TABLE OF CONTENTS

Unit III

Unit IV

Unit V

Unit VI

Unit III

3 Context Free Grammar (CFG) and Context Free Language (CFL)

3.1 : Formal Definition of Context Free Grammar

Q.1 Define the term Context Free Grammar.

Ans. : The context free grammar can be formally defined as a set denoted by $G = (V, T, P, S)$ where V and T are set of non terminals and terminals respectively. P is set of production rules, where each production rule is in the form of

Non terminal $\rightarrow$ Non terminals

or Non terminal $\rightarrow$ Terminals

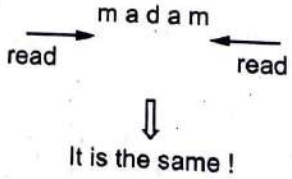
S is a start symbol.

For example

$$P = \{ S \rightarrow S + S$$
$$S \rightarrow S * S$$
$$S \rightarrow (S)$$
$$S \rightarrow 4 \}$$

Q.2 Construct CFG for the language L which has all the strings which are all palindrome over $\Sigma = \{a, b\}$.

Ans.: As we know the strings are palindrome if they possess same alphabets from forward as well as from backward.



For example, the string "madam" is a palindrome because

Since the language L is over $\Sigma = \{a, b\}$. We want the production rules to be build *a*'s and b's. As ε can be the palindrome, *a* can be palindrome even b can be palindrome. So we can write the production rules as

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

P can be $S \rightarrow a\ S\ a$

$S \rightarrow b\ S\ b$

$S \rightarrow a$

$S \rightarrow b$

$S \rightarrow \varepsilon$

The string abaaba can be derived as

S	
a S *a*	*a* b a a b *a*
a b S b *a*	*a* b *a* a b *a*
a b *a* S *a* b	*a a* b *a a* b *a*
a b *a* ε *a* b	*a a* b *a a* b *a*
a b *a a* b *a*	*a* b *a a* b *a*

which is a palindrome.

Q.3 Build a CFG for the language L = $\{0^i\ 1^j\ 2^k \mid j > i+k\}$.

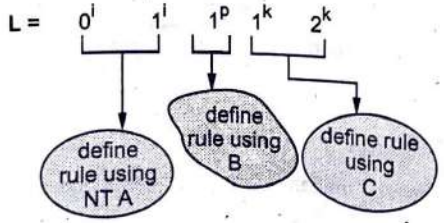
Ans. : As the language L = $0^i\ 1^j 2^k$ such that j > i + k.

Hence we can rewrite L as

$L = 0^i\ 1^i\ 1^k\ 1^p\ 2^k$

This will be greater than i+k

We can again rewrite L for simplification as -



$A \rightarrow 0\ A\ 1 \mid \varepsilon$

$B \rightarrow 1\ B \mid 1$

$C \rightarrow 1\ C\ 2 \mid \varepsilon$

Hence the CFG can be

$S \rightarrow A\ B\ C$

$A \rightarrow 0\ A\ 1 \mid \varepsilon$

$B \rightarrow 1\ B \mid 1$

$C \rightarrow 1\ C\ 2 \mid \varepsilon$

Q.4 If the grammar G is given by production S → aSa|bSb|aa|bb|ε show that

- **Any string in L(G) is of length 2n, n > = 0**
- **The number of strings of length 2n is 2^n**

☞[SPPU : Dec.-13, Marks 2]

Ans. : The CFG denotes the language L of even length in which the two middle symbols equal. Hence it will generate the strings such as

L(G) = { ε, aa, bb, aaaa, aaab, baab, baaa, abba, abbb, bbbb, bbba, ...}

- If n = 0, the string is ε.
- If n = 1, the string of length 2 (i.e. 2 × 1) are aa and bb (i.e. $2^n = 2$).

Q.5 Write a CFG for generating identifiers in higher-level languages such as 'C'. Identifiers can be defined by the regular expression (letter). (letter | digit)*. **☞[SPPU : May-13, Marks 4]**

Ans. : $S \rightarrow LA$

$A \rightarrow LA \mid DA \mid \in$

$L \rightarrow a|b|c \ldots z$

$D \rightarrow 0|1|2 \ldots 9$

Q.6 In each case find context free grammar generating given language

a) The set of odd length strings in {a, b}* with middle symbol a.

b) The set of even length strings {a, b}* with the two middle symbols equal

☞[SPPU : Dec.-07, May-08, Marks 8]

Ans. : a) Required CFG is -

$S \rightarrow aSa$

$S \rightarrow aSb$

$S \rightarrow bSb$

$S \rightarrow bSb$

$S \rightarrow a$

b) This language can be denoted by regular expression.

regular expression = $(a + b)^* (aa + bb) (a + b)^*$

Hence CFG can be given as

$S \rightarrow aSa|aSb|bSb|bSa$

$S \rightarrow aa$

$S \rightarrow bb$

Q.7 Give the context free grammar for the following languages

a) $L = \{a^n b^{2n} | n>1\}$ **b)** $L = \{a^m b^n | n>m\}$

☞[SPPU : Dec.-08, Marks 12]

Ans. : a) Let, $L = \{a^n b^{2n} | n>1\}$

The production rules for the CFG of above language will be -

$S \rightarrow aAbb$

$A \rightarrow aAbb|abb$

b) Let, $L = \{a^m b^n \mid n > m\}$

The L can be written as $L = a^m b^m b^K$ where $K \geq 1$.

Hence required CFG will be

$S \rightarrow aSbB|\varepsilon$

$B \rightarrow bB|b$

Q.8 Give the context free grammars for the following languages.

a) $(011 + 1)^* (01)^*$ **b)** $0^i 1^{i+k} 0^k$ **where i, k > = 0.**

☞ [SPPU : May-09, Marks 12, May-14, Marks 8]

Ans. : a) The CFG can be denoted by

$G = (\{S, A, B, C, D\}, \{0, 1\}, P, S)$

where $P = \{$

$S \rightarrow AB$

$A \rightarrow CA \mid \varepsilon$

$C \rightarrow 011 \mid 1$

$B \rightarrow DB \mid \varepsilon$

$D \rightarrow 01$

$\}$

Let us derive a valid string $01110101 \in (011 + 1)^* (01)^*$

S

AB

CAB

CCAB

CCADB

CCADDB

011CADDB

0111ADDB

0111DDB

011101DB

01110101B

0111101

b) Let us rewrite the language L as

$L = 0^i 1^{i+k} 0^k$

$L = 0^i 1^i 1^k 0^k$

Hence required CFG $G = (\{S, A, B\}, \{0, 1\}, P, S)$

Hence

$P = \{S \rightarrow AB$

$= A \rightarrow 0A1 \mid \varepsilon$

$= B \rightarrow 1B0 \mid \varepsilon$

$\}$

Q.9 Give context free grammars for the following languages.
i) L = {S → aAb, A → aA|bA|ε, x|x ε {a, b}* with strings of starting with 'a' and ending 'b'}
ii) L = {x|x ε {a, b}* with strings of even length palindrome}.

☞[SPPU : Dec.-14, In Sem, Marks 4]

Ans. : i) S → aAb, A → aA|bA|ε.

ii)

Step 1 : As we know the strings are palindrome if they possess same alphabets from forward as well as from backward.

Step 2 : Let G = ({S}, {a, b}, P, S)

Step 3 : The string abaaba can be derived as

P can be
S → a S a
S → b S b
S → a
S → b
S → ε

S
a S a ⇒ b a a b
a b S b a ⇒ b a a b
a b a S a b ⇒ b a b
a b a ε a b a ⇒ b a b
a b a a b a ⇒ b a b

Q.10 Give context free grammars for the following languages :
i) L = {x | x ∈ (,)}* with strings having well-formed parentheses (WFP) ii) L = {$a^m b^n c^{m+n}$ | m, n >=0}

☞[SPPU : Oct.-16 In sem, Marks 6]

Ans. : i)The production rule are

P = {
S → ( ) S | ( ) S | (S) | ( )
}

ii) We will first rewrite the given L as

L = $a^m b^n c^n c^m$

Hence P = {
S → aSc | abcc
S → abAcc
A → bAc | ε
}

Consider the derivation for string
aabccc
S
aSc
aabccc

Q.11 Write context free grammar for the following language 0 (0 + 1)* 01 (0 + 1)* 1. **☞ [SPPU : Aug.-17 In Sem, Marks 4]**

Ans. :

P → {
S → 0A01A1
A → 0A | 1A | ε

Q.12 Write CFGs for given CFLs :
i) Languages containing the strings with equal number of a's and b's
ii) Languages containing the strings containing a's and b's with at least 2 a's. **☞ [SPPU : Dec 17 End Sem, Marks 8]**

Ans. : i) The production rules are

S → aB | bA
A → a | aS | bAA
B → b | bS | aBB

ii)
S → BAB
A → aa
B → aB | bB | ε

Q.13 Write the CFG for following language.
L = {$a^{m+n} b^m c^n$ | n, m >=0} **☞ [SPPU : Oct 18 In Sem, Marks 4]**

Ans. : We will rewrite language L as

L = $a^n a^m b^m c^n$

Hence P = {
S → aSc | abcc
S → abAcc
A → bAc | ε
}

Q.14 Write the grammar generating all strings consisting of a's and b's with at least two a's. **[SPPU : Oct 18 In Sem, Marks 2]**

Ans. : We will write regular expression for this language

r.e. = $(a + b)^* a (a + b)^* a (a + b)^*$

The production rules $(a + b)^*$

$$A \rightarrow aA \mid bA \mid \varepsilon$$

Hence production rules for regular expression

$$S \rightarrow A X A X A$$

$$X \rightarrow a$$

$$A \rightarrow aA \mid bA \mid \varepsilon$$

Q.15 Write CFG for following language.

L = $\{a^n b^m a^n \mid n>=0,\ m>=1\}$ **[SPPU : Oct 18 In Sem, Marks 4]**

Ans. : The CFG can be given by following production rule.

P = {

$S \rightarrow aSa \mid bB$

$B \rightarrow bB \mid \varepsilon$

}

Simulation :

The string aabbbaa can be derived as

S

aSa

aaSaa

aabBaa

aabbBaa

aabbbBaa

aabbbaa

3.2 Sentential Form

Q.16 Write in brief about sentential form with reference to context free grammar. **[SPPU : Aug.17, Marks 3]**

Ans. : Consider G = (V, T, P, S) be a context free grammar then, we can derive a string w from it. This w can obtained from (VUT)* where V denotes the set of non-terminal symbols and T denotes the set of terminal symbols. The derivation of w from start symbol S can be written as $S \overset{*}{\Rightarrow} w$ which is called as **sentential form**. If $S \underset{lm}{\overset{*}{\Rightarrow}} w$ then w is a **left-sentential form.**

If $S \underset{rm}{\overset{*}{\Rightarrow}} w$ then w is a **right-sentential form.**

For example

Consider the grammar,

$$S \rightarrow S + S \mid S * S \mid 4$$

Then for deriving the string w = 4+4*4 we use above grammar as -

$$S \underset{lm}{\Rightarrow} S * S \underset{lm}{\Rightarrow} S + S * S \underset{lm}{\Rightarrow} 4 + S * S \underset{lm}{\Rightarrow} 4 + 4 * S \underset{lm}{\Rightarrow} 4 + 4 * 4$$

Here S+S*S is called left-sentential form.

$$S \underset{rm}{\Rightarrow} S + S \underset{rm}{\Rightarrow} S + S * S \underset{rm}{\Rightarrow} S + S * 4 \underset{rm}{\Rightarrow} S + 4 * 4 \underset{rm}{\Rightarrow} 4 + 4 * 4$$

S+S*4 is called right sentential form. Thus we can obtain the desired language L by certain rules. Let us now solve some examples for derivation of CFG.

3.3 Derivation and Derivation Tree/ Parse Tree

Q.17 Explain the term - derivation. Also explain leftmost derivation and rightmost derivation

Ans. : The production rules are used to derive certain strings. If G = (V, T, P, S) be some context free grammar then generation of some language using specific rules is called **derivation.**

- The **leftmost derivation** is a derivation in which the leftmost non terminal is replaced first from the sentential form.

- The **rightmost derivation** is a derivation in which rightmost non terminal is replaced first from the sentential form.

Let us see how it works.

For example

$S \rightarrow XYX$	$S \rightarrow XYX$
$S \rightarrow aYX$	$S \rightarrow XYa$
$S \rightarrow abX$	$S \rightarrow Xba$
$S \rightarrow aba$	$S \rightarrow aba$
Leftmost derivation	**Rightmost derivation**

Q.18 Write the CFG for language $L = \{0^i 1^j 0^k \mid j > i + k\}$. Show the derivation of the string '0111100'.

[SPPU : May-15, End Sem, Marks 4]

Ans. : The CFG can be

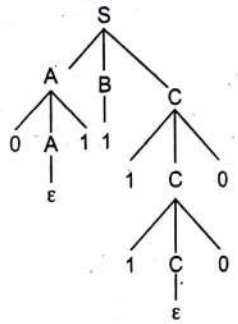
$S \rightarrow ABC$

$A \rightarrow 0A1 \mid \varepsilon$

$B \rightarrow 1B \mid 1$

$C \rightarrow 1C0 \mid \varepsilon$

Derivation of 0111100



Q.19 Define the term - Parse Tree

Ans. : Parse trees is a graphical representation for the derivation of the given production rules for a given CFG. It is the simple way to show how the derivation can be done to obtain some string from given set of production rules.

Q.20 Construct the parse tree for the string aabbabba from the CFG given by

$S \rightarrow aB \mid bA$

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

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

Ans. : To draw a tree we will first try to obtain derivation for the string *aabbabba*

S

*a*B $S \rightarrow aB$

a [*a*BB] $S \rightarrow aBB$

aa [bS] B $B \rightarrow bS$

aab [bA] B $S \rightarrow bA$

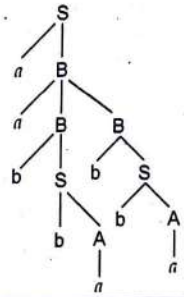
aabb [*a* B] $A \rightarrow a$

aabba [bS] $B \rightarrow bS$

aabbab [bA] $S \rightarrow bA$

*aa*Bb*ab*b [*a*] $A \rightarrow a$

Now let us draw a tree.



3.4 Ambiguous Grammar

Q.21 Explain the concept of ambiguity with suitable example.

Ans. : • **Definition :** If there exists more than one parse trees for a given grammar, that means there could be more than one leftmost or rightmost derivation possible and then that grammar is said to be ambiguous grammar.

• **Example**

For example : The CFG given by G = (V, T, P, S)

where V = {E}

T = {id}

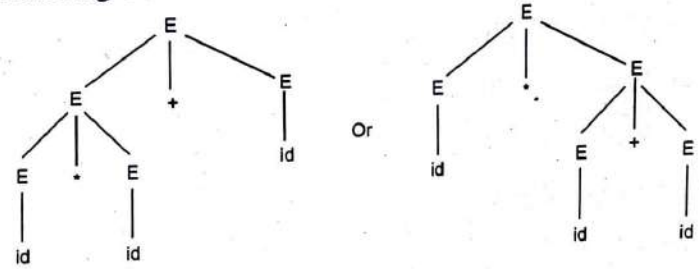
P = {E → E + E

E → E * E

E → id }

S = {E}

Now if the string is id * id + id then we can draw the two different parse trees indicating our id * id + id.

