placed on transitions.

e. Determine the type of the following language

i) S $\rightarrow$aAb ii) Ab$\rightarrow$bA

iii) aBA$\rightarrow$ ac iv) A$\rightarrow$a

Solution: i) Type-2 ii) Type -1, iii) Type-0 iv) Type-3

f. Compare PDA with FA.

Solution:

FINITE AUTOMATA	PUSHDOWN AUTOMATA
Finite automaton (FA) is a simple idealized machine used to recognize patterns within input taken from some character set	Pushdown automaton (PDA) is a type of automaton that employs a stack.
It doesn't has the capability to store long sequence of input alphabets	It has stack to store the input alphabets
Finite Automata can be constructed for Type-3 grammar	Pushdown Automata can be constructed for Type-2 grammar
Input alphabets are accepted by reaching "final states"	Input alphabets are accepted by reaching : Empty stack Final state
NFA can be converted into equivalent DFA	NPDA has more capability than DPDA

g. Write definition of recursive and recursively enumerable languages.

Solution:

A language L over alphabet Σ is called *recursively enumerable* (r.e.) if there is a Turing Machine T that accepts every word in L and either rejects or loops forever on every word that is not in L. A language L over alphabet Σ is called *recursive* (or decidable) if there is a Turing machine T that accepts every word in L and rejects every word that is not in L. In other words, a language is recursive if there is a Turing machine that accepts the language and never goes into an infinite loop for the strings that do not belong to the language. There are languages that are r.e. but not recursive; that is, they are undecidable. We will see some of these languages later. Here are some facts about r.e. and recursive languages.

h. Write brief note about church thesis.

Solution:

"The power of any computational process is captured within the class of Turing Machines."

It may be noted that Turing thesis is just a conjecture and not a theorem, hence, Turing thesis cannot be logically deduced from more elementary facts. However, the conjecture can be shown to be false, if a more powerful computational model is proposed that can recognize all the languages which are recognized by the TM model and also recognizes at least one more language that is not recognized by the TM.

The Turing Machines are designed to play atleast the following three different roles :

(*i*) *As accepting devices for languages,* similar to the role played by FAs and PDAs.

(*ii*) *As a computer of functions.* In this role, a TM represents a particular function. Initial input is treated as representing an *argument* of the function. And the (final) string on the tape when the TM enters the Halt state treated as representative of the value obtained by an application of the function to the argument represented by the initial string.

(*iii*) *As an enumerator of strings of a language*, that outputs the strings of a language, one at a time, in some systematic order that is as a list.

i. Write down the Arden's Theorem.

Solution:

Let P and Q be two regular expression over alphabet Σ. If P does not contain null string $\in$, then

$$R = Q + RP$$

has a unique solution that is $R = QP^*$

It can be understand as : $R = Q + RP$

Put the value of R in R.H.S.

$$R = Q + (Q + RP)P = Q + QP + RP^2$$

When we put the value of R again and again we got the following equation.

$$R = Q + QP + QP^2 + QP^3 \ldots$$

$$R = Q(1 + P + P^2 + P^3 \ldots$$

$$R = Q(\in + P + P^2 + P^3 + \ldots$$

$$R = QP^*$$

(By the definition of closure operation for regular expression.)

j. Sate the Halting Problem.

Solution:

In short halting problem is :

To determine for an arbitrary given Turing machine Tm and input w,

Whether Tm will eventually halt on input w.

SECTION B

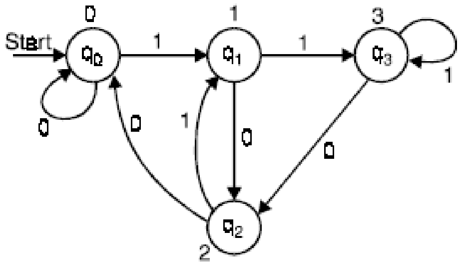
2. Attempt any *three* of the following: 10 x 3 = 30

a) i) Design a Moore Machine which calculates residue mod-4 when binary string is treated as integer.

Solution. This problem can be solved as we solved example 3.1. When we divide any number by 4 then reminder can be 0, 1, 2 and 3, so clearly moore machine will have four state.

Let moore machine is M_0

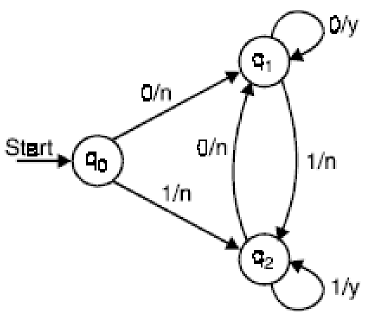
$$M_0 = (Q, \Sigma, \Delta, \delta, \lambda', q_0)$$
$$Q = \{q_0, q_1, q_2, q_3\}$$
$$\Sigma = \{0, 1\}$$
$$\Delta = \{0, 1, 2, 3\}$$
$$\lambda'(q_0) = 0, \lambda'(q_1) = 1, \lambda'(q_2) = 2 \text{ and } \lambda'(q_3) = 3$$



ii) Construct a Mealy Machine for regular expression $(0+1)^*$ (00+11)

Solution. Let us first analyse the problem, check the string set that is language of $(0 + 1)^*$ (00 + 11). It is set of strings either end with 00 or end with 11, like 00, 11, 1011, 010100, ...