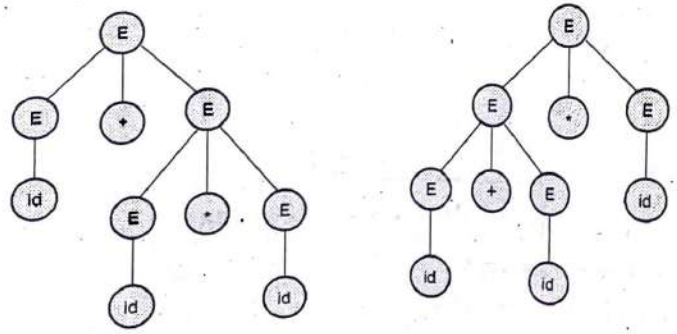


Thus the above grammar is an ambiguous grammar.

Q.22 Explain with suitable example, how to remove the ambiguity from the grammar.

Ans. : Consider the ambiguous grammar as -

E → E + E | E * E |id



(a) Parse tree 1 (b) Parse tree 2

Fig. Q.22.1 Ambiguous grammar

is an ambiguous grammar. We will design the parse tree for id + id * id as follows.

For removing the ambiguity we will apply one rule : If the grammar has left associative operator (such as +, –, *, /) then induce the left recursion and if the grammar has right associative operator (exponential operator) then induce the right recursion.

The unambiguous grammar is

E → E+T

E → T

T → T*F

T → F

F → id

Q.23 Give an ambiguous grammar for if then else statement and then re-write an equivalent unambiguous grammar.

☞ **[SPPU : May-14, Marks 8]**

Ans. : The ambiguous grammar for **if then else** statement is as follows -

S → iCtS

S → iCtSeS

S → a

C → b

Where i stands for **if**, C stands for **condition**, S stands for statement, e stands for **else**, and t stands for **then**. For making the given grammar unambiguous we match **else** with closest previous **unmatched then**. Hence on removing ambiguity we will get following rules -

S → M|U

M → iCtMeM |a

U → iCtS | iCtMeU

C → b

Here M and U non-terminals for matched and unmatched statments.

Q.24 Given a CFG as :

S → aB | bA

A → a | aS |bAA

B → b | bS |a BB

Is ab, baba, abbbaa in L (G) ? What is the language of this CFG ? Is the CFG ambiguous ? **☞[SPPU : May-12, Marks 8]**

Ans. : The {ab, baba, abbbaa}∈ L (G). The language of this CFG is L which contains equal number of a's and b's. The given CFG is ambiguous because consider the string bbabaa.

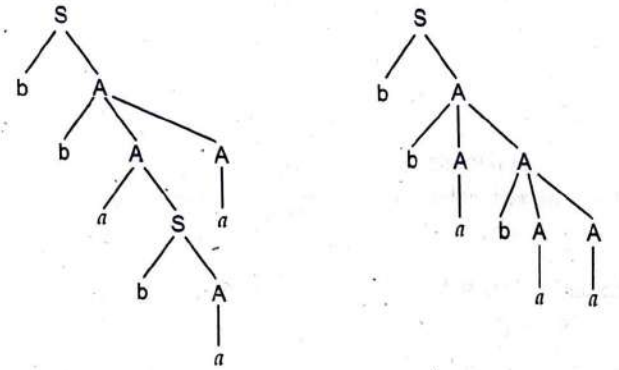


Fig. Q.24.1

Q.25 Consider the grammar having production.

S → aS | ε

S → a S b S

This grammar is ambiguous.

i) Show in particular that the string aab has two parse trees.

ii) Find an unambiguous grammar for the same.

☞[SPPU : Dec.-05, Marks 8]

Ans. : Given grammar with production

S → a S | ε

S → a Sb S.

i) a a b has following parse trees.

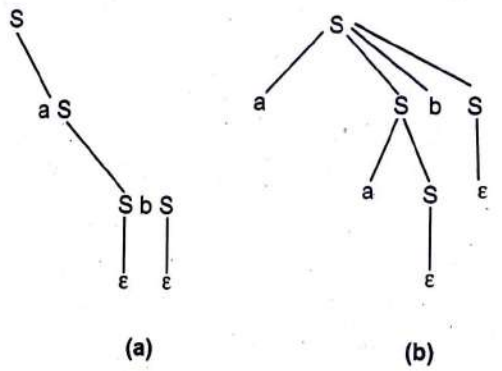


Fig. Q.25.1

Unambiguous grammar for above is

S→ a S | XS | ε

X→ aXXb | ε

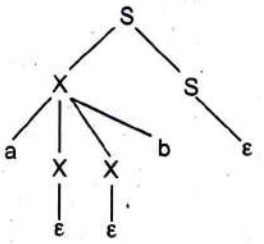


Fig. Q.25.2

Q.26 Let G be the grammar

S → aB|bA

A → a|aS|bAA

B → b|bS|aBB

For the string aaabbabbba find

1) Leftmost derivation 2) Rightmost derivation

3) Parse tree 4) Is the grammar unambiguous

☞[SPPU : May-07, Marks 8; Dec.-14, End Sem, Marks 4]

Ans. : 1) Leftmost derivation :

$S \rightarrow aB$
$\rightarrow aaBB$
$\rightarrow aaaBBB$
$\rightarrow aaabSBB$
$\rightarrow aaabbABB$
$\rightarrow aaabbaBB$
$\rightarrow aaabbababB$
$\rightarrow aaabbabbS$
$\rightarrow aaabbabbbA$
$\rightarrow aaabbabbba.$

2) Rightmost derivation :

$S \rightarrow aB$
$\rightarrow aaBB$
$\rightarrow aaBbS$
$\rightarrow aaBbbA$
$\rightarrow aaBbba$
$\rightarrow aaaBBbba$
$\rightarrow aaaBbbba$
$\rightarrow aaabSbbba$
$\rightarrow aaabbAbbba$
$\rightarrow aaabbabbba.$

3) Parse tree :

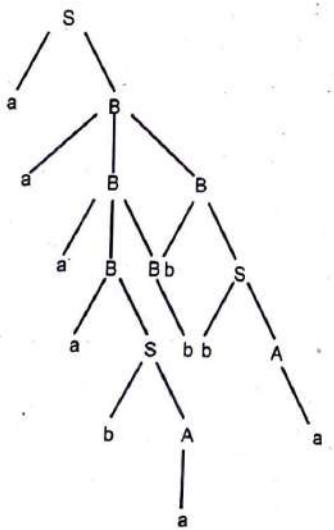
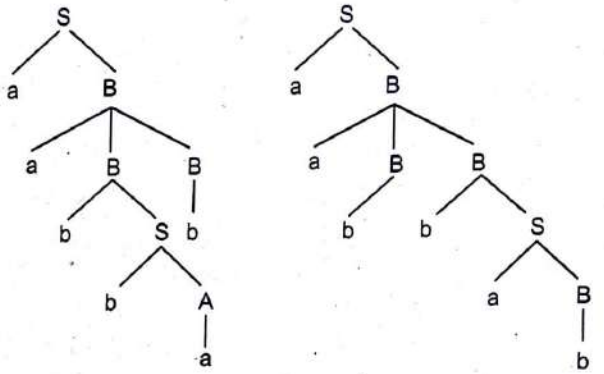


Fig. Q.26.1

4) No, the grammar is ambiguous. Following derivations for **aabbab** shows -



Q.27 What is ambiguous grammar ? Show that the grammar below is ambiguous and find the equivalent unambiguous grammar.
i) $S \rightarrow SS \mid a \mid b$, ii) $S \rightarrow ABA$, $A \rightarrow aAb \mid \varepsilon$, $B \rightarrow bB$

☞ [SPPU : May-17 (End Sem), Marks 8]

Ans. : When more than one different parse trees can be generated using the given grammar then that grammar is said to be ambiguous grammar.

i) Consider string **aab**. The Parse trees for generating this string are :

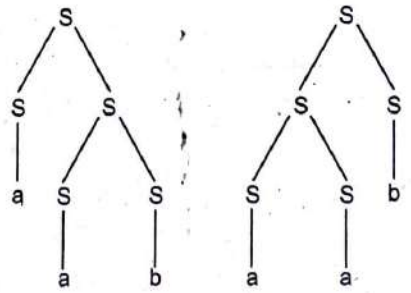


Fig. Q.27.1

The unambiguous grammar will be

$$S \rightarrow X \mid a \mid b$$

$$X \rightarrow Xa \mid Xb \mid \varepsilon$$

ii) In given grammar, the rule $b \rightarrow bB$ gives rise to infinite productions. Hence B seems to be invalid symbol. Due to this the grammar is invalid.

Q.28 Write in brief about "Sentential form" with reference to context free grammar. ☞ **[SPPU : Aug.-17 In Sem, Marks 3]**

Ans. : Consider G = (V, T, P, S) be a context free grammar then, we can derive a string w from it. This w can obtained from (VUT)* where V denotes the set of non-terminal symbols and T denotes the set of terminal symbols. The derivation of w from start symbol S can be written as $S \overset{*}{\Rightarrow} w$ which is called as **sentential form.** If $S \underset{lm}{\overset{*}{\Rightarrow}} w$ then w is a **left-sentential form.**

If $S \underset{rm}{\overset{*}{\Rightarrow}} w$ then w is a **right-sentential form.**

For example

Consider the grammar,

$$S \rightarrow S + S \mid S * S \mid 4$$

Then for deriving the string w = 4+4*4 we use above grammar as -

$$S \underset{lm}{\Rightarrow} S * S \underset{lm}{\Rightarrow} S + S * S \underset{lm}{\Rightarrow} 4 + S * S \underset{lm}{\Rightarrow} 4 + 4 * S \underset{lm}{\Rightarrow} 4 + 4 * 4$$

Here S+S*S is called left-sentential form.

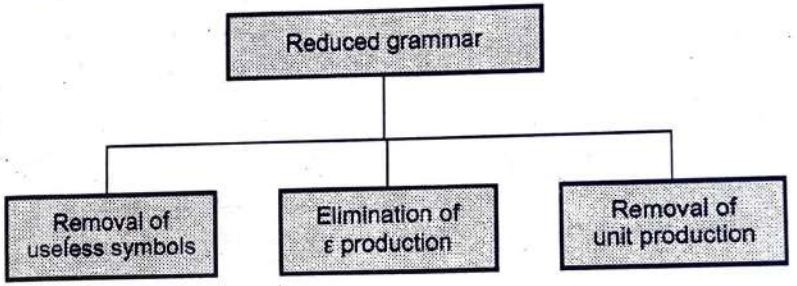
$$S \underset{rm}{\Rightarrow} S + S \underset{rm}{\Rightarrow} S + S * S \underset{rm}{\Rightarrow} S + S * 4 \underset{rm}{\Rightarrow} S + 4 * 4 \underset{rm}{\Rightarrow} 4 + 4 * 4$$

S+S*4 is called right sentential form. Thus we can obtain the desired language L by certain rules. Let us now solve some examples for derivation of CFG.

3.5 Simplification of CFG

Q.29 What do you mean by simplification of grammar ?

Ans. : Simplification of grammar means reduction of grammar by removing useless symbols.



Q.30 Eliminate the useless symbols from following grammar.

S → aA | a | Bb | cC
A → aB
B → a | Aa
C → cCD
D → ddd

Ans. : Consider all the productions that are giving terminal symbols.

$$S \rightarrow a$$

$$B \rightarrow a$$

$$D \rightarrow ddd$$

Now consider

$$S \rightarrow cC$$

$$C \rightarrow cCD$$

$$D \rightarrow ddd.$$

When we try to derive the string using production rule for C we get,

$S \rightarrow cC \rightarrow ccCD \rightarrow ccCddd \rightarrow cccCDddd \rightarrow cccCdddddd \rightarrow \ldots$

We will not get any terminal symbol for C. Thus we get a useless symbol C. To reach to D the only rule available is by using C. But as C gets eliminated, there is no point in keeping D. Hence D also will be removed.

$\therefore$ $S \rightarrow aA \mid a \mid Bb$

$A \rightarrow aB$

$B \rightarrow a \mid Aa$

is the reduced grammar.

Q.31 Construct CFG without ε production from the one which is given below.

$S \rightarrow a|Ab|aBa$

$A \rightarrow b|\varepsilon$

$B \rightarrow b|A$

Ans. : If you observe carefully, then not only A have ε production but even B also indicates ε production i.e. $A \rightarrow \varepsilon$ straight forward but $B \rightarrow A \rightarrow \varepsilon$. Let us apply the method of replacement.

$S \rightarrow A\ b$

if $A = \varepsilon$ then

$S \rightarrow b$

if $B = \varepsilon$

$S \rightarrow a\ a$

$S \rightarrow a|A\ b|b|a\ a|a\ B\ a$

$A \rightarrow b$

$B \rightarrow b$

Finally the rules are

$S \rightarrow a|A\ b|b|a\ á|a\ B\ a$

$A \rightarrow b$

$B \rightarrow b$

Q.32 Consider the CFG given below,

For eliminating ε productions

$S \rightarrow P\ 0\ Q \mid Q\ Q \mid 0\ R \mid R\ Q\ P$

$P \rightarrow R\ 0 \mid 1\ R \mid R\ R \mid R\ Q\ P$

$Q \rightarrow Q\ 0 \mid P\ Q|\varepsilon$

$R \rightarrow 0\ P \mid Q\ Q\ Q$

Ans. : As we can see that $Q \rightarrow \varepsilon$.

Similarly $R \rightarrow QQQ \rightarrow \varepsilon$.

Hence we will replace Q and R by ε and accordingly add the production rules.

Similarly $P \rightarrow R\ R \rightarrow \varepsilon$

Let us modify the rules if P or Q or R becomes ε.

$S \rightarrow P\ 0\ Q|0|0\ R|Q\ Q|R\ Q\ P$

$P \rightarrow R\ 0|0|1|1\ R|RR|RQP$

$Q \rightarrow Q\ 0|0|PQ$

$R \rightarrow 0\ P|0|Q\ Q\ Q$

Note that with P, Q, R other meanings are associated, they are not purely giving ε to us. Therefore we cannot remove P, Q and R symbols.

Q.33 Eliminate the unit productions from following grammar

S → AB

A → a

B → C|b

C → D

D → E|bC

E → d|Ab

Ans. : As,

$S \rightarrow AB$ and

$A \rightarrow a$

There is only one rule with A and that too giving terminal symbol. Hence there is no question of getting unit production with A. Now consider

$B \rightarrow C$

$C \rightarrow D$

$D \rightarrow E$

It is clear that B, C and D are unit productions. As $E \rightarrow d|Ab$, we will replace value of D. Then,

$D \rightarrow d \mid Ab \mid bC$

Similarly as $C \rightarrow D$ we can write

$C \rightarrow d|Ab|bC$

Hence B becomes

$B \rightarrow d|Ab|bC|b$

Thus the grammar after removing unit productions

$S \rightarrow AB$

$A \rightarrow a$

$B \rightarrow d|Ab|bC|b$

$C \rightarrow d|Ab|bC$

$D \rightarrow d|Ab|bC$

$E \rightarrow d|Ab.$

Now there is no path for D and E. From start state we can remove them by considering useless symbols. The optimized grammar will be

$S \rightarrow AB$

$A \rightarrow a$

$B \rightarrow d|Ab|bC|b$

$C \rightarrow d|Ab|bC$

Q.34 Simplify the following grammar :

i) $S \rightarrow Ab$, $A \rightarrow a$, $B \rightarrow C \mid b$, $C \rightarrow D$, $D \rightarrow E$, $E \rightarrow a$

ii) $S \rightarrow 0A0 \mid 1B1 \mid BB$, $A \rightarrow C$, $B \rightarrow S \mid A$, $C \rightarrow S \mid \varepsilon$

☞[SPPU : Dec.-16, End Sem, Marks 8, Oct.-19, Marks 4]

Ans. : i) Let,

$S \rightarrow Ab$

$A \rightarrow a$

$B \rightarrow C \mid b$

$C \rightarrow D$

$D \rightarrow E$

$E \rightarrow a$

Step 1 : Here $B \rightarrow C$, $C \rightarrow D$, $D \rightarrow E$ and $E \rightarrow a$, are unit productions. These can be eliminated and we get simply $B \rightarrow a \mid b$.

Step 2 : Now if we observe the start state S, we can clearly say that non terminal B is non-reachable. Hence eliminate it as useless symbol.

Step 3 : Thus we get the simplified grammar, on removing B, C, D and E as

$S \rightarrow Ab$

$A \rightarrow a$

ii) Let,

$S \rightarrow 0A0$

$S \rightarrow 1B1$

$S \rightarrow BB$

$A \rightarrow C$

$B \rightarrow S \mid A$

$C \rightarrow S \mid \varepsilon$

Step 1 : The $A \rightarrow C$, $C \rightarrow S \mid \varepsilon$, are unit productions. These can be eliminated $A \rightarrow S \mid \varepsilon$.

Step 2 : $B \rightarrow A$, $A \rightarrow S \mid \varepsilon$ are unit productions, these can be eliminated by $B \rightarrow \varepsilon$

Step 3 : As we have

$S \rightarrow 0A0 \mid 1B1$

Also $A \rightarrow S$ and $B \rightarrow S$

$\therefore$ We can reduce it as

$S \rightarrow 0S0 \mid 1S1$

Step 4 : Similarly $S \rightarrow BB$ can be reduced.

Step 5 : Finally we get

$S \rightarrow 0S0 \mid 1S1 \mid \varepsilon$

as simplified productions.

Q.35 Eliminate $\in$ - production from the grammar G

$A \rightarrow aBb|bBa$

$B \rightarrow aB|bB|\in$. **☞ [SPPU : Aug.-17, In Sem, Marks 3]**

Ans. : $B \rightarrow \varepsilon$ is used at RHS of each production rule. On eliminating ε production we get

$A \rightarrow aBb \mid bBa \mid ab \mid ba$

$B \rightarrow aB \mid bB \mid a \mid b$

3.6 : Normal Forms

3.6.1 Chomsky's Normal Form

Q.36 What is Chomsky's Normal Form ?

Ans. : The Chomsky's Normal Form can be defined as

Non terminal → Non terminal · Non terminal
Non terminal → Terminal

Q.37 Convert the given CFG to CNF.

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

Where V = {S, A, B}

T = {a, b}

P consists of

S → *a*B **A → bAA**

S → bA **B → b**

A → *a* **B → *b*S**

A → *a*S **B → *a*BB**

☞ **[SPPU : Dec.-05, Marks 12; Dec.-14, In Sem, Marks 6; Dec.-16, End Sem, Marks 8]**

Ans. : Let us start with first rule.

$R_1 \rightarrow a$

$S \rightarrow R_1B$ is in CNF for $S \rightarrow aB$

$R_2 \rightarrow b$

$S \rightarrow R_2A$ is in CNF for $S \rightarrow bA$

$A \rightarrow a$

$A \rightarrow a\,S$ can be written as

$A \rightarrow R_1S$

Now $A \rightarrow bAA$ can be written as

$A \rightarrow R_2\ AA$ replace it by R_3

i.e. $R_3 \rightarrow AA$

then $A \rightarrow R_2R_3$

$B \rightarrow a$

$B \rightarrow bS$ can be written as

$B \rightarrow R_2S$ is in CNF

Now, $B \rightarrow aBB$

i.e. $B \rightarrow R_1BB$ $\because$ $R_1 \rightarrow a$

Let $R_4 \rightarrow BB$

then $B \rightarrow R_1\ R_4$

Finally we can write,

$S \rightarrow R_1B$

$S \rightarrow R_2A$

$A \rightarrow R_1S$

$A \rightarrow R_2\ R_3$

$B \rightarrow b$

$B \rightarrow R_2S$

$B \rightarrow R_1\ R_4$

$R_1 \rightarrow a$

$R_2 \rightarrow b$

$R_3 \rightarrow AA$

$R_4 \rightarrow BB$

is a Chomsky's normal form.

3.6.2 : Greibach Normal Form

Q.38 What is Greibach Normal Form ?

Ans. : The rule for GNF is

Non-terminal → One terminal · Any number of non-terminals

Q.39 Convert the given CFG to GNF.

$S \rightarrow ABA$

$A \rightarrow aA|\varepsilon$

$B \rightarrow bB|\varepsilon$

☞ **[SPPU : Dec.-10, Marks 6]**

Ans. : As before converting the CFG to its normal form we usually reduce the grammar i.e. we first eliminate ε production, unit production and useless symbols.

As $A \rightarrow \varepsilon$ and $B \rightarrow \varepsilon$

Let us simplify the given CFG by removing ε productions and the CFG will look like this,

$S \rightarrow ABA|AB|BA|AA|A|B$

$A \rightarrow aA|a$

$B \rightarrow bB|b$

[We can even remove the unit productions]

Let us replace A by aA or a in S

$S \rightarrow aABA|aAB|aBA|aB|aAA|aA|a$

which is GNF

Similarly $S \rightarrow BA|B$ can be written as

$S \rightarrow bBA|bA|bB|b$

is also in GNF.

Finally equivalent GNF will be -

$S \rightarrow aABA|aAB|aBA|aB|aAA|aA|a$

$S \rightarrow bBA|bA|bB|b$

$A \rightarrow aA|a$

$B \rightarrow bB|b$

Q.40 Convert given CFG to GNF where V = {S, A}, T = {0, 1} and P is

$S \rightarrow AA|0$

$A \rightarrow SS|1$

☞ **[SPPU : Dec.-12, Marks 6]**

Ans. : Let us rename S as A_1 and A as A_2 then given CFG becomes

$A_1 \rightarrow A_2A_2|0$

$A_2 \rightarrow A_1A_1|1$

Let us start with A_2. The rule for A_2 is

$A_2 \rightarrow A_1A_1|1$

Now replace first A_1 on R.H.S. by rule of A_1

$\therefore \quad A_2 \rightarrow A_2A_2A_1|0A_1|1$

According to lemma 1 if

$A \rightarrow Aa_1 \mid Aa_2 \mid \ldots \mid Aa_n \mid \beta_1 \mid \beta_2 \mid \ldots \mid \beta_n$

Then,

$A \rightarrow \beta_1 \mid \beta_2 \ldots \mid \beta_n$

$A \rightarrow \beta_1Z \mid \beta_2Z \ldots \mid \beta_nZ$

$Z \rightarrow a_1 \mid a_2 \mid \ldots \mid a_n$

$Z \rightarrow a_1Z \mid a_2Z \mid \ldots \mid a_nZ$

We can map this lemma to our A_2 rule as $A = A_2$, $a_1 = A_2A_1$, $\beta_1 = 0A_1$ and $\beta_2 = 1$.

Then we get,

$A_2 \rightarrow 0A_1|1$

$A_2 \rightarrow 0A_1Z \mid 1Z$

$Z \rightarrow A_2A_1$

$Z \rightarrow A_2A_1Z$

Thus now we have obtained A_2 being in GNF. Now, consider production for A_1.

As $A_1 \rightarrow A_2A_2 \mid 0$

We will replace first A_2 on R.H.S. by its recently GNF rules, then we get,

$A_1 \rightarrow 0A_1A_2 \mid 1A_2 \mid 0A_1ZA_2 \mid 1ZA_2 \mid 0 \rightarrow$ using lemma 2

Thus now A_1 is also in GNF.

Now, consider Z rule.

$Z \rightarrow A_2A_1$ } using lemma 2

$Z \rightarrow A_2A_1Z$ } replace A_2 at R.H.S.

Hence we get,

$Z \rightarrow 0A_1A_1 \mid 1A_1 \mid 0A_1ZA_1 \mid 1ZA_1$

$Z \rightarrow 0A_1A_1Z \mid 1A_1Z \mid 0A_1ZA_1Z \mid 1ZA_1Z$

Now let us rewrite the rules by converting back $A_1 = S$ and $A_2 = A$

$$S \rightarrow 0SA \mid 1A \mid 0SZA \mid 1ZA \mid 0$$

$$A \rightarrow 0S \mid 1 \mid 0SZ \mid 1Z$$

$$Z \rightarrow 0SS \mid 1S \mid 0SZS \mid 1ZS$$

$$Z \rightarrow 0SSZ \mid 1SZ \mid 0SZSZ \mid 1ZSZ$$

is equivalent GNF.

Q.41 Convert the grammar $S \rightarrow AB$, $A \rightarrow BS|b$, $B \rightarrow SA|a$ into Greibach normal form.

[SPPU : Dec.-06,07, May-07, Marks 8; Dec.-14, End Sem, Marks 4]

Ans. : Consider the grammar

$$S \rightarrow AB$$

$$A \rightarrow BS|b$$

$$B \rightarrow SA|a$$

Now assume $S = A_1$, $A = A_2$ and $B = A_3$ then we can rewrite the rules as-

$$A_1 \rightarrow A_2 A_3$$

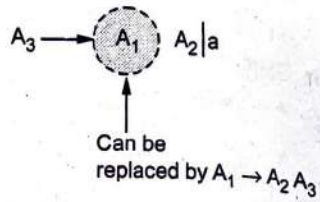
$$A_2 \rightarrow A_3 A_1 | b$$

$$A_3 \rightarrow A_1 A_2 | a$$

Now we will consider A_3 productions and let us apply two important lemma required to convert given CFG to GNF.

$$A_3 \rightarrow A_1 A_2 | a$$

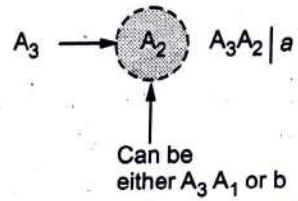
The lemma 1 suggests us to replace A_1 present in R.H.S. of rule A_3 by the rule of A_1. Hence



Hence we get

$$A_3 \rightarrow A_2 A_3 A_2 | a$$

Again in rule A_3 we get $A_2 A_3 A_2$ i.e. we can replace first A_2 by its R.H.S. production. By this we are actually applying lemma 1. Hence



Hence we get,

$$A_3 \rightarrow A_3 A_1 A_3 A_2 \mid bA_3 A_2 | a$$

Now let us apply lemma 1 on this production.

Lemma 1 is -

If there is a rule

$$A \rightarrow Aa_1 | Aa_2 | Aa_3 | \ldots Aa_n \ldots | \beta_1 | \beta_2 | \beta_3 | \ldots | \beta_k$$

Such that β_i do not start with A then

$$A \rightarrow \beta_1 | \beta_2 | \beta_3 \ldots | \beta_k$$

$$A \rightarrow \beta_1 Z | \beta_2 Z$$

$$Z \rightarrow a_i$$

$$Z \rightarrow a_i Z$$

Hence for rule

$A_3 \rightarrow A_3 A_1 A_3 A_2 | bA_3 A_2 | a$ we get

$A = A_3$, $\beta_1 = bA_3 A_2$, $\beta_2 = a$, $a_1 = A_1 A_3 A_2$ we get,

$$A_3 \rightarrow bA_3 A_2 | a \quad \ldots (1)$$

$$A_3 \rightarrow bA_3 A_2 Z | aZ \quad \ldots (2)$$

$$Z \rightarrow A_1 A_3 A_2$$

$$Z \rightarrow A_1 A_3 A_2 Z$$

Now if we observe A_3 productions denoted in equations (1) and (2) we get it in GNF form i.e. NT → t · NT form. Finally we can rewrite A_3 as

$$\boxed{A_3 \rightarrow bA_3A_2 | a \mid bA_3A_2Z | aZ} \quad \ldots (3)$$

Now consider A_2 production and let us apply lemma 1 for this rule.

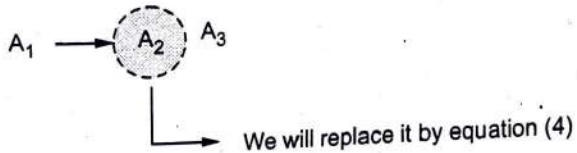
$$A_2 \rightarrow A_3 A_1 | b$$

Replace it by equation (3), we get,

$$A_2 \rightarrow bA_3\ A_2\ A_1 | aA_1 | bA_3A_2ZA_1 | aZA_1 | b \quad \ldots (4)$$

Clearly, A_2 is now in GNF.

Now let us concentrate on $A_1 . A_S$



Hence

$$A_1 \rightarrow bA_3A_2A_1A_3 | aA_1A_3 | bA_3A_2ZA_1A_3 | aZA_1A_3 | bA_3 \quad \ldots (5)$$

is in GNF form. Now the only remaining production of conversion is Z. Let us deal with it. As

$$Z \rightarrow A_1A_3A_2 | A_1A_3A_2Z$$

At R.H.S. A_1 comes as the first symbol. Hence instead of A_1 if we put it R.H.S. we get,

$$Z \rightarrow bA_3A_3A_2bA_3A_2A_1A_3A_3A_2 | aA_1A_3A_3A_2 | bA_3A_2ZA_1A_3A_3A_2 | aZA_1A_3A_3A_2 |$$

$$Z \rightarrow bA_3A_2A_1A_3A_3A_2Z | aA_1A_3A_3A_2Z | bA_3A_2ZA_1A_3A_3A_2Z | aZA_1A_3A_3A_2Z | bA_3A_3A_2Z$$

Thus we get Z production also in GNF form.

Q.42 What are different types of normal forms of context free grammar ? ☞ **[SPPU : Dec.-19, Marks 6]**

Ans. : Chomsky Normal Form : Refer Q.36 and Q.37.

Greibach Normal Form: Refer Q.38 and Q.41.

3.7 Pumping Lemma for CFG

Q.43 State and prove pumping lemma for CFG.

Ans. : Lemma : Let L be any context free language, then there is a constant n, which depends only upon L, such that there exist a string $w \in L$ and $|w| \geq n$ where $w = pqrst$ such that

1. $|qs| \geq 1$
2. $|prs| \leq n$ and
3. For all $i \geq 0$ $p\ q^i\ r\ s^i\ t$ is in L.

Proof :

- This pumping lemma states that if there is a language L which is without unit productions and without a null production and there exist w where $w \in L$.
- The string w can be derived by a context free grammar G. The G be a grammar which is in Chomsky's Normal Form.
- The grammar G generates language L. For the string w, we can obtain a parse tree which derives the string w. Then if the length of the path to w is less than equal to i then the length of the word w is less than or equal to 2^{i-1}.
- We can prove this by induction step.

Basis : If i = 1

Let G contains the rule $S \rightarrow a$ where length of the derived string is 1 i.e. i = 1. Now according to the rule the word length should be $\leq 2^{i-1}$ i.e. $2^0 = 1$. Observe that we have a word 'a' which is of length 1. Also observe that the grammar G is in Chomsky's Normal Form. This language is regular since $|w| = |pqrst| = 1$.

Fig. Q.43.1

Induction step : Let w be a string which is derived by grammar G. Let k be a variable such that $n = 2^k$, $|w| \geq n$ then $|w| > 2^{k-1}$ while deriving w string we may get some non-terminals of CFG - G can be repeated for any number of times and will give the string w. If we pump the substrings to w such that the path length of this newly formed string w′ (w + pumped string = w′) is i and the word

length of w′ is $\leq 2^{i-1}$ then the grammar G deriving w′ is called a regular grammar. The necessary condition is that grammar G is in Chomsky's Normal Form.

Let us consider a grammar,

$G = (\{A, B, C\}, \{a\}, \{A \rightarrow BC$

$B \rightarrow BA$

$C \rightarrow BA$

$A \rightarrow a$

$B \rightarrow b\}, A)$

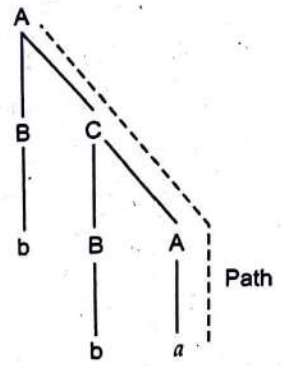


Fig. Q.43.2

Thus

$$A \overset{*}{\Rightarrow} bba = w$$

i.e. path length i = 3

$|w| \leq 2^{i-1}$ i.e. $3 \leq 2^2$ If we pump a substring into w which satisfies the condition as $i \leq |w| \leq 2^{i-1} \leq n$ the grammar producing string w is a regular grammar.