Here we define a three state mealy machine that use it's state to remember the last symbol read, emits output *y* whenever the current input matches the previous one, and emits '*n*' otherwise. Here *y* is for the string belongs to regular expression and '*n*' otherwise.



b) Prove that following function is Turing Computable

$$f(m) = \begin{cases} m - 2, \text{ if } m > 2 \\ 1 \ \ , \text{ if } m \le 2 \end{cases}$$

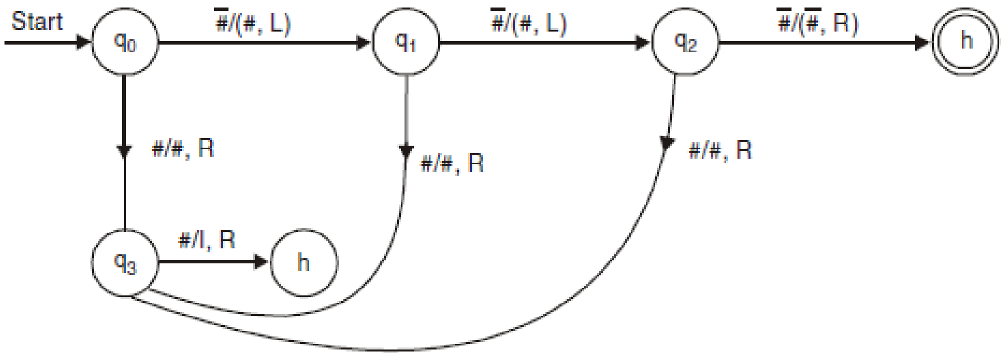
Solution. We know that if any function is turing computable then there exist a turing machine for it, so we have to design a turing machine of the $f(m)$ function. For the input strings #*I*# ($m = 1$), ##($m = 0$) and #*II*# ($m = 2$) output will be #*I*# ($m = 1$). For the input string #*IIII*# ($m = 4$), output will be #*II*# ($m = 4 - 2 = 2$, since $m > 2$).

Let Turing machine be $\quad Tm = (Q, \Sigma, \delta, \Gamma, q_0, h)$

where
$$Q = \{q_0, q_1, q_2, q_3, h\}$$
$$\Sigma = \{I\}$$
$$\Gamma = \{I, \#\}$$

q_0 is initial state and h is halting state.

Transition relation δ is defined as follows :



c) **Consider the following context free grammar G with start symbol S, which generates a set of arithmetic expressions:**

$$E \to I$$
$$E \to E + E$$
$$E \to E * E$$
$$E \to (E)$$
$$I \to a$$
$$I \to b$$
$$I \to Ia$$
$$I \to Ib$$
$$I \to I0$$
$$I \to I1$$

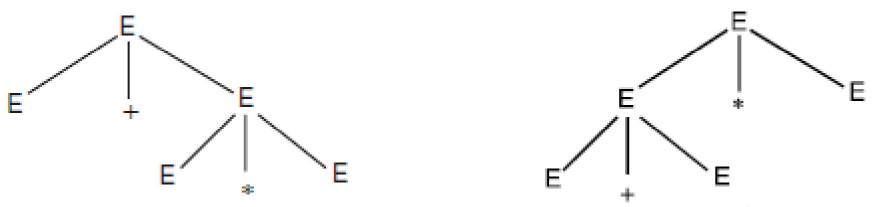
i. **Show that the given grammar is ambiguous.**

ii. **Write an equivalent unambiguous context free grammar G which generates the same language.**

Solution. For instance, consider the sentential form $E + E * E$. It has two derivation from E :

(1) $E \Rightarrow E + E \Rightarrow E + E * E$

(2) $E \Rightarrow E * E \Rightarrow E + E * E$



(a) (b)

The cause of the ambiguity in string a+a+a is that the associativity of + operator is not respected. Since, we assume that + operator is left associative, the string a+a+a is equivalent to (a+a)+a. Thus we need to force only the structure of fig ‘a’ to be legal in an ambiguous grammar.

Now, expressions are sums of one or more terms, terms are products of one or more factors, and factors are either parenthetical expressions or single identifiers. Sum of terms might suggest something such as $E \rightarrow T + T \mid T$. (keep in mind that an expression can consist of single term)

However, to obtain the sum of three terms with this approach, we would be forced to try $T \rightarrow T + T$ or something comparable, and again we would have ambiguity. What we say instead is that an expression is either a single term or the sum of a term and another expression. The only question is whether we want $E \rightarrow E + T$ or $E \rightarrow T + E$. Since + operator is left associative, we would probably choose $E \rightarrow E + T$ as more appropriate. Similarly, we choose the production $T \rightarrow T * F$ rather than $T \rightarrow F * T$.