Q.44 Show that $L = \{a^n b^n c^n | n \geq 0\}$ is not a context free language.

Ans. : Let us assume that

$L = a^n b^n c^n$ is a context free language.

Let, w be any string such that $w \in L$.

Let,

$w = pqrst$

Let, $|qs| \geq 1$ $|prs| \leq n$

Now let us consider

$w = pq^i rs^i t$

i.e. we have additional occurrences of q and s. Consider various cases.

Case 1 : Consider i = 0 then,

$w = pq^i rs^i t$

if i = 0 then q^0 and s^0. That means q and s are absent then,

$w = prt$

w = aa bb cc
p q r s t

$= a^2bc$

i.e $= aabc \notin L$

Hence our assumption of L being CFG is wrong.

Case 2 : Consider i = 2 then,

$w = pq^i rs^i t$

$= pqqrsst$

Consider,

w = aa bb cc
p q r s t

Now with i = 2 we get,

$= a^2 b^3 c^3 \notin L$

w = aa bbb ccc
p qq r ss t

Thus our assumption of L being CFG is wrong. This proves that the given language L is not a context free.

3.8 Closure properties of CFL

Q.45 Write a note on closure properties of CFL.

☞[SPPU : May 09, Marks 4]

Ans. : The context free languages are closed under some operation means after performing that particular operation on those CFLs the resultant language is context free language. These properties are as below

1. The context free languages are closed under union.
2. The context free languages are closed under concatenation.

3. The context free languages are closed under kleen closure.
4. The context free languages are not closed under intersection.
5. The context free languages are not closed under complement.

3.9 Decision properties of CFL

Q.46 What are the decision properties of CFL ?

Ans. : Various decision properties of CFL are as follows :

1. **Emptiness :** There exists an algorithm which can determine whether or not the given context free grammar can generate any word at all.
2. **Finiteness :** There exists an algorithm to determine whether the given context free grammar generates a finite or infinite language.
3. **Membership :** There exists an algorithm which tells whether given string belongs to given grammar.

3.10 Chomsky Hierarchy

Q.47 Write a short note on Chomsky hierarchy.

☞ [SPPU : Dec 2011, Marks 8]

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

Type 3 - Regular languages

Regular languages are those languages which can be described using regular expressions. These languages can be modelled by NFA or DFA.

Type 2 - Context free languages

The context free languages are the languages which can be represented by Context Free Grammar (CFG). The production rule is of the form

$$A \rightarrow \alpha$$

where A is any single non-terminal and α is any combination of terminals and non-terminals.

A NFA or DFA cannot recognize strings of this language because these automata are not having "stack" to memorize. Instead of it the Push Down Automata can be used to represent these languages.

Type 1 - Context sensitive languages

The context sensitive grammars are used to represent context sensitive languages. The context sensitive grammar is follows the following rules -

1. The context sensitive grammar may have more than one symbol on the left hand side of their production rules.
2. The number of symbols on the left hand side must not exceed the number of symbols on the right hand side.
3. The rule of the form $A \rightarrow \varepsilon$ is not allowed unless A is a start symbol. It does not occur on the right hand side of any rule.

The automaton which recognizes context sensitive languages is called linear bounded automaton. While deriving using context sensitive grammar the sentential form must always increase in length every time a production rule is applied. Thus the size of a sentential form is bounded by a length of the sentence we are deriving.

Type 0 - Unrestricted languages

There is no restriction on the grammar rules of these type of languages. These languages can be effectively modeled by turing machines.

3.11 Cock-Younger-Kasami Algorithm

Q.48 State and explain Cock-Younger-Kasami Algorithm with an illustrative example

Ans. : Cock-Younger-Kasami or CYK algorithm is used to decide whether a string belongs to language of grammar or not.

- It is also called as **membership algorithm** of context free grammar.
- CYK algorithm works only on CFG which are in Chomsky-Normal-Form (CNF).

Algorithm

```
Begin
for ( i = 1 to n do )
Vi1 { A | A → a is a production where ith symbol of x is a }
for ( j = 2 to n do )
for ( i = 1 to n - j + 1 do )
Begin
Vij = → Φ
For k = 1 to j - 1 do
Vij = Vij → ∪
{A | A → BC is a production where B is in Vik and C is in V(i + k)(j - k) }
End
End
```

Let us understand this algorithm with some suitable example.

Consider the following grammar to check the acceptance of the string **w = baaba**

$S \rightarrow AB \mid BC$

$A \rightarrow BA \mid a$

$B \rightarrow CC \mid b$

$C \rightarrow AB \mid a$

The string can be represented in a triangular table as -

x_{15}				
x_{14}	x_{24}			
x_{13}	x_{23}	x_{33}		
x_{12}	x_{22}	x_{32}	x_{42}	
x_{11}	x_{21}	x_{31}	x_{41}	x_{51}

baaba				
baab	aaba			
baa	aab	aba		
ba	aa	ab	ba	
b	a	a	b	a

The string 'x' has following positions -

Notation for Non-terminal V_{ij}

The V_{ij} represents a **set of variables** in the grammar which can drive the substring x_{ij}.

If set of variables contain start symbol then -

1) The substring x_{ij} can be derived from given grammar.

2) The substring x_{ij} is a memeber of the language of the given grammar.

The triangular table for V_{ij} is as given below -

V_{15}				
V_{14}	V_{24}			
V_{13}	V_{23}	V_{33}		
V_{12}	V_{22}	V_{32}	V_{42}	
V_{11}	V_{21}	V_{31}	V_{41}	V_{51}

Step 1 : We will fill up x_{ij} and V_{ij} table as -

- **Finding x_{11} and V_{11}** - The V_{11} represents the set of variables deriving x_{11}. Here x_{11} = b. From given CFG, only non-terminal B derives 'b' in given grammar.

  $\therefore \quad x_{11} = b, \qquad V_{11} = \{B\}$

- **Finding x_{21} and V_{21}** - The V_{21} represents the set of variables deriving x_{21}. Here x_{21} = a. From given CFG, only non-terminals A, C derives 'a' in given grammar.

  $\therefore \quad x_{21} = a, \qquad V_{21} = \{A, C\}$

- **Finding x_{31} and V_{31}** - The V_{31} represents the set of variables deriving x_{31}. Here x_{31} = a. From given CFG, only non-terminals A and C derive 'a'.

  $\therefore \quad x_{31} = a, \qquad V_{31} = \{A, C\}$

- **Finding x_{41} and V_{41}** - The V_{41} represents the set of variables deriving x_{41}. Here x_{41} = b. From given CFG, the non-terminal B derives 'b'.

  $\therefore \quad x_{41} = b, \qquad V_{41} = \{B\}$

- **Finding x_{51} and V_{51}** - The V_{51} represents the set of variables deriving x_{51}. Here x_{51} = a. From given CFG, the non-terminals A and C derive 'a'.

  $\therefore \quad x_{51} = a, \qquad V_{51} = \{A, C\}$

The x_{ij} and V_{ij} table, partially filled up with bottom-most row is -

The x_{ij} table

b	a	a	b	a

The V_{ij} table

{B}	{A, C}	{A, C}	{B}	{A, C}

Step 2 : Now for filling up the table in upward direction from the 2nd row from bottom, we will use following formula -

$$V_{ij} = V_{ik}\ V_{(i+k)(j-k)}$$

where k varies from 1 to j – 1.

- **Finding x_{12} and V_{12} -** We have i = 1, j = 2, k = 1

Here x_{12} = 'ba'

$V_{12} = V_{11}\ V_{(1+1)(2-1)}$

$= V_{11}\ V_{21}$

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

$= \{BA, BC\}$

$\therefore\ V_{12} = \{A, S\}$

- **Finding x_{22} and V_{22} -** We have i = 2, j = 2, k = 1

Here x_{22} = 'aa'

$V_{ij} = V_{ik}\ V_{(i+k)(j-k)}$

$V_{22} = V_{21}\ V_{31}$

$V_{22} = \{A, C\}\ \{A, C\}$

$= \{AA, AC, CA, CC\}$

The AA, AC and CA do not exist.

$\therefore\ V_{22} = \{CC\}$

$\therefore\ V_{22} = \{B\}$

- **Finding x_{32} and V_{32} -** We have i = 3, j = 2, k = 1

Here x_{32} = 'ab'

$V_{ij} = V_{ik}\ V_{(i+k)(j-k)}$

$V_{32} = V_{31}\ V_{41}$

$V_{32} = \{A, C\}\ \{B\}$

$= \{AB, CB\}$

As 'CB' can not be derived from any non-terminal,

$V_{32} = \{AB\}$

$\therefore\ V_{32} = \{S, C\}$

- **Finding x_{42} and V_{42} -** We have i = 4, j = 2, k = 1

Here x_{42} = 'ba'

$V_{ij} = V_{ik}\ V_{(i+k)(j-k)}$

$V_{42} = V_{41}\ V_{51}$

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

$= \{BA, BC\}$

$\therefore\ V_{42} = \{A, S\}$

The partially filled x_{ij} and V_{ij} tables are

The x_{ij} table

ba	aa	ab	ba	
b	a	a	b	a

The V_{ij} table

{A, S}	{B}	{S, C}	{A, S}	
{B}	{A, C}	{A, C}	{B}	{A, C}

Step 3 :

- **Finding x_{13} and V_{13} -** Here i = 1, j = 3, k = j – 1, (1, 2)

$V_{ij} = V_{ik}\ V_{(i+k)(j-k)}$

$V_{13} = V_{11}\ V_{22} \cup V_{12}\ V_{41}$

$= \{B\}\ \{B\} \cup \{A, S\}\ \{A, C\}$

$= \{BB\} \cup \{AA, SA, AC, SC\}$

As we can not get BB, AA, SA, AC, SC.

$\therefore\ V_{13} = \phi \cup \phi$

$V_{13} = 0$

- **Finding x_{23} and V_{23} -** Here i = 2, j = 3, k = (3 – 1) = (1, 2)

$x_{23} = aab$

$V_{ij} = V_{ik}\ V_{(i+k)(j-k)}$

$V_{23} = V_{21}\ V_{32} \cup V_{22}\ V_{41}$

$= \{A, C\}\ \{S, C\} \cup \{B\}\ \{B\}$

$= \{AS, AC, CS, CC\} \cup \{BB\}$

Out of the above, only CC can be obtained.

$\therefore\ V_{23} = \{CC\}$

$\therefore\ V_{23} = \{B\}$

- **Finding x_{33} and V_{33} -**

$x_{33} = aba$

$V_{ij} = V_{ik}\ V_{(i+k)(j-k)}$

We have i = 3, j = 3, k = 1 to (3 – 1) = 1, 2

Substituting values in the formula, we get -

$V_{33} = V_{31} \cdot V_{42} \cup V_{32} \cdot V_{51}$

$V_{33} = \{A, C\}\ \{A, S\} \cup \{S, C\}\ \{A, C\}$

$V_{33} = \{AA, AS, CA, CS\} \cup \{SA, SC, CA, CC\}$

Since AA, AS, CA, CS, SA, SC and CA do not exist, so we have -

$V_{33} = \phi \cup \{CC\}$

$V_{33} = \phi \cup \{B\}$

$\therefore\ V_{33} = \{B\}$

The partially filled table is -

The x_{ij} table

baa	aab	aba		
ba	aa	ab	ba	
b	a	a	b	a

The V_{ij} table

ϕ	{B}	{B}		
{A, S}	{B}	{S, C}	{A, S}	
{B}	{A, C}	{A, C}	{B}	{A, C}

Step 4 :

- **Finding x_{14} and V_{14} -**

$x_{14} = baab$

Here i = 1, j = 4, k = j – 1 = 4 – 1 = 1, 2, 3

$\therefore\ V_{ij} = V_{ik}\ V_{(i+k)(j-k)}$

$V_{14} = V_{11}\ V_{23} \cup V_{12}\ V_{32} \cup V_{13}\ V_{41}$

$= \{B\}\ \{B\} \cup \{A, S\}\ \{S, C\} \cup \{\phi, B\}$

$= \{BB, AS, AC, SS, SC, B\}$

As none of the above production exist,

$\therefore\ V_{14} = \phi$

- **Finding x_{24} and V_{24} -**

$x_{24} = aaba$

Here i = 2, j = 4, k = j – 1 = 4 – 1 = 1, 2, 3

$V_{ij} = V_{ik}\ V_{(i+k)(j-k)}$

$V_{24} = V_{21}\ V_{33} \cup V_{22}\ V_{42} \cup V_{23}\ V_{51}$

$= \{A, C\}\ \{B\} \cup \{B\}\ \{A, S\} \cup \{B\}\ \{A, C\}$

$= \{AB, CB\} \cup \{BA, BS\} \cup \{BA, BC\}$

The CB can not be obtained

$= \{AB, BA, BS, BC\}$

$\therefore\ V_{24} = \{S, C, A\}$

Step 5 : Finding x_{15} and V_{15} -

x_{15} = baaba

Here i = 1, j = 5, k = 5 – 1 = 1, 2, 3, 4

$V_{ij} = V_{ik}\ V_{(i+k)(j-k)}$

$V_{15} = V_{11} \cdot V_{24} \cup V_{12} \cdot V_{33} \cup V_{13} \cdot V_{42} \cup V_{14} \cdot V_{51}$

$V_{15} = \{B\}\ \{S, C, A\} \cup \{A, S\}\ \{B\} \cup \{\phi\}\ \{A, S\} \cup \{\phi\}\ \{A, C\}$

$V_{15} = \{BS, BC, BA\} \cup \{AB, SB\} \cup \{A, S\} \cup \{A, C\}$

Since BS, SB, A, S and C do not exist, so we have -

$V_{15} = \{BC, BA\} \cup \{AB\} \cup \phi \cup \phi$

$V_{15} = \{S, A\} \cup \{S, C\} \cup \phi \cup \phi$

$\therefore\ V_{15} = \{S, A, C\}$

The table is

The x_{ij} table

baaba				
baab	aaba			
baa	aab	aba		
ba	aa	ab	ba	
b	a	a	b	a

The V_{ij} table

{S, A, C}				
ϕ	{S, C, A}			
ϕ	{B}	{B}		
{A, S}	{B}	{S, C}	{A, S}	
{B}	{A, C}	{A, C}	{B}	{A, C}

As we can see, x_{15} = baaba is a string and V_{15} = {S, A, C}

That means V_{15} contains start symbol S. Thus x_{15} = baaba is a member of given grammar. In other words, the given string baaba can be obtained by given grammar.

The value ϕ in V_{ij} table indicates that corresponding string in x_{ij} are not the members of given grammar.

Thus 'baab' and 'baa' can not be obtained from given grammar.

3.12 Applications of CFG

Q.49 Explain with suitable examples, any two applications of context free grammars. **☞[SPPU : Dec. 2015, End Sem, Oct.-19, Marks 4]**

Ans. : Various applications of context free grammar are

1. Parsing Technique : The context free grammar is used for parsing the programming constructs and for finding the syntactical errors from source language.

For example, if in your C program the statement is

x = y + z;

Then the x, y and z are identified as 'id' by lexical analyzer. It will be interpreted as

'id' = 'id' + 'id' ;

The CFG for this will be

$S \rightarrow id = ET$

$E \rightarrow E + F$

$E \rightarrow F$

$F \rightarrow id$

$T \rightarrow ;$

Let us build the parse tree accordingly.

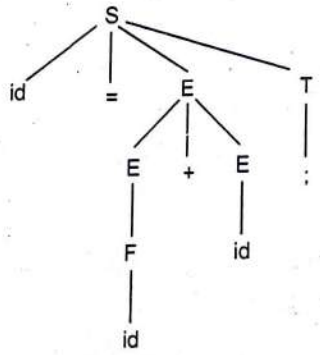


Fig. Q.49.1 Parse tree

Read out the leaf nodes and you will get id = id + id ; If ; is missing in your statement rule won't get matched with T branch and hence it will generate syntax error as 'statement ; missing'.

2. Markup Languages : Markup languages is a family of languages in which the certain strings have special meaning. These strings are referred as **tags**. Thus tags tell us about the semantics of various strings within the document. The most commonly used example of markup language is **Hyper Text Markup Language** (HTML). The context free language is used to describe the structure of HTML document. Consider following HTML document -

* Wonderful colors *

<h1>Red</h1>

<h2>Blue</h2>

<h3>Green</h3>

3. XML and Document Type Definition(DTD) : Extensible Markup Language (XML) is a special kind of language using which user can define his own tags. Purpose of XML is to describe the semantics of the text. For example, consider following XML document

```
<Person>
  <Personal-Info>
    <Name>My Name is Jaya</Name>
    <City>I live in Pune</City>
  </Personal-Info>
  <Hobby>
    <first>I like reading</first>
    <second>I like programming</second>
    <third>I like singing</third>
  </Hobby>
</Person>
```

The **Document Type Definition (DTD)** is a kind of context free grammar used to define the structure of the XML document. The DTD has its own notations and variables.

In DTD, The basic entity is **ELEMENT**. The notations of context free grammar can also be used in DTD. For instance :

hobby → programming|reading|singing

can be denoted as :

<!ELEMENT hobby(programming | reading |singing)

END...

Unit IV

4 Pushdown Automata (PDA)

4.1 : Formal Definition of PDA

Q.1 Define - Pushdown automata.

Ans. : The PDA can be defined as a collection of seven components.

1. The finite set of states Q.
2. The input set Σ.
3. Γ is a stack alphabet.
4. q_0 is initial state, $q_0 \in Q$.
5. Z_0 is a start symbol which is in Γ.
6. Set of final states $F \subseteq Q$.
7. δ is mapping function used for moving from current state to next state.

Q.2 Design a PDA for accepting a language $\{L = a^n b^n \mid n \geq 1\}$.

☞ [SPPU : May-16, End Sem. Marks 7, Dec.-16, End Sem. Marks 6, May-17, Marks 8]

Ans. : This is a language in which equal number of a's are followed by equal number of b's. The logic for this PDA can be applied as : First we will push all a's onto the stack. Then on reading every single b each a is popped from the stack. If we read all b and remove all a's and if we get stack empty then that string will be accepted. The instantaneous description can be given as -

$$\delta(q_0, a, Z_0) = (q_0, aZ_0)$$
$$\delta(q_0, a, a) = (q_0, aa)$$
$$\delta(q_0, b, a) = (q_1, \varepsilon)$$
$$\delta(q_1, b, a) = (q_1, \varepsilon)$$
$$\delta(q_1, \varepsilon, Z_0) = (q_2, \varepsilon)$$

where q_0 is a start state and q_2 is accept state or final state

We will simulate this PDA for following string -

$(q_0, aaabbb, Z_0) \vdash (q_0, aabbb, aZ_0)$
$\vdash (q_0, abbb, aaZ_0)$
$\vdash (q_0, bbb, aaaZ_0)$
$\vdash (q_1, bb, aaZ_0)$
$\vdash (q_1, b, aZ_0)$
$\vdash (q_1, \varepsilon, Z_0)$
$\vdash (q_2, \varepsilon)$

ACCEPT state.

Q.3 Construct PDA for the language $L = \{a^n b^{2n} \mid n \geq 1\}$.

☞ [SPPU : May-11, Marks 7]

Ans. : In this language n number of *a*'s should be followed by 2n number of b's. Hence we will apply a very simple logic and that is if we read single '*a*' we will push two *a*'s onto the stack. As soon as we read 'b' then for every single 'b' only one '*a*' should get popped from the stack. This basically maintains the $a^n b^{2n}$ count and sequence. The ID can be constructed as follows -

$\delta(q_0, a, Z_0) = (q_0, aaZ_0)$
$\delta(q_0, a, a) = (q_0, aaa)$

Now when we read b we will change the state from q_0 to q_1 and start popping corresponding '*a*'. Hence,

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

Thus this process of popping will be repeated unless all the symbols are read. Note that popping action occurs in state q_1 only.

$\therefore \quad \delta(q_1, b, a) = (q_1, \varepsilon)$

After reading all b's all the corresponding *a*'s should get popped. Hence when we read ε as input symbol there should be nothing in the stack. Hence the move will be -

$\delta(q_1, \varepsilon, Z_0) = (q_2, \varepsilon)$

where the PDA $P = (\{q_0, q_1, q_2\}, \{a, b\}, \{a, Z_0\}, \delta, q_0, Z_0, \{q_2\})$

We can summarize the ID as

$\delta(q_0, a, Z_0) = (q_0, aaZ_0)$
$\delta(q_0, a, a) = (q_0, aaa)$

$\delta(q_0, b, a) = (q_1, \varepsilon)$
$\delta(q_1, b, a) = (q_1, \varepsilon)$
$\delta(q_1, \varepsilon, Z_0) = (q_2, \varepsilon)$

Let us simulate this PDA for some input string "*aaa*bbbbbb".

$\delta(q_0, aaabbbbbb, Z_0) \vdash (q_0, aabbbbbb, aaZ_0)$
$\vdash (q_0, abbbbbb, aaaaZ_0)$
$\vdash (q_0, bbbbbb, aaaaaaZ_0)$
$\vdash (q_1, bbbbb, aaaaaZ_0)$
$\vdash (q_1, bbbb, aaaaZ_0)$
$\vdash (q_1, bbb, aaaZ_0)$
$\vdash (q_1, bb, aaZ_0)$
$\vdash (q_1, b, aZ_0)$
$\vdash (q_1, \varepsilon, Z_0)$
$\vdash (q_2, \varepsilon)$

Final state or ACCEPT state.

Thus input gets accepted by using the constructed PDA.

Q.4 Design PDA to accept the language

$$L = \left\{ w / w \in (a + b)^* \text{ and } n_a(w) = n_b(w) \right\}.$$

Ans. : This is a language of equal number of *a*'s and equal number of b's. Initially when stack is empty then whatever we read either '*a*' or 'b' we will simply push it onto the stack. Now if we read '*a*' and at the top of the stack if 'b' is present then it will be erased by ε. (i.e. we pop it) Similarly if we read 'b' and top of the stack contains '*a*' then erase it by ε. The instantaneous description for this language is as given below.

$\delta(q_0, a, Z_0) = (q_0, aZ_0)$
$\delta(q_0, b, Z_0) = (q_0, bZ_0)$
$\delta(q_0, a, a) = (q_0, aa)$
$\delta(q_0, b, b) = (q_0, bb)$
$\delta(q_0, a, b) = (q_0, \varepsilon)$
$\delta(q_0, b, a) = (q_0, \varepsilon)$
$\delta(q_0, \varepsilon, Z_0) = (q_1, Z_0)$

where q_0 is a start state and q_1 is a final state. When we reach to this state with ε input and having an empty stack. We move to accept/final state. Let us simulate this for a string '*aabab*b'.

$$\begin{aligned}\delta(q_0, aabbabb, Z_0) &\vdash (q_0, abbabb, aZ_0)\\ &\vdash (q_0, babb, aaZ_0)\\ &\vdash (q_0, abb, aZ_0)\\ &\vdash (q_0, bb, aaZ_0)\\ &\vdash (q_0, b, aZ_0)\\ &\vdash (q_0, \varepsilon, Z_0)\\ &\vdash (q_1, Z_0)\end{aligned}$$

ACCEPT state.

Q.5 Design a PDA for the language

$L = \{w \mid w \in (a + b)^* \text{ and } n_a(w) > n_b(w)\}$.

Ans. : • **Logic :** $n_a(w)$ means total number of *a*'s in input string and $n_b(w)$ means total number of b's in input string. The problem states that total number of *a*'s are more than total number of b's in input string. The logic for this PDA will be -

If we read *a* or b we will simply push it onto the stack. If the stack top has a symbol *a* and we read *a*, then also push it onto the stack. Same is true for b. But if we read '*a*' and stack top symbol is 'b' then pop. Also if we read 'b' and stack top symbol is '*a*' then pop the stack. Finally if we read ε (i.e. Complete string is read) then stack should contain *a* on the top. This means that total number of *a*'s are more than total number of b's. The ID can be

• **Design :**

$$\begin{aligned}\delta(q_0, a, Z_0) &= (q_0, aZ_0)\\ \delta(q_0, b, Z_0) &= (q_0, bZ_0)\\ \delta(q_0, a, a) &= (q_0, aa)\\ \delta(q_0, b, b) &= (q_0, bb)\\ \delta(q_0, a, b) &= (q_0, \varepsilon)\\ \delta(q_0, b, a) &= (q_0, \varepsilon)\\ \delta(q_0, \varepsilon, a) &= (q_f, a)\end{aligned}$$

where $P = (\{q_0, q_f\}, \{a, b\}, \{a, b, Z_0\}, \delta, q_0, Z_0, \{q_f\})$

Simulation : We will take some string to simulate this PDA for some input string.

Consider the input *aababab*.

$$\begin{aligned}\delta(q_0, aababab, Z_0) &\vdash (q_0, ababab, aZ_0)\\ &\vdash (q_0, babab, aZ_0)\\ &\vdash (q_0, abab, aZ_0)\\ &\vdash (q_0, bab, aaZ_0)\\ &\vdash (q_0, ab, aZ_0)\\ &\vdash (q_0, b, aaZ_0)\\ &\vdash (q_0, \varepsilon, aZ_0)\\ &\vdash (q_f, a)\end{aligned}$$

Accept state.

Thus input gets accepted.

Q.6 Design PDA for the language that accepts strings with $n_a(w) < n_b(w)$ where $w \in (a + b)^*$.

Ans. : This problem is similar to previous problem. The only difference is that in this problem the language requires more number of b's than total number of *a*'s.

Hence the ID will be -

$$\begin{aligned}\delta(q_0, a, Z_0) &= (q_0, aZ_0)\\ \delta(q_0, b, Z_0) &= (q_0, bZ_0)\\ \delta(q_0, a, a) &= (q_0, aa)\\ \delta(q_0, b, b) &= (q_0, bb)\\ \delta(q_0, b, a) &= (q_0, \varepsilon)\\ \delta(q_0, a, b) &= (q_0, \varepsilon)\\ \delta(q_0, \varepsilon, b) &= (q_f, b) \leftarrow \text{The b's are more in number.}\end{aligned}$$

Now we will **simulate** this PDA for the input "*abbab*".

$$\begin{aligned}\delta(q_0, abbab, Z_0) &\vdash (q_0, bbab, aZ_0)\\ &\vdash (q_0, bab, Z_0)\\ &\vdash (q_0, ab, bZ_0)\\ &\vdash (q_0, b, Z_0)\\ &\vdash (q_0, \varepsilon, bZ_0)\\ &\vdash (q_f, b)\end{aligned}$$

Accept state.

where q_0 is a start state and q_1 is a final state. When we reach to this state with ε input and having an empty stack. We move to accept/final state. Let us simulate this for a string '*aababb*'.

$$\begin{aligned}\delta(q_0, aababb, Z_0) &\vdash (q_0, ababb, aZ_0)\\ &\vdash (q_0, babb, aaZ_0)\\ &\vdash (q_0, abb, aZ_0)\\ &\vdash (q_0, bb, aaZ_0)\\ &\vdash (q_0, b, aZ_0)\\ &\vdash (q_0, \varepsilon, Z_0)\\ &\vdash (q_1, Z_0)\end{aligned}$$

ACCEPT state.

Q.5 Design a PDA for the language

$L = \{w \mid w \in (a + b)^* \text{ and } n_a(w) > n_b(w)\}$.

Ans. : • **Logic :** $n_a(w)$ means total number of *a*'s in input string and $n_b(w)$ means total number of b's in input string. The problem states that total number of *a*'s are more than total number of b's in input string. The logic for this PDA will be -

If we read *a* or b we will simply push it onto the stack. If the stack top has a symbol *a* and we read *a*, then also push it onto the stack. Same is true for b. But if we read '*a*' and stack top symbol is 'b' then pop. Also if we read 'b' and stack top symbol is '*a*' then pop the stack. Finally if we read ε (i.e. Complete string is read) then stack should contain *a* on the top. This means that total number of *a*'s are more than total number of b's. The ID can be

• **Design :**

$$\begin{aligned}\delta(q_0, a, Z_0) &= (q_0, aZ_0)\\ \delta(q_0, b, Z_0) &= (q_0, bZ_0)\\ \delta(q_0, a, a) &= (q_0, aa)\\ \delta(q_0, b, b) &= (q_0, bb)\\ \delta(q_0, a, b) &= (q_0, \varepsilon)\\ \delta(q_0, b, a) &= (q_0, \varepsilon)\\ \delta(q_0, \varepsilon, a) &= (q_f, a)\end{aligned}$$

where $P = (\{q_0, q_f\}, \{a, b\}, \{a, b, Z_0\}, \delta, q_0, Z_0, \{q_f\})$

Simulation : We will take some string to simulate this PDA for some input string.

Consider the input *aababab*.

$$\begin{aligned}\delta(q_0, aababab, Z_0) &\vdash (q_0, ababab, aZ_0)\\ &\vdash (q_0, babab, aZ_0)\\ &\vdash (q_0, abab, aZ_0)\\ &\vdash (q_0, bab, aaZ_0)\\ &\vdash (q_0, ab, aZ_0)\\ &\vdash (q_0, b, aaZ_0)\\ &\vdash (q_0, \varepsilon, aZ_0)\\ &\vdash (q_f, a)\end{aligned}$$

Accept state.

Thus input gets accepted.

Q.6 Design PDA for the language that accepts strings with $n_a(w) < n_b(w)$ where $w \in (a + b)^*$.

Ans. : This problem is similar to previous problem. The only difference is that in this problem the language requires more number of b's than total number of *a*'s.

Hence the ID will be -

$$\begin{aligned}\delta(q_0, a, Z_0) &= (q_0, aZ_0)\\ \delta(q_0, b, Z_0) &= (q_0, bZ_0)\\ \delta(q_0, a, a) &= (q_0, aa)\\ \delta(q_0, b, b) &= (q_0, bb)\\ \delta(q_0, b, a) &= (q_0, \varepsilon)\\ \delta(q_0, a, b) &= (q_0, \varepsilon)\\ \delta(q_0, \varepsilon, b) &= (q_f, b) \leftarrow \text{The b's are more in number.}\end{aligned}$$

Now we will **simulate** this PDA for the input "*abbab*".