The resulting unambiguous grammar is

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

In the similar fashion we can find the unambiguous grammar for algebraic expressions having other operators.

d) Write short notes on following:

- **i. Universal Turing Machines**
- **ii. The compliment of recursive language is recursive**
- **iii. Post Correspondence Problem & Modified Post Correspondence Problem.**
- **iv. CYK Algorithm**
- **v. Define two stack PDA**

Solution:

i)

We can consider turing machine in both ways : The turing machine is an "unprogrammable" piece of hardware, specialized at solving one particular problem, with instructions that are "*hard-wired at the factory*".

We shall now take the opposite point of view. We shall argue that turing machines are also software. That is, we shall show that there is a certain "generic" turing machine that can be programmed, about the same way that a general purpose computer can, to solve any problem that can be solved by turing machine. The "program" that makes this generic machine behave like a specific machine Tm will have to be a description of Tm. In other words, we shall be thinking of the formalism of turing machines as a programming language, in which we can write programs. Programs written in this language can then be **interpreted by universal machine** that is to say another program in same language.

To begin, we must present a general way of specifying turing machines, so that their descriptions can be used as input to other turing machines.

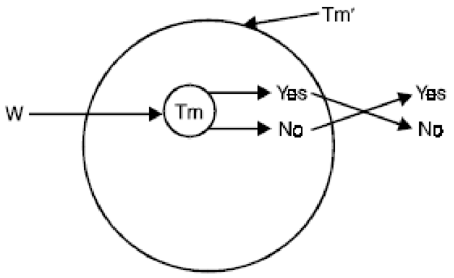
Universal turing machines '*U*' takes two arguments, a description of a machine Tm, "Tm", and a description of an input string *w*, "*w*". We want *U* to have the following property : *U* halts on input "Tm" "*w*" if and only if Tm halts on input *w*.

$$U(\text{“}m\text{” “}w\text{”}) = \text{“}m(w)\text{”}.$$

It is the functional notation of universal turing machine.

ii)

Proof. Let L be a recursive language and Tm be turing machine that halts on all inputs and accepts L. Let us construct a turing machine Tm′ from Tm so that if Tm enters a final state on input w, then Tm′ halts without accepting. If Tm halts without accepting, Tm′ enters a final state.



Since one of these two events occurs, Tm′ is an algorithm. So clearly T(Tm′) is the component of L and thus the complement of L is recursive language. Following figure shows construction of Tm′ from Tm.

iii)

Let Σ be an alphabet with at least two letters. An instance of the post corres-ponding problem (for short PCP) is given by two sequences $U = (u_1, u_2, ..., u_m)$ and $V = (v_1, v_2, ..., v_m)$ of strings $u_i, v_i \in \Sigma^*$. The problem is to find whether there is a (finite) sequence $(i_1, i_2, ..., i_p)$, with $i_j \in \{1, 2, ..., m\}$ for

$$i_j = 1, 2, ... p \text{ so that, } P \geq 1$$

$$u_{i_1}, u_{i_2} u_{i_3} ... u_{i_p} = v_{i_1} v_{i_2} ... v_{i_p}.$$

Equivalently, on instance of the PCP is a sequence of pairs

$$\begin{pmatrix} \mu_1 \\ \vdots \\ v_1 \end{pmatrix}, ..., \begin{pmatrix} \mu_m \\ \vdots \\ v_m \end{pmatrix}$$

The sequence $i_1, i_2 ... i_p$ is said to be solution to this instance of PCP.

In the modified PCP, there is the additional requirement on a solution that the first pair on the list X and list Y must be the first pair in the solution. More formally, an instance of MPCP is two lists

$$X = w_1, w_2, ..., w_k$$

and $$Y = x_1, x_2, x_3, ..., x_k,$$

and a solution is a list of 0 or more integers $i_1, i_2, i_3, ... i_p$ such that

$$w_1, w_{i_1}, w_{i_2}, ..., w_{i_m} = x_1, x_{i_1}, x_{i_2}, ..., x_{i_p}$$

iv)

Algorithm is named CYK because it was invented by John Cocke and subsequently also published by Tadao Kasami (1965) and Daniel H. Younger (1967).

First let us change the grammar from CFG to CNF form and let us make a list of all the non terminals in the grammar $S, X_1, X_2, X_3, ...$ and let the string we are examining for membership in the language be denoted by

$$w = a_1, a_2, a_3, a_4, ..., a_n$$

In general, it may be that the letters are not all different, but what we are interested in here in the position of every possible substring of w. We shall be answering the questions of which substring of w are producible from which non-terminal S. For example, if we already know that the substring $a_3 \ldots a_7$ can be derived from the non-terminal X_8, the substring $a_8, \ldots a_{11}$ can be derived from the non-terminal X_2, and we have the CNF production

$$X_4 \to X_8X_2,$$

then we can conclude that the total substring $a_3 \ldots a_{11}$ can be derived from the non-terminal X_4, that is:

$$X_8 \overset{*}{\Rightarrow} a_3 \ldots a_7 \text{ and } X_2 \overset{*}{\Rightarrow} x_8 \ldots x_{11} \text{ and } X_4 \overset{*}{\Rightarrow} X_8X_2$$

We can conclude that $X_4 \overset{*}{\Rightarrow} x_3 \ldots x_{11}$.

v)