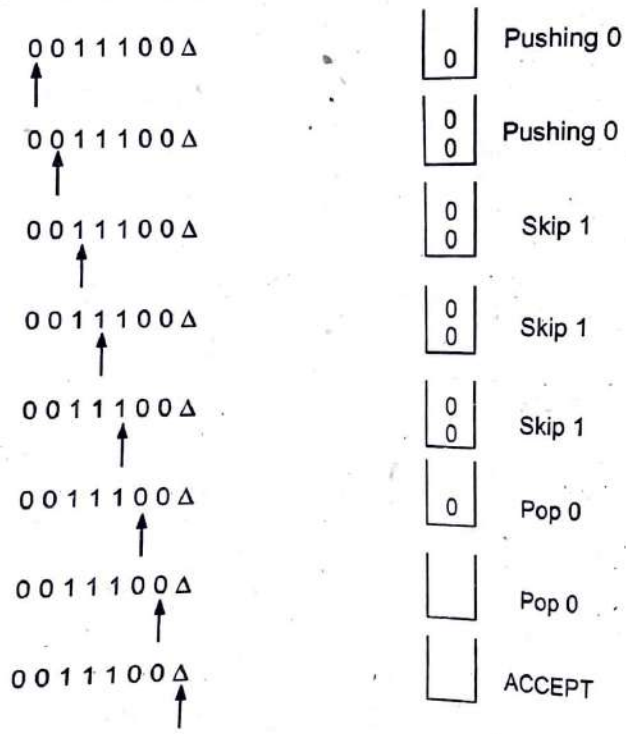
$$\begin{aligned}\delta(q_0, abbab, Z_0) &\vdash (q_0, bbab, aZ_0)\\ &\vdash (q_0, bab, Z_0)\\ &\vdash (q_0, ab, bZ_0)\\ &\vdash (q_0, b, Z_0)\\ &\vdash (q_0, \varepsilon, bZ_0)\\ &\vdash (q_f, b)\end{aligned}$$

Accept state.

Q.7 Build a PDA for the language $L = \{0^n\ 1^m\ 0^n \mid m, n \geq 1\}$ by empty stack. ☞ **[SPPU : Dec.-13, Marks 7; Dec.-19, Marks 6]**

Ans. : In this PDA n number of 0's are followed by any number of 1's followed by n number of 0's. Hence the logic for design of such PDA will be as follows. Push all 0's onto the stack on encountering first zeros. Then if we read 1, just do nothing. Then read 0 and on each read of 0, pop one 0 from the stack. For instance :

This scenario can be written in the ID form as



$$\delta(q_0, 0, Z_0) = \delta(q_0, 0Z_0)$$
$$\delta(q_0, 0, 0) = \delta(q_0, 00)$$
$$\delta(q_0, 1, 0) = \delta(q_1, 0)$$
$$\delta(q_0, 1, 0) = \delta(q_1, 0)$$
$$\delta(q_1, 0, 0) = \delta(q_1, \varepsilon)$$
$$\delta(q_0, \varepsilon, Z_0) = \delta(q_2, Z_0) \leftarrow \text{ACCEPT state}$$

For example :

$$\begin{aligned}\delta(q_0, 0011100, Z_0) &\vdash \delta(q_0, 011100, 0Z_0)\\ &\vdash \delta(q_0, 11100, 00Z_0)\\ &\vdash \delta(q_0, 1100, 00Z_0)\\ &\vdash \delta(q_1, 100, 00Z_0)\\ &\vdash \delta(q_1, 00, 00Z_0)\\ &\vdash \delta(q_1, 0, 0Z_0)\\ &\vdash \delta(q_1, \varepsilon, Z_0)\\ &\vdash \delta(q_2, Z_0)\\ &\text{ACCEPT}\end{aligned}$$

Q.8 Construct pushdown automata for each of the following language

1) The set of palindromes over alphabet {a, b}

2) The set of all string over alphabet {a, b} with exactly twice many a's as b's

3) $\{a^i b^j c^k \mid i \neq j \text{ or } j \neq k\}$

☞ **[SPPU : May-07, Marks 12, Dec.-16, End Sem., Marks 10]**

Ans. : **1)** For a given PDA

$$M = (\{q_0, q_1, q_2\}, \{a, b, c\}, \{a, b, Z_0\}\delta, q_0, Z_0, \{q_2\})$$

Mapping function δ is given as

$$\left.\begin{aligned} R_1 &: \delta(q_0, a, Z_0) = \delta(q_0, aZ_0)\\ R_2 &: \delta(q_0, b, Z_0) = \delta(q_0, bZ_0)\\ R_3 &: \delta(q_0, a, a) = \delta(q_0, aa)\\ R_4 &: \delta(q_0, b, a) = \delta(q_0, ba)\\ R_5 &: \delta(q_0, a, b) = \delta(q_0, ab)\\ R_6 &: \delta(q_0, b, a) = \delta(q_0, ba)\end{aligned}\right\}$$ Pushing the elements onto the stack

$$\left.\begin{aligned} R_7 &: \delta(q_0, \varepsilon, Z_0) = \delta(q_1, Z_0)\\ R_8 &: \delta(q_0, \varepsilon, a) = \delta(q_1, a)\end{aligned}\right\}$$ Reading middle a or b

$$\left.\begin{aligned} R_9 &: \delta(q_0, \varepsilon, b) = \delta(q_1, b)\\ R_{10} &: \delta(q_1, a, a) = \delta(q_1, \varepsilon)\\ R_{11} &: \delta(q_1, b, b) = \delta(q_1, \varepsilon)\end{aligned}\right\}$$ Popping the elements if matching

$$R_{12} : \delta(q_1, \varepsilon, Z_0) = \delta(q_2, Z_0)$$

We will draw the transition graph for given δ transitions

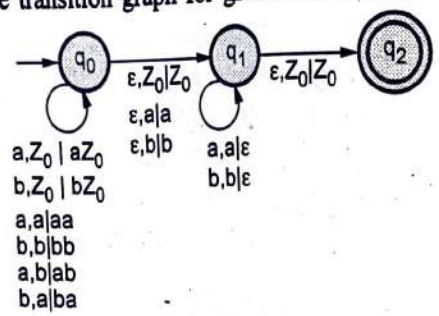


Fig. Q.8.1

We will simulate PDA for input

From initial ID, we will try to match rules R_1 to R_{12}

$(q_0, \underline{b}\ baabb, \underline{Z}_0)$

$\vdash \delta(q_0, \underline{b}aabb, \underline{b}Z_0)$

$\vdash \delta(q_0, \underline{a}\ abb, \underline{b}\ b\ Z_0)$

$\vdash \delta(q_0, abb, abb\ Z_0)$

$\vdash \delta(q_0, \underline{\varepsilon}abb, \underline{a}bb\ Z_0)$

$\vdash \delta(q_1, \underline{a}bb, \underline{a}bb\ Z_0)$

$\vdash \delta(q_1, \underline{b}\ b, \underline{b}\ b\ Z_0)$

$\vdash \delta(q_1, \underline{b}, \underline{b}Z_0)$

$\vdash \delta(q_1, \varepsilon, Z_0)$

$\vdash (q_1, Z_0)$

$\vdash$ ACCEPT

2) We will apply a simple logic to design this PDA if we read first 'a' then simply change the state from q_0 to q_1. In state q_1 if a is read then push 'A', and for this second 'a', change the state from q_1 to q_0 again.

If we read single b when we are there in state q_0 then push B onto the stack. This state indicates that we are waiting for two a's to read out. On receiving two a's, this B will be popped.

The simple rule in the design of this PDA is : State q_1 indicates that a single 'a' is read so far now. The δ transitions for this PDA are as given below -

1. $\delta(q_0, a, Z_0) = (q_1, Z_0)$

2. $\delta(q_1, a, Z_0) = (q_0, AZ_0)$ i.e. push A on reading second a and change the state to q_0
3. $\delta(q_0, b, Z_0) = (q_0, BZ_0)$ i.e. initially simply push B on reading single B.
4. $\delta(q_1, b, Z_0) = (q_1, BZ_0)$ i.e. push B on reading b
5. $\delta(q_0, a, A) = (q_1, A)$ i.e. simply read the a and change state to q_1.
6. $\delta(q_0, a, B) = (q_1, B)$ i.e. one b is already read and now one a is read. so keep B as it is
7. $\delta(q_0, b, B) = (q_0, BB)$ i.e. push B for reading a
8. $\delta(q_1, a, B) = (q_0, \varepsilon)$ i.e. read two a's for single b.
9. $\delta(q_1, a, A) = (q_0, AA)$ i.e. push A onto the stack.
10. $\delta(q_0, b, A) = (q_0, \varepsilon)$ pop A from stack.
11. $\delta(q_1, b, B) = (q_0, BB)$ push B for reading b.
12. $\delta(q_0, \varepsilon, Z_0) = (q_0, \varepsilon)$ **ACCEPT.**

Simulation for abbaaa

$\delta(q_0, \underline{a}\ b\ b\ a\ a\ a, \underline{Z}_0) = (q_1, \underline{a}\ b\ b\ a\ a\ a, Z_0)$ rule 1

$= (q_1, a\ \underline{b}\ b\ a\ a\ a, BZ_0)$ rule 4

$= (q_1, a\ b\ \underline{b}\ a\ a\ a, BZ_0)$ rule 11

$= (q_1, a\ b\ b\ \underline{a}\ a\ a, \underline{B}BZ_0)$ rule 8

$= (q_0, a\ b\ b\ a\ \underline{a}\ a, BZ_0)$ rule 6

$= (q_1, a\ b\ b\ b\ \underline{a}, \underline{B}Z_0)$ rule 8

$= (q_0, a\ b\ b\ a\ a\ a, Z_0)$ rule 12

ACCEPT.

3) Let, $L = \{a^i b^j c^k \mid i \neq j \text{ or } j \neq k\}$

Note that this is an or condition. If we consider the first condition i.e. $i \neq j$ then we have to focus on a's and b's only. At the same time any number of c's are allowed. Here $i > j$ or $j > i$ are the two applicable conditions.

Similarly in the second condition $j \neq k$ then we have to focus on b's and c's only. At the same time any number of a's are allowed.

Here $j > k$ or $k > j$ are the two applicable conditions.

We will first write the CFG for these conditions.

$S \rightarrow P|Q|R|T|M$

Let, A, B and C are the non terminals that generate a^+, b^+ and c^+ respectively.

$A \rightarrow a\ A|a$

$B \rightarrow b\ B|b$

$C \rightarrow c\ C|c$

i) P generates $a^i b^j c^k$ such that $i > j$

$P \rightarrow A\ D|A\ D\ C$

$D \rightarrow a\ D\ b|\varepsilon$ Here $D = a^n b^n$ where $n \geq 0$.

ii) Q generates $a^i b^j c^k$ such that $j > i$

$Q \rightarrow D\ B|D\ B\ C$

iii) R generates $a^i b^j c^k$ such that $j > k$

$R \rightarrow B\ E|A\ B\ E$

$E \rightarrow b\ E\ c|\varepsilon$ Here $E = b^n c^n$ where $n \geq 0$

iv) M generates $a^i b^j c^k$ such that $k > j$

$M \rightarrow E\ C|A\ E\ C$

From these above given production rules we can write δ for PDA as follows.

$(q, \varepsilon, S) = \{(q, P), (q, Q), (q, R), (q, T), (q, M)\}$

$(q, \varepsilon, A) = \{(q, aA), (q, a)\}$

$(q, \varepsilon, B) = \{(q, bB), (q, b)\}$

$(q, \varepsilon, C) = \{(q, cC), (q, c)\}$

$(q, \varepsilon, P) = \{(q, AD), (q, ADC)\}$

$(q, \varepsilon, D) = \{(q, aDb), (q, \varepsilon)\}$

$(q, \varepsilon, Q) = \{(q, DB), (q, DBC)\}$

$(q, \varepsilon, R) = \{(q, BE), (q, ABE)\}$

$(q, \varepsilon, E) = \{(q, bEc), (q, \varepsilon)\}$

$(q, \varepsilon, M) = \{(q, E, C), (q, AEC)\}$

$(q, a, a) = \{(q, \varepsilon)\}$

$(q, b, b) = \{(q, \varepsilon)\}$

$(q, c, c) = \{(q, \varepsilon)\}$

Q.9 Construct transition table for PDA that accepts the languages $L = \{a^{2n} b^n \mid n >= 1\}$. Trace your PDA for the input with n = 3.

☞ [SPPU : May-15, End Sem, Marks 10]

Ans. : The logic for building this PDA is as follows :

Just read one a and push and change state. Then read and push next a onto the stack. Repeat this for all strings of a's. On reading each b pop one a from stack.

The δ function can be as given below.

$\delta(q_0, a, Z_0) = (q_1, Z_0)$ Simply read a and change state.

$\delta(q_1, a, Z_0) = (q_0, AZ_0)$ Push a onto the stack as A.Change state to q_0.

$\delta(q_0, a, A) = (q_1, A)$ Simply read a and change state.

$\delta(q_1, a, A) = (q_1, AA)$ Push a as A

$\delta(q_0, b, A) = (q_0, \varepsilon)$ POP A

$\delta(q_0, \varepsilon, Z_0)$ ACCEPT.

Simulation for n = 3 i.e. aaaaaabbb

$(q_0, aaaaaabbb, Z_0) \vdash (q_1, aaaaabbb, Z_0)$

$\vdash (q_0, aaaa\ bbb, AZ_0)$

$\vdash (q_1, aaa\ bbb, AZ_0)$

$\vdash (q_0, aa\ bbb, AAZ_0)$

$\vdash (q_1, abbb, AAZ_0)$

$\vdash (q_0, bbb, AAAZ_0)$

$\vdash (q_0, bb, AAZ_0)$

$\vdash (q_0, b, AZ_0)$

$\vdash (q_0, \varepsilon, Z_0)$

$\vdash$ ACCEPT

Q.10 Construct PDA that accepts language generated by following CFG : $S \rightarrow SS \mid (S) \mid ( )$ **☞ [SPPU : Dec.-17, End Sem, Marks 6]**

Ans. : The δ for given CFG for constructing PDA is as follows.

$\delta(q, \varepsilon, S) = \{(q, SS), (q, (S)), (q, ( ))\}$

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

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

4.2 : Acceptance by Final State

Important Points to Remember

- The PDA accepts its input by consuming it and then enters in the final state. The formal definition is
- Let $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ be a PDA. Then the language L(P) is called the **language accepted by final state.**
- The language L(P) can be defined as

$L(P) = \{w \mid (q_0, w, Z_0) \vdash^* (p, \varepsilon, \alpha)\}$

4.3 : Acceptance by an Empty stack

Important Points to Remember

- On reading the input string from initial configuration for some PDA, the stack of PDA gets empty. The formal definition is -
- Let $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ be the PDA then the **language accepted by empty stack N(P)** is defined as

$N(P) = \{w \mid (q_0, w, Z_0) \vdash^* (p, \varepsilon, \varepsilon)\}$

4.4 : Equivalence of Acceptance by Final State an Empty Stack

Q.11 Explain the equivalence of PDA with acceptance by final state and empty stack.

[SPPU : Dec.-15, End Sem, Marks 6; Dec.-19, Marks 4]

Ans. :

1. If $L = N(P_1)$ for some PDA P_1 then there is a PDA P_2 such that $L = L(P_2)$. That means the language accepted by empty stack PDA will also be accepted by final state PDA.
2. If there is a language $L = L(P_1)$ for some PDA P_1 then there is a PDA P_2 such that $L = N(P_2)$. That means language accepted by final state PDA is also acceptable by empty stack PDA.

Q.12 Let $L = \{a^j b^j c^k / i, j, k \geq$ and $i + j = k\}$

a) Find a PDA (which accepts via final state) that recognizes L

b) Find a PDA (which accepts via empty stack) that recognizes L.

[SPPU : Dec.-09, Marks 18, Dec.-15, End Sem, Marks 10]

Ans. : Let, $L = \{a^i b^j c^k \mid i+j = k\}$ be a given language. The simple logic for designing this PDA will be - just read a's and push A for each corresponding a. Then read b's and push A for each corresponding b. Now on reading each c just pop a single A from stack.

a) Acceptance via final state.

$\delta(q_0, a, Z_0) = (q_1, AZ_0)$

$\delta(q_1, a, A) = (q_1, AA)$

$\delta(q_0, b, Z_0) = (q_2, AZ_0)$ — no 'a' is present b^j c^k is a string.