We define the machine as a six tuple where

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

where Q : set of states

Σ : alphabet of input tape

Γ : alphabet of the stacks (assume same alphabet for both the stacks)

q_0 : the starting state ($q_0 \in Q$)

F : set of accepting states ($F \subseteq Q$) and

δ : the transition relation, a finite subset of $(Q \times \Sigma \times \Gamma^* \times \Gamma^*) \times (Q \times \Gamma^* \times \Gamma^*)$ or we can say that δ is transition function which maps

$$(Q \times \Sigma \times \Gamma^* \times \Gamma^*) \longrightarrow (Q \times \Gamma^* \times \Gamma^*)$$

e) Construct a Finite Machine :

i. For L = { $(01)^i 1^{2j}$ / $i>=1, j>=1$}

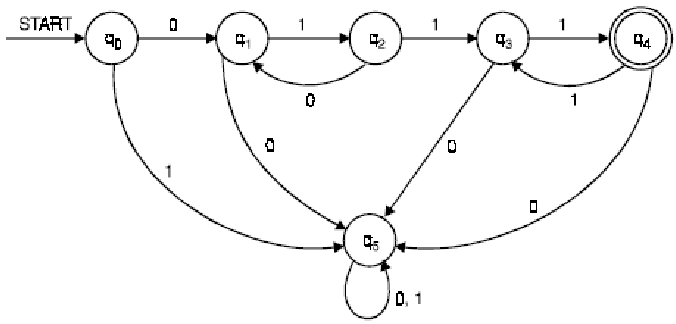
Solution. By analysing language L, it is clear that FA will accepts strings start with any number of 01 (not empty) and end with even number of 1's.

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

q : initial state.

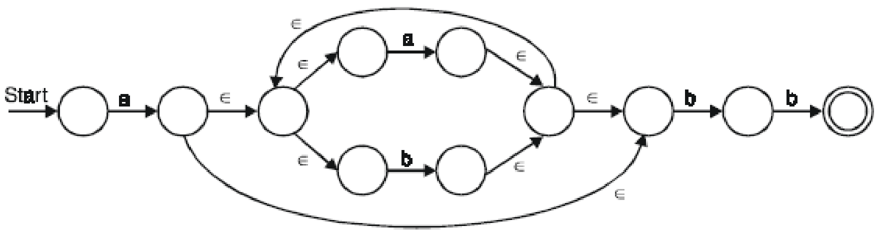
δ : is transition function.

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



ii. For the RE a.(a+b)*b.b

Solution:



SECTION C

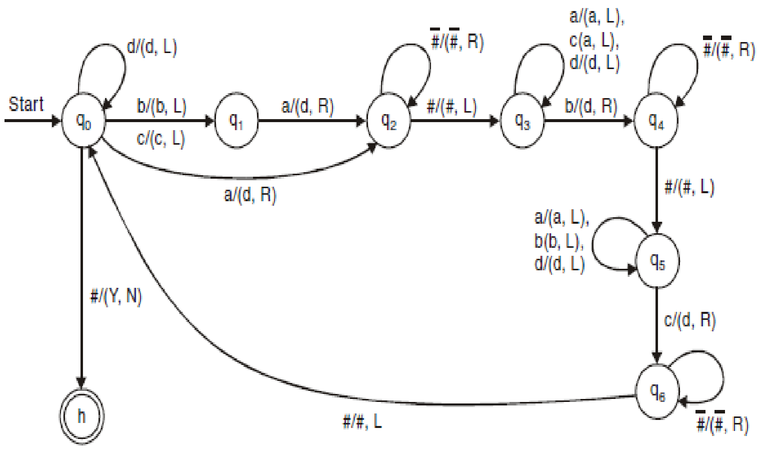
3. Attempt any *one* part of the following: **10 x 1 = 10**

a) Design a Turing Machine for language

L= {every string have equal number of a, b and c}.

Solution. Here turing machine start searching from right and first search '*a*' then move to extreme right and this '*a*' is repalced by *d*. Now machine search for '*b*' and if find replace it by '*d*' and again move on extreme right and search corresponding '*c*' and if find replace it by *d*, this completes one cycle. This process reepated and if all symbols are replaced by *d*'s then string is accepted and rejected in any other case.

Turing machine is :

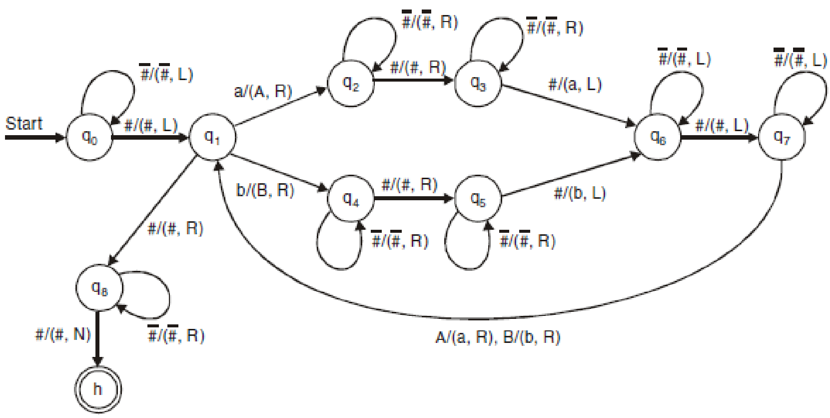


b) Design a Turing Machine which works as copying machine (c), for w is from (a, b) $^{+}$.