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

$\delta(q_2, b, A) = (q_2, AA)$

} Pushing A onto the stack

$\delta(q_2, c, A) = (q_3, \varepsilon)$

$\delta(q_3, c, A) = (q_3, \varepsilon)$

} popping A from the stack.

$\delta(q_3, \varepsilon, Z_0) = (q_3, \varepsilon) \rightarrow$ Acceptance by final state q_3.

b) Acceptance via empty stack.

$\delta(q_0, a, Z_0) = (q_1, A\,Z_0)$

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

$\delta(q_0, b, Z_0) = (q_2, A\,Z_0)$

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

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

$\delta(q_2, c, A) = (q_3, \varepsilon)$

$\delta(q_3, c, A) = (q_3, \varepsilon)$

$\delta(q_3, \varepsilon, Z_0) = (q_0, \varepsilon) \rightarrow$ Acceptance by empty stack.

4.5 : Non-deterministic PDA (NPDA)

Q.13 Construct NPDA that accepts the language generated by
S = S + S | S * S | 4. ☞ **[SPPU : Dec.-17, End Sem, Marks 6]**

Solution : The PDA can be,

$$A = \{\{q\}, \{*, 4+\}, \{S, 4, +, *\}, \delta, q, S, \phi\}$$

The δ can be

R1 : $\delta(q,\varepsilon,S) = \{(q, S+S), (q, S*S\}, (q, 4)\}$

R2 : $\delta(q,\varepsilon,4) = \{(q,\varepsilon)\}$

R3 : $\delta(q,\varepsilon,+) = \{(q,\varepsilon)\}$

R4 : $\delta(q,\varepsilon,*) = \{(q,\varepsilon)\}$

Thus the generated PDA is NPDA because there are more than one moves for same input symbol and from same current state.

4.6 : PDA and Context Free Language

4.6.1 : PDA to CFG

Important Points to Remember

Algorithm for getting production rules of CFG

1. If q_0 is start state in PDA and q_n is final state of PDA then $[q_0\ Z\ q_n]$ becomes start state of CFG. Here Z represents the stack symbol.
2. The production rule for the ID of the form $\delta(q_i, a, Z_0) = (q_{i+1}, Z_1 Z_2)$ can be obtained as
   $\delta(q_i\ Z_0\ q_{i+k}) \rightarrow a\ (q_i\ Z_1\ q_m)\ (q_m\ Z_2\ q_{i+k})$
   where q_i, q_m represents the intermediate states, Z_0, Z_1, Z_2 are stack symbols and *a* is input symbol.
3. The production rule for the ID of the form
   $\delta(q_i, a, Z_0) = (q_{i+1}, \varepsilon)$ can be converted as $(q_i Z_0 q_{i+1}) \rightarrow a$

Q.14 Consider the PDA with following moves,
$\delta(q_0, a, Z_0) = \{(q_0, a, Z_0)\}$
$\delta(q_0, a, a) = \{(q_0, a\ a)\}$
$\delta(q_0, b, a) = \{(q_1\ \varepsilon)\}$
$\delta(q_1, b, a) = \{(q_1, \varepsilon)\}$
$\delta(q_0, \varepsilon, Z_0) = \{(q_1, \varepsilon)\}$
Obtain CFG equivalent to PDA.

☞ **[SPPU : Dec.-08, Marks 8, May-14, Marks 12, May-17, End Sem, Marks 8]**

Ans. : 1) We will create production rules for start symbol, S. These are as follow :

$$S \rightarrow (q_0, Z_0, q_0)$$
$$S \rightarrow (q_0, Z_0, q_1)$$

2) Consider the rule $\delta(q_0, a, Z_0) = (q_0, aZ_0)$

i) $(q_0\ Z_0\ q_0) \rightarrow a\ (q_0\ a\ q_0)(q_0\ Z_0\ q_0)$
ii) $(q_0\ Z_0\ q_0) \rightarrow a\ (q_0\ a\ q_1)(q_1\ Z_0\ q_0)$
iii) $(q_0\ Z_0\ q_1) \rightarrow a\ (q_0\ a\ q_0)(q_0\ Z_0\ q_1)$
iv) $(q_0\ Z_0\ q_1) \rightarrow a\ (q_0\ a\ q_1)(q_1\ Z_0\ q_1)$

3) Consider the rule $\delta(q_0, a, a) = (q_0\ a\ a)$

v) $(q_0\ a\ q_0) \rightarrow a\ (q_0\ a\ q_0)(q_0\ a\ q_0)$
vi) $(q_0\ a\ q_0) \rightarrow a\ (q_0\ a\ q_1)(q_1\ a\ q_0)$
vii) $(q_0\ a\ q_1) \rightarrow a\ (q_0\ a\ q_0)(q_0\ a\ q_1)$
viii) $(q_0\ a\ q_1) \rightarrow a\ (q_0\ a\ q_1)\ (q_1\ a\ q_1)$

4) Consider the rule $(q_0, b, a) = (q_1, \varepsilon)$

ix) $(q_0\ a\ q_1) \rightarrow b$

5) Consider the rule $(q_1, b, a) = (q_1, \varepsilon)$

x) $(q_1\ a\ q_1) \rightarrow b$

6) Consider the rule $(q_0, \varepsilon, Z_0) = (q_1, \varepsilon)$

xi) $(q_0\ Z_0\ q_1) \rightarrow \varepsilon$

If we assume the state names as follows -

$(q_0 Z_0\ q_0) = A$
$(q_0 Z_0\ q_1) = B$
$(q_0\ a\ q_0) = C$
$(q_1\ a\ q_1) = D$

Above obtained — Conversion rules →

$S \rightarrow A \mid B$
$A \rightarrow a\ CA \mid a\ DE$
$B \rightarrow a\ CB \mid a\ DF$
$C \rightarrow a\ CC \mid a\ DG$

$(q_1 Z_0 q_0) = E$

$(q_1 Z_0 q_1) = F$

$(q_1 a q_0) = G$

$(q_1 a q_1) = H$

$D \rightarrow a\,CD \mid a\,DH$

$D \rightarrow b$

$H \rightarrow b$

$B \rightarrow \varepsilon$

The rules for E, F, G are missing. Hence we will eliminate production rules on RHS containing E, F or G non terminals. Finally the production rules for the equivalent grammar are

$S \rightarrow A \mid B$

$A \rightarrow aCA$

$B \rightarrow aCB \mid \varepsilon$

$C \rightarrow aCC$

$D \rightarrow aCD \mid aDH \mid b$

$H \rightarrow b$

Q.15 Obtain CFG for the PDA given below

$\delta(q_0, 1, Z_0) = \{q_0, XZ_0)$

$\delta(q_0, 1, X) = (q_0, XX)$

$\delta(q_0, 0, X) = (q_1, X)$

$\delta(q_0, \varepsilon, Z_0) = (q_0, \varepsilon)$

$\delta(q_1, 1, X) = (q_1, \varepsilon)$

$\delta(q_0, 1, Z_0) = (q_0, Z_0)$. ☞[SPPU : May-15, 16, End Sem, Marks 9]

Ans. : 1) We will construct production rules for start symbol S.

$S \rightarrow (q_0 Z_0 q_0)$

$S \rightarrow (q_0 Z_0 q_1)$

2) Consider the rule $\delta(q_0, 1, Z_0) = (q_0, XZ_0)$

i) $(q_0 Z_0 q_0) \rightarrow 1(q_0 X q_0)(q_0 Z_0 q_0)$

ii) $(q_0 Z_0 q_0) \rightarrow 1(q_0, X, q_0)(q_1, Z_0 q_0)$

iii) $(q_0 Z_0 q_1) \rightarrow 1(q_0 X q_0)(q_0 Z_0 q_1)$

iv) $(q_0 Z_0 q_1) \rightarrow 1(q_0 X q_1)(q_1 Z_0 q_1)$

3) Consider the rule $\delta(q_0, 1, X) = (q_0 X X)$

v) $(q_0 X q_0) \rightarrow 1 (q_0 X q_0)(q_0 X q_0)$

vi) $(q_0 X q_0) \rightarrow 1 (q_0 X q_1)(q_1 X q_0)$

vii) $(q_0 X q_1) \rightarrow 1 (q_0 X q_0)(q_0 X q_1)$

viii) $(q_0 X q_1) \rightarrow 1 (q_0 X q_1)(q_1 X q_1)$

4) Consider the rule $\delta(q_0, 0, X) = (q_1, X)$

ix) $(q_0 X q_0) \rightarrow 0(q_0 X q_0)$

x) $(q_0 X q_1) \rightarrow 0(q_0 X q_1)$

5) Consider the rule $\delta(q_0, \varepsilon, Z_0) = (q_0, \varepsilon)$

xi) $(q_0 Z_0 q_0) \rightarrow \varepsilon$

6) Consider the rule $\delta(q_1, 1, X) = (q_1, \varepsilon)$

xii) $(q_1 X q_1) \rightarrow 1$

7) Consider the rule $\delta(q_0, 1, Z_0) = (q_0, Z_0)$

xiii) $(q_0 Z_0 q_0) \rightarrow 1(q_0 Z_0 q_0)$

xiv) $(q_0 Z_0 q_1) \rightarrow 1 (q_0 Z_0 q_1)$

If we assume following state names for above generated (i to xiv) production rules then

$(q_0 Z_0 q_0) = A$

$(q_0 Z_0 q_1) = B$

$(q_0 X q_0) = C$

$(q_0 X q_1) = D$

$(q_1 Z_0 q_0) = E$

$(q_1 Z_0 q_1) = F$

$(q_1 X q_0) = G$

$(q_1 X q_1) = H$

CFG rules

$S \rightarrow A \mid B$

$A \rightarrow 1\,CA \mid 1\,CE$

$B \rightarrow 1\,CB \mid 1\,DF$

$C \rightarrow 1\,CC \mid 1\,DG$

$D \rightarrow 1\,CD \mid 1\,DH$

$C \rightarrow 0C$

$D \rightarrow 0D$

$A \rightarrow \varepsilon$

$H \rightarrow 1$

$A \rightarrow 1\,A$

$B \rightarrow 1\,B$

The production rules for E, F and G are not defined. Hence we will eliminate the production rules on RHS containing the non terminals E, F and G. The resultant CFG rules will be

$S \rightarrow A \mid B$

$A \rightarrow 1\,CA \mid 1A \mid \varepsilon$

$B \rightarrow 1\ CB \mid 1\ B$

$C \rightarrow 1\ CC \mid 0\ C$

$D \rightarrow 1\ CD \mid 1\ DH \mid 0D$

$H \rightarrow 1$

The rules for B, C and D does not lead to any terminal symbols (i.e. the rules generate continuous non terminals). Hence we will eliminate them. Finally we will get -

$S \rightarrow A \quad \Rightarrow S \rightarrow 1\ S \mid \varepsilon$

$A \rightarrow 1\ A \mid \varepsilon$

Q.16 Find the grammar G equivalent to the following PDA.
$M = (\{q_0, q_1\}, \{0,1\}, \{X, Z_0\}, q_0, Z_0, \delta, \phi)$ and δ is
i) $\delta(q_0, 0, Z_0) = (q_0, XZ_0)$ **ii) $\delta(q_0, 0, X) = (q_0, XX)$**
iii) $\delta(q_0, 1, X) = (q_1, \Delta)$ **iv) $\delta(q_1, 1, X) = (q_1, \Delta)$**
v) $\delta(q_1, \Delta, X) = (q_1, \Delta)$ **vi) $\delta(q_1, \Delta, Z_0) = (q_1, \Delta)$**

☞ **[SPPU : Dec.-14, End Sem, Marks 9]**

Ans. : The corresponding grammar G can be constructed as follows

1) The production rule for start symbol S is

$S \rightarrow (q_0\ Z_0\ q_0)$

$S \rightarrow (q_0\ Z_0\ q_1)$

2) Consider the rule $\delta(q_0, 0, Z_0) = (q_0, X Z_0)$

i) $(q_0\ Z_0\ q_0) \rightarrow 0\ (q_0 X q_0)\ (q_0\ Z_0\ q_0)$

ii) $(q_0\ Z_0\ q_0) \rightarrow 0\ (q_0 X q_1)\ (q_1\ Z_0\ q_0)$

iii) $(q_0\ Z_0\ q_1) \rightarrow 0\ (q_0 X q_0)\ (q_0\ Z_0\ q_1)$

iv) $(q_0\ Z_0\ q_1) \rightarrow 0\ (q_0 X q_1)\ (q_1\ Z_0\ q_1)$

3) Consider the rule $\delta(q_0, 0, X) = (q_0, X X)$

v) $(q_0\ X\ q_0) \rightarrow 0\ (q_0 X q_0)\ (q_0\ X\ q_0)$

vi) $(q_0\ X\ q_0) \rightarrow 0\ (q_0 X q_1)\ (q_1\ X\ q_0)$

vii) $(q_0\ X\ q_1) \rightarrow 0\ (q_0 X q_0)\ (q_0\ X\ q_1)$

viii) $(q_0\ X\ q_1) \rightarrow 0\ (q_0 X q_1)\ (q_1\ X\ q_1)$

4) Consider the rule $\delta(q_0, 1, X) = (q_1, \Delta)$

ix) $(q_0\ X\ q_1) \rightarrow 1$

5) Consider the rule $\delta(q_1, 1, X) = (q_1, \Delta)$

x) $(q_1, X\ q_1) \rightarrow 1$

6) Consider the rule $\delta(q_1, \Delta, X) = (q_1, \Delta)$

xi) $(q_1 X q_1) \rightarrow \varepsilon$

7) Consider the rule $\delta(q_1, \Delta, Z_0) = (q_1, \Delta)$

xii) $(q_1 Z_0 q_1) \rightarrow \varepsilon$

If we assume

$(q_1\ Z_0\ q_0) = A$

$(q_0\ Z_0\ q_1) = B$

$(q_0\ X\ q_0) = C$

$(q_0\ X\ q_1) = D$

$(q_1\ Z_0\ q_0) = E$

$(q_1\ X\ q_0) = F$

$(q_1\ Z_0\ q_1) = G$

$(q_1\ X\ q_1) = H$

$\xrightarrow[\text{CFG will be}]{\text{The}}$

$S \rightarrow A \mid B$

$A \rightarrow 0CA \mid 0DE$

$B \rightarrow 0CB \mid 0DG$

$C \rightarrow 0CC \mid 0DF$

$D \rightarrow 0CD \mid 0DH$

$D \rightarrow 1$

$H \rightarrow 1$

$H \rightarrow \varepsilon$

$G \rightarrow \varepsilon$

There is no rule for E and F. Hence

We will eliminate RHS rules containing E and F. Hence the final production rules are

$S \rightarrow A \mid B$

$A \rightarrow 0CA$

$B \rightarrow 0CB \mid 0DG$

$C \rightarrow 0CC$

$D \rightarrow 0CD \mid 0DH \mid 1$

$H \rightarrow 1 \mid \varepsilon$

$G \rightarrow \varepsilon$

The rule for A, C generate infinite sequence of non terminals. Hence we will eliminate these non terminals. Finally we will get simplified rules as

$S \rightarrow B$

$B \rightarrow 0DG$

$D \rightarrow 0DH \mid 1$

$H \rightarrow 1 \mid \varepsilon$

$G \rightarrow \varepsilon$

4.6.2 : CFG to PDA

Important Points to Remember

The rule that can be used to obtain PDA from CFG is

Rule 1 : For non terminal symbols, add following rule

$\delta(q, \varepsilon, A) = (q, \alpha)$

where the production rule is $A \rightarrow \alpha$.

Rule 2 : For each terminal symbols, add following rule -

$\delta(q, a, a) = (q, \varepsilon)$ for every terminal symbol.

Q.17 Construct PDA for the given CFG

$S \rightarrow 0BB$

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

Test whether 010^4 is acceptable by this PDA.