Solution. We want to design a turing machine which gives output for input $w = \#ab\#$ as $\#ab\#ab\#$ that is

$$\#w\# \vdash \text{Tm } \#w\#w\#$$

The transition diagram of turing machine will be as follows :



4. **Attempt any two part of the following:** **5x2=10**

a) **Prove that following instance of PCP does not have any solution:**

	List X	*List Y*
i	w_i	x_i
1	10	101
2	011	11
3	101	011

Solution. Let us assume that this instance of PCP has solution $i_1, i_2, ..., i_p$. Clearly $i_1 = 1$ since no string beginning with $w_2 = 011$ can equal a string beginning with $x_2 = 11$; no string beginning with $w_3 = 101$ can equal a string beginning with $x_3 = 011$.

We write the string from list X the corresponding string from Y. So for we have

10
101

The next selection from X must begin with a 1. Thus $i_2 = 1$ or $i_2 = 3$. But $i_2 = 1$ will not do, since no string beginning with $w_1\ w_1 = 1010$ can equal a string beginning with $x_1\ x_1 = 101101$. with $i_2 = 3$ we have

10101
101011

Since the string from list Y again exceeds the string from list X by the single symbol 1, a similar argument shows that $i_3 = i_4 = ... = 3$. Thus there is only one sequence of choices that generates compatible strings, and for this sequence string Y is always one character longer. Thus this instance of PCP has no solution.

b) Convert CFG which is given below into CNF form.

S → bA/aB
A → bAA/aS/a
B → aBB/bS/b.

Solution. Let us replace b by C_b and a by C_a, then CFG becomes

$$S \rightarrow C_bA/C_aB$$
$$A \rightarrow C_bAA/C_aS/a$$
$$B \rightarrow C_aBB/C_bS/b$$
$$C_b \rightarrow b$$
$$C_a \rightarrow a$$

Now let us replace C_bA by D and C_aB by E then grammar becomes as follows :

$$S \rightarrow C_bA/C_aB$$
$$A \rightarrow DA/C_aS/a$$
$$B \rightarrow EB/C_bS/b$$
$$C_b \rightarrow b$$
$$C_a \rightarrow a$$
$$D \rightarrow C_bA$$
$$E \rightarrow C_aB$$

Now every production of the grammar is in the CNF form.

c) Convert the following NFA to DFA.

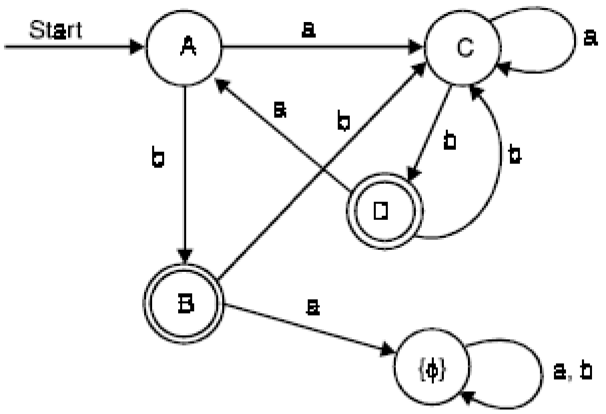
δ/Σ	a	b
$\rightarrow q_0$	$\{q_0, q_1\}$	$\{q_2\}$
q_1	$\{q_0\}$	$\{q_1\}$
$*q_2$	ϕ	$\{q_0, q_1\}$

Solution:

Transform table for DFA

δ/Σ	a	b
$\rightarrow A$	C	B
$*B$	$\{\phi\}$	C
C	C	D
$*D$	A	C

Transition diagram for DFA



5. Attempt any *one* part of the following: 10 x 1 = 10

a) Convert the following grammar into CNF

S→ XA |BB, B → b/SB, X→ b/a

Solution.

1. Rewrite G in Chomsky Normal Form (CNF)

   It is already in CNF.

2. Re-label the variables

   S with A_1

   X with A_2

A with A_3

B with A_4

$$A \to A_2A_3 \mid A_4A_4$$
$$A_4 \to b \mid A_1A_4$$
$$A_2 \to b$$
$$A_3 \to a$$

3. Identify all productions in which left side has a lower numbered variable than first variable in the right side.

4. $$A_4 \to A_1A_4$$ ...identified

5. $$A_4 \to A_1A_4/b$$

To eliminate A_1 we will use the substitution rule

Substituting for $A_1 \to A_2A_3 \mid A_4A_4$

$$A_4 \to A_2A_3A_4 \mid A_4A_4A_4 \mid b$$

The above two productions still do not conform to any of the types in step 3.

Substituing for $A_2 \to b$

$$A_4 \to bA_3A_4 \mid A_4A_4A_4 \mid \mathrm{b}$$

Now we have to remove left recursive production

$$A_4 \to A_4A_4A_4$$
$$A_4 \to bA_3A_4 \mid b \mid bA_3A_4Z \mid bZ$$
$$Z \to A_4A_4 \mid A_4A_4Z$$

6. At this stage our grammar now looks like

$$A_1 \to A_2A_3 \mid A_4A_4$$
$$A_4 \to bA_3A_4 \mid b \mid bA_3A_4Z \mid bZ$$
$$Z \to A_4A_4 \mid A_4A_4Z$$
$$A_2 \to b$$
$$A_3 \to a$$

All rules now conform to one of the types in step 3.

But the grammer is still not in Greibach Normal Form!

7. All productions for A_2, A_3 and A_4 are in GNF

for $A_1 \to A_2A_3 \mid A_4A_4$

Substitute for A_2 and A_4 to convert it to GNF

$$A_1 \to bA_3 \mid bA_3A_4A_4/bA_4/bA_3A_4ZA_4/bZA_4$$

for $Z \to A_4A_4 \mid A_4A_4Z$

Substitute for A_4 to convert it to GNF

$$Z \to bA_3A_4A_4 \mid bA_4 \mid bA_3A_4ZA_4 \mid bZA_4 \mid bA_3A_4A_4Z \mid bA_4Z \mid bA_3A_4ZA_4Z \mid bZA_4Z$$

8. Finally the Grammar in GNF is

$$A_1 \to bA_3 \mid bA_3A_4A_4 \mid bA_4 \mid bA_3A_4ZA_4 \mid bZA_4$$
$$A_4 \to bA_3A_4 \mid b \mid bA_3A_4Z \mid bZ$$
$$Z \to bA_3A_4A_4 \mid bA_4 \mid bA_3A_4ZA_4 \mid bZA_4 \mid bA_3A_4A_4Z \mid bA_4Z \mid bA_3A_4ZA_4Z \mid bZA_4Z$$
$$A_2 \to b$$
$$A_3 \to a$$

b) Convert the following PDA into equivalent CFG.

1. **$(q_0, a_1, z_0) \to (q_0, z_1 z_0)$**
2. **$(q_0, a_1, z_1) \to (q_0, z_1 z_1)$**
3. **$(q_0, b_1, z_1) \to (q_1, \in)$**
4. **$(q_1, b, z_1) \to (q_1, \in)$**
5. **$(q_1, b, z_0) \to (q_1, z_2 z_0)$**
6. **$(q_1, b, z_2) \to (q_1, z_2 z_2)$**
7. **$(q_1, c, z_2) \to (q_2, \in)$**
8. **$(q_2, c, z_2) \to (q_2, \in)$**
9. **$(q_2, \in, z_0) \to (q_2, \in)$**

Solution. The PDA have 3 states: q_0, q_1 and q_2. So, we will add following 3 productions to CFG as per rule-1.

$$S \rightarrow [q_0\ z_0\ q_0\ ] | [q_0\ z_0\ q_1\ ] | [q_0\ z_0\ z_2\ ]$$

The transition relation 3, 4, 7, 8 and 9 are applicable as per 2nd rule. Thus, we will add following 5 productions to CFG.

$(q_0\ z_1\ q_1) \rightarrow b$ (From transition function 3)

$(q_1\ z_1\ q_1) \rightarrow b$ (From transition function 4)

$(q_1\ z_2\ q_1) \rightarrow c$ (From transition function 7)

$(q_2\ z_2\ q_2) \rightarrow c$ (From transition function 8)

$(q_2\ z_0\ q_2) \rightarrow \in$ (From transition function 9)

For the remaining transition functions, rule-3 will be applicable.

For $\delta(q_0, a, z_0) \rightarrow (q_0, z_1 z_2)$ add following production in G:

$$(q_0\ z_0\ q_0) \rightarrow a(q_0\ z_1\ q_0)\ (q_0\ z_0\ q_0)$$
$$(q_0\ z_0\ q_0) \rightarrow a(q_0\ z_1\ q_1)\ (q_1\ z_0\ q_0)$$
$$(q_0\ z_0\ q_0) \rightarrow a(q_0\ z_1\ q_2)\ (q_2\ z_0\ q_0)$$
$$(q_0\ z_0\ q_1) \rightarrow a(q_0\ z_1\ q_0)\ (q_0\ z_0\ q_1)$$
$$(q_0\ z_0\ q_1) \rightarrow a(q_0\ z_1\ q_1)\ (q_1\ z_0\ q_1)$$
$$(q_0\ z_0\ q_1) \rightarrow a(q_0\ z_1\ q_2)\ (q_2\ z_0\ q_1)$$
$$(q_0\ z_0\ q_2) \rightarrow a(q_0\ z_1\ q_0)\ (q_0\ z_0\ q_2)$$
$$(q_0\ z_0\ q_2) \rightarrow a(q_0\ z_1\ q_1)\ (q_1\ z_0\ q_2)$$
$$(q_0\ z_0\ q_2) \rightarrow a(q_0\ z_1\ q_2)\ (q_2\ z_0\ q_2)$$

For $\delta(q_0, a, z_1) \rightarrow (q_0, z_1 z_1)$ add following production to G:

$$(q_0 z_1 q_0) \rightarrow a(q_0 z_1 q_0)\ (q_0 z_1 q_0)$$
$$(q_0 z_1 q_0) \rightarrow a(q_0 z_1 q_1)\ (q_1 z_1 q_0)$$
$$(q_0 z_1 q_0) \rightarrow a(q_0 z_1 q_2)\ (q_2 z_1 q_0)$$
$$(q_0 z_1 q_1) \rightarrow a(q_0 z_1 q_0)\ (q_0 z_1 q_1)$$
$$(q_0 z_1 q_1) \rightarrow a(q_0 z_1 q_1)\ (q_1 z_1 q_1)$$
$$(q_0 z_1 q_1) \rightarrow a(q_0 z_1 q_2)\ (q_2 z_1 q_1)$$
$$(q_0 z_1 q_2) \rightarrow a(q_0 z_1 q_0)\ (q_0 z_1 q_2)$$
$$(q_0 z_1 q_2) \rightarrow a(q_0 z_1 q_1)\ (q_1 z_1 q_2)$$
$$(q_0 z_1 q_2) \rightarrow a(q_0 z_1 q_2)\ (q_2 z_1 q_2)$$

For $\delta(q_1, b, z_0) \rightarrow (q_0, z_2 z_0)$ add following production to G:

$$(q_1 z_0 q_0) \rightarrow b(q_1 z_2 q_0)\ (q_0 z_0 q_0)$$
$$(q_1 z_0 q_0) \rightarrow b(q_1 z_2 q_1)\ (q_1 z_0 q_0)$$
$$(q_1 z_0 q_0) \rightarrow b(q_1 z_2 q_2)\ (q_2 z_0 q_0)$$
$$(q_1 z_0 q_1) \rightarrow b(q_1 z_2 q_0)\ (q_0 z_0 q_1)$$
$$(q_1 z_0 q_1) \rightarrow b(q_1 z_2 q_1)\ (q_1 z_0 q_1)$$
$$(q_1 z_0 q_1) \rightarrow b(q_1 z_2 q_2)\ (q_2 z_0 q_1)$$
$$(q_1 z_0 q_2) \rightarrow b(q_1 z_2 q_0)\ (q_0 z_0 q_2)$$
$$(q_1 z_0 q_2) \rightarrow b(q_1 z_2 q_1)\ (q_1 z_0 q_2)$$
$$(q_1 z_0 q_2) \rightarrow b(q_1 z_2 q_2)\ (q_2 z_0 q_2)$$

For $\delta(q_1, b, z_2) \rightarrow (q_1, z_2 z_2)$ add following production to G:

$$(q_1 z_2 q_0) \rightarrow b(q_1 z_2 q_0)\ (q_0 z_2 q_0)$$
$$(q_1 z_2 q_0) \rightarrow b(q_1 z_2 q_1)\ (q_1 z_2 q_0)$$
$$(q_1 z_2 q_0) \rightarrow b(q_1 z_2 q_2)\ (q_2 z_2 q_0)$$
$$(q_1 z_2 q_1) \rightarrow b(q_1 z_2 q_0)\ (q_0 z_2 q_1)$$
$$(q_1 z_2 q_1) \rightarrow b(q_1 z_2 q_1)\ (q_1 z_2 q_1)$$
$$(q_1 z_2 q_1) \rightarrow b(q_1 z_2 q_2)\ (q_2 z_2 q_1)$$
$$(q_1 z_2 q_2) \rightarrow b(q_1 z_2 q_0)\ (q_0 z_2 q_2)$$
$$(q_1 z_2 q_2) \rightarrow b(q_1 z_2 q_1)\ (q_1 z_2 q_2)$$
$$(q_1 z_2 q_2) \rightarrow b(q_1 z_2 q_2)\ (q_2 z_2 q_2)$$

6. Attempt any *one* part of the following: 10 x 1 = 10

a) Define push down automata. Design a PDA for the following language:
$L = \{a^n b^{2n} \mid n>0\}$.

Solution. Language $L = \{a^n b^{2n} : n > 0\}$ is the set of strings, in which every string starts with a, ends with b, no 'a' comes after b, no b comes before a and number of b's are double than number of a's. For example, strings like *aabbbb, abb, aaabbbbbb* will be accepted by PDA and strings like *a, b, ab, ba, aab, aaabbb* will be rejected.

Let PDA be $P = (Q, \Sigma, \Gamma, \delta, S, F)$

where $Q = (p, q, f, r)$

$S = \{P\}$

$F = \{f\}$

and transition relations are defined as follows :

(1) $((p, a, \in), (p, a))$

(2) $((p, a, a), (p, a))$

(3) $((p, b, a), (q, b))$

(4) $((q, b, ba), (r, \in))$

(5) $((r, b, a), (q, ba))$ $\rightarrow$ Here these two transition 4 and 5 reprsent the looping.

(6) $((r, \in, \in), (f, \in))$

b) Find a context free grammar for the following language:

$L = \{a^i b^j c^k \mid j = i \text{ or } k=j\}$

Solution. Let us assume $L = L_1 \cup L_2$

where $L_1 = \{0^i\ 1^j\ 2^k / i = j\}$

and $L_2 = \{0^i\ 1^j\ 2^k / j = k\}$

Let us consider L_1 first, let CFG for L_1 be G_1

$$G_1 = (V_n, V_t, P_1, S_1)$$
$$V_n = \{S_1, A, B\}$$
$$V_t = \{0, 1, 2\}$$

P_1 is defined as follows

$$S_1 \to AB$$
$$A \to 0A1/\in$$
$$B \to 2B/\in$$

Now let us assume that CFG for language L_2 is G_2

$$G_2 = (V_n, V_t, P_2, S_2)$$
$$V_n = (S_2, C, D)$$
$$V_t = (0, 1, 2)$$

Productions are defined as follows :

$$S_2 \to CD$$
$$C \to 0C/\in$$
$$D \to 1D2/\in$$

By the help G_1 and G_2 we can define CFG for the language L, Let it is G.

$$G = (V_n, V_t, P, S)$$
$$V_n = \{S, S_1, S_2, A, B, C, D\}$$
$$V_t = \{0, 1, 2\}$$

Productionsa are defined as follows

$$S \to S_1/S_2$$
$$S_1 \to AB$$
$$A \to 0A1/\in$$
$$B \to 2B/\in$$
$$S_2 \to CD$$
$$C \to 0C/\in$$
$$D \to 1D2/\in$$

which is required CFG.

7. Attempt any two part of the following: 5x2=10

a) Using pumping lemma for context free languages, prove that the following language is not context free:-

$L = \{a^p \mid p \text{ is a prime number}\}$

Solution. By contradiction. Let us assume that L is regular, let n be constant guaranteed by pumping lemma. Let $w = 0^p$ where p is some prime number so smaller then n. Clearly w is in L.

By pumping lemma we can write $w = xyz$ such that $| xy | < n$, $y \neq \in$ and for all $k \in N$ $xy^k z \in L$. Then

$$\begin{aligned} xy^{p+1}z &= 0^{p + p|y|} \\ &= 0^{P(1 + |y|)} \\ &\notin L \text{ because } |y| > 0 \end{aligned}$$

a contradiction. Therefore, L is not regular.

b) Let G be CFG

S → bB/aA,

A → b/bS/aAA

B → a/aS/bBB.

For the string bbaababa find

(i) left most derivation

(ii) rightmost derivation and

(iii) parse tree.

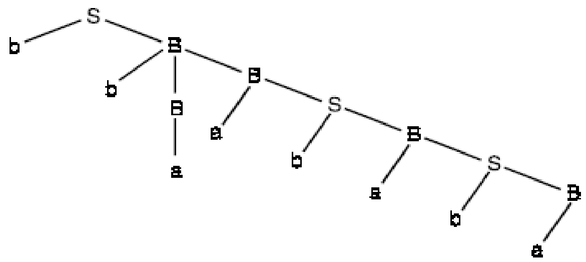
Solution. (*i*) Left most derivation for string $w = bbaababa$ is

$S \Rightarrow b\underline{B} \Rightarrow bb\underline{BB} \Rightarrow bba\underline{B} \Rightarrow bbaa\underline{S} \Rightarrow b^2a^2b\underline{B} \Rightarrow b^2a^2ba\underline{S} \Rightarrow b^2a^2bab\underline{B} \Rightarrow \underline{b}^2a^2baba$

(*ii*) The right most derivation is

$S \Rightarrow b\underline{B} \Rightarrow bbB\underline{B} \Rightarrow bbBa\underline{S} \Rightarrow bbBab\underline{B} \Rightarrow b^2Baba\underline{S} \Rightarrow b^2Babab\underline{B} \Rightarrow b^2\underline{B}ababa \Rightarrow b^2a^2baba$

The derivation tree is following in Fig. 6.9.



c) Show that L = { $a^n b^n c^m$ } U { $a^n b^m c^m$ } with m & n are non-negative is an inherently ambiguous context free language.

Solution. Let us say $L = L_1 \cup L_2$

where $L_1 = \{a^n b^n c^m\}$

and $L_2 = \{a^n b^m c^m\}$

Let us write CFG for L_1, let it is G_1, as follows :

$$S_1 \rightarrow S_1 c / A$$
$$A \rightarrow aAb/\in$$

Similarly we can write CFG for L_2, Let it is G_2, as follows :

$$S_2 \rightarrow aS_2/B$$
$$B \rightarrow bBc/\in$$

Now with the help of G_1 and G_2 we can write CFG for the language L, as follows

$$S \rightarrow S_1/S_2$$

where S is starting non-terminal for CFG of language L.

The grammar is ambiguous since the string $a^n\, b^n\, c^m$ has two distinct derivation, one starting with $S \Rightarrow S_1$, and another with $S \Rightarrow S_2$. It does of course not follows that L is in herently ambiguous as there might exist some other non-ambiguous grammar for it. But in some way L_1 and L_2 have some conflicting requirements, the first putting a restriction on the number of a's and b's, while the second does the same for b's and c's. A few tries will quickly convince us of the impossibility of combining these requirements in a single set of rules that cover the case $n = m$ uniquely.