Ans. : Let PDA $A = \{\{q\}, \{0,1\}, \{S,B,0,1\}, \delta, q, S, \phi\}$

The production rules δ can be given as -

R1 : $\delta(q, \varepsilon, S) = \{(q, 0BB)\}$

R2 : $\delta(q, \varepsilon, B) = \{(q, 0S), (q, 1S), (q, 0)\}$

R3 : $\delta(q, 0, 0) = \{(q, \varepsilon)\}$

R4 : $\delta(q, 1, 1) = \{(q, \varepsilon)\}$

Testing 010^4 i.e. 010000 against PDA

$\delta(q, 010000, S) \vdash \delta(q, 010000, 0BB)$ ∵ R1

$\vdash \delta(q, 10000, BB)$ ∵ R3

$\vdash \delta(q, 10000, 1SB)$ ∵ R2

$\vdash \delta(q, 0000, SB)$ ∵ R4

$\vdash \delta(q, 0000, 0BBB)$ ∵ R1

$\vdash \delta(q, 000, BBB)$ ∵ R3

$\vdash \delta(q, 000, 0BB)$ ∵ R2

$\vdash \delta(q, 00, BB)$ ∵ R3

$\vdash \delta(q, 00, 0B)$ ∵ R2

$\vdash \delta(q, 0, B)$ ∵ R3

$\vdash \delta(q, 0, 0)$ ∵ R2

$\vdash \delta(q, \varepsilon)$ ∵ R3

ACCEPT.

Thus 010^4 is accepted by the PDA.

Q.18 Design a PDA for the following grammar : $S \rightarrow aSb \mid bSa \mid SS \mid \wedge$ Is the resultant PDA deterministic or non-deterministic ? Justify your answer. **☞[SPPU : May-14, Marks 6]**

Ans. : The PDA can be

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

The δ can be -

R1 : $\delta(q, \varepsilon, S) = \{(q, aSb), (q, bSa), (q, SS), (q, \varepsilon)\}$

R2 : $\delta(q, a, a) = \{(q, \varepsilon)\}$

R3 : $\delta(q, b, b) = \{q, \varepsilon)\}$

The generated PDA is nondeterministic PDA because there can be more than one different moves for the same input symbol and from same current state.

Q.19 Construct PDA for the given grammar containing

i) $S \rightarrow BD$ ii) $D \rightarrow SC$ iii) $C \rightarrow AA$ iv) $S \rightarrow BC$

v) $B \rightarrow 0$ vi) $A \rightarrow 1$ **☞[SPPU : Dec.-14, End Sem, Marks 8]**

Ans. : We will convert the given grammar to Greibach Normal Form. Hence equivalent grammar will be -

$S \rightarrow 0D \mid 0C$

$D \rightarrow 0DC \mid 0CC$

$C \rightarrow 1A$

$B \rightarrow 0$

$A \rightarrow 1$

The PDA $A = \{\{q\}, \{0, 1\}, \{S, D, C, B, A, 0, 1\}, \delta, q, S, \phi\}$

The production rules δ can be given as -

$\delta(q, \varepsilon, S) = \{(q, 0D), (q, 0C)\}$

$\delta(q, \varepsilon, D) = \{(q, 0DC), (q, 0CC)\}$

$\delta(q, \varepsilon, C) = \{(q, 1A)\}$

$\delta(q, \varepsilon, B) = \{(q, 0)\}$

$\delta(q, \varepsilon, A) = \{(q, 1)\}$

$\delta(q, 0, 0) = \{(q, \varepsilon)\}$

$\delta(q, 1, 1) = \{(q, \varepsilon)\}$

Q.20 Construct PDA for the following regular grammar :
S → 0A | 1B | 0

A → A0 | B

B → c | d

☞ [SPPU : May-15, End Sem, Marks 8]

Ans. : We will convert given grammar to Greibach normal form.

Step 1 : S → 0A|1B|0 is already in GNF

Step 2 : A →A0 | B gives

A → A0 | c| d can be written as

A → 0A′

A′ → c A′| d A′|ε

Step 3 : B → c|d already in GNF

The production rules for δ for PDA can be as given below.

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

$\delta(q, \varepsilon, A) = \{(q,0A')\}$

$\delta(q, \varepsilon, A') = \{(q, cA')(q, dA'), (q, \varepsilon)\}$

$\delta(q, \varepsilon, B) = \{(q, c), (q, d)\}$

Q.21 Construct a PDA that accepts the language generated by grammer : i) S → 0S1|A, A → | A0 | S | ε
ii) S → aABB|aAA, A → aBB|a, B→bAA|A

☞ [SPPU : May-17 End Sem, Marks 8]

Ans. : Let, PDA A = {{q}, {0,1}, {S,A,0,1}, δ,q, S, ϕ}

The production rules δ can be given as

$R1: \delta(q, \varepsilon, S) = \{(q,0S1), (q,A)\}$

$R2: \delta(q, \varepsilon, A) = \{(q,1A0),(q,s),(q,\varepsilon)\}$

$R3: \delta(q,0,0) = \{(q,\varepsilon)\}$

$R3: \delta(q,1,1) = \{(q,\varepsilon)\}$

ii) Let, PDA A = {{q}, {a, b}, {S, A, B, a, b}, δ, q, S, ϕ}

The production rules δ can be as given below

$R1: \delta(q, \varepsilon, S) = \{(q,aABB), (q,aAA)\}$

$R2: \delta(q, \varepsilon, A) = \{q,aBB, (q,a)\}$

$R3: \delta(q, \varepsilon, B) = \{(q,bAA)\}, (q,A)\}$

4.7 : Deterministic CFL

Q.22 Explain the concept of deterministic CFG.

Ans. : • The Deterministic Context-Free Grammars (DCFGs) are a proper subset of the context-free grammars.

- They are the subset of context-free grammars that can be derived from deterministic pushdown automata and they generate the deterministic context-free languages.
- DCFGs are always unambiguous.

Q.23 Prove that if L is a regular language, then L = L (P) for some DPDA P.

Ans. : • Fig. Q.23.1 shows the class of various languages.

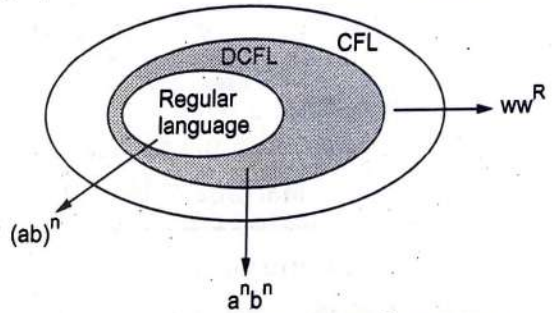


Fig. Q.23.1 Class of various languages

- A language L is a deterministic context free language (**DCFL**) if it is accepted by **DPDA**. Any language L is a subset of DCFL. For example $L = a^*b^*$ is a regular language and $L = a^nb^n$ is a DCFL, and it is accepted by DPDA. And therefore $L = a^*b^*$ is also acceptable by DPDA. Every regular language is accepted by DFA. And every DFA is a DPDA without stack. Hence it is proved that for a regular language L = L(P) there exists some DPDA P.

END...

Unit V

5 Turing Machines (TM)

5.1 : Formal Definition of Turing Machines

Q.1 Give the formal definition of Turing Machine.

☞ **[SPPU : Dec.-17, May-18, Marks 2]**

Ans. : The Turing machine is a collection of following components.

$$M = (Q, \Sigma, \Gamma, \delta, q_0, \Delta \text{ or } B, F)$$

1) Q is a finite set of states.

2) Γ is finite set of external symbols.

3) Σ is a finite set of input symbols.

4) Δ or b or $B \in \Gamma$ is a blank symbol majorly used as end marker for input.

5) δ is a transition or a mapping function.

5.2 : Turing Machine Model

Q.2 Explain the instantaneous description of Turing Machine.

OR Explain representation of TM.

☞ **[SPPU : Dec.-17, May-18, Marks 4]**

Ans. : Let M = (Q, Σ, Γ, δ, q_0, Δ or B, F) be TM then the Instantaneous Description (ID) of a TM can be given by a **triplet** or a **program** which is as given below -

$$(q_0, a) \rightarrow (q_1, A, L)$$

That means currently we are reading the 'input symbol '*a*' and we are in q_0 state then we can go to q_1 state by replacing or printing '*a*' by A and then we must move in **left** direction.

5.3 : Language Acceptability by Turing Machines

Q.3 Construct TM for the language $L = \{a^n b^n\}$ where $n \geq 1$.

☞ **[SPPU : May-13, 18, Marks 8]**

Ans. : The simple logic which we will apply is read out each *a* mark it by A and then move ahead along the input tape and find out the b convert it to B. Repeat this process for all *a*'s and b's. If there is some kind of unequality in number of *a*'s and b's the TM will not end up in HALT state. Let us formulize the logic for $a^n b^n$ and then construct it

a a b b Δ ↑	Convert it A and move right in search of b.
A *a* b b Δ ↑	Skip it, move ahead
A *a* b b Δ ↑	Convert it to B and move left till A
A *a* B b Δ ↑	Move left
A *a* B b Δ ↑	Move right
A *a* B b Δ ↑	Convert *a* to A and move right in search of b
A A B b Δ ↑	Move right
A A B b Δ ↑	Convert b to B and move left
A A B B Δ ↑	Move left
A A B B Δ ↑	Now immediately before B is A that means all the *a*'s are marked by A. So we will move right to ensure that no b is present
A A B B Δ ↑	Move right

A A B B Δ Move right
↑

A A B B Δ HALT
↑

Thus equality between *a*'s and b's can be checked by moving tape head to and fro. Let us try to design a TM using above logic.

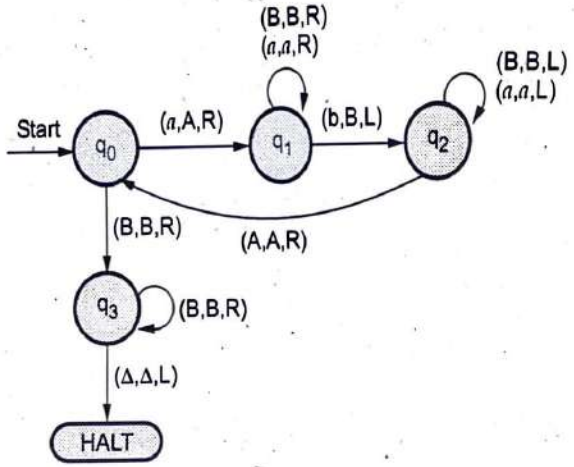


Fig. Q.3.1

Note that for the invalid inputs the machine does not go to HALT state.

For example

a a b b b Δ Convert to A, move right and next state = q_1
↑

A a b b b Δ Move right, keeping *a* as it is, state = q_1
↑

A a b b b Δ Convert to B, move to left
↑

A a B b b Δ Move to left, state = q_2
↑

A a B b b Δ Move to right and state = q_0
↑

A a B b b Δ Convert to A, move to right now in state q_1
↑

A A B b b Δ Move right keeping B as it is.
↑

A A B b b Δ Convert b to B and move left state is q_2
↑

A A B B b Δ Go on moving left
↑

A A B B b Δ Now state q_0 move right
↑

A A B B b Δ Move right skipping all B's
↑

A A B B b Δ From state q_3 there is no path for input b and TM stops there
↑

Let us design the transition table for the same.

	a	b	A	B	Δ
q_0	(q_1, A, R)	–	–	(q_3, B, R)	–
q_1	(q_1, *a*, R)	(q_2, B, L)	–	(q_1, B, R)	–
q_2	(q_2, *a*, L)	–	(q_0, A, R)	(q_2, B, L)	–
q_3	–	–	–	(q_3, B, R)	(HALT, Δ, L)
HALT	–	–	–	–	–

Table Q.3.1 Transition table

Thus TM does not lead to HALT state for such a invalid input.

Q.4 Construct a TM for $L = \{a^n b^n c^n \mid n \geq 1\}$.

Ans. : This problem cannot be solved even by PDA. We have solved this type of problem by two stack PDA.

The simulation for *a a* b b c c can be shown as below.

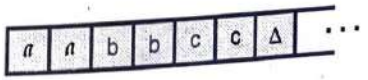


Fig. Q.4.1

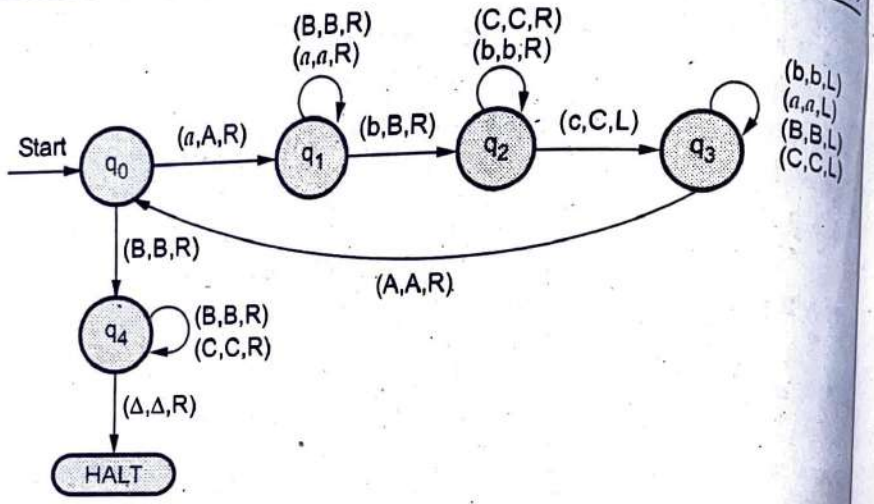


Fig. Q.4.2

a a b b c c Δ ↑	Convert *a* to A, move right the transition is from q_0 to q_1
A a b b c c Δ ↑	Move right upto b
A a b b c c Δ ↑	Convert b to B, move right, transition from q_1 to q_2
A a B b c c Δ ↑	Move right upto c
A a B b c c Δ ↑	Convert c to C, move right, transition from q_2 to q_3
A a B b C c Δ ↑	Move left upto C
A a B b C c Δ ↑	Move left till A, skipping *a*, b, B, C, see state q_3 then move one step right
A a B b C c Δ ↑	Convert *a* to A, move right

A a B b C c Δ ↑	Move right
A A B b C c Δ ↑	Convert b to B and move right, transition from q_1 to q_2
A A B B C c Δ ↑	Move right
A A B B C c Δ ↑	Convert c to C, move left till A, refer q_3 state
A A B B C C Δ ↑	Move right, keep B as it is refer transition from q_0 to q_4
A A B B C C Δ ↑	Go on moving right by ignoring B, C refer q_4 state
A A B B C C Δ ↑	Since Δ is read TM will reach to HALT state
A A B B C C Δ ↑	

Thus TM can very effectively recognize $a^n b^n c^n$ and more powerful than PDA.

Q.5 Construct a TM machine for checking the palindrome of the string of even length. **☞[SPPU : May-11, Marks 8]**

Ans. : The logic is we will read first symbol from left and then we compare it with the first symbol from right to check whether it is the same.

Again we compare second symbol from left with the second symbol from right. We repeat this process for all the symbols. If we found any symbol not matching, we cannot lead the machine to HALT state. The simulation of machine for a valid input such as

a b a b b a b a Δ ↑	We will mark it by * and move to right end in search of *a*
* b a b b a b a Δ ↑	Move right
* b a b b a b a Δ ↑	Moved right upto Δ
* b a b b a b a Δ ↑	Moved left, checked if it is *a*
* b a b b a b Δ Δ ↑	If it is *a*, replaced it by Δ
* b a b b a b Δ ↑	Move to left, upto *
* b a b b a b Δ ↑	Move right, read it
* b a b b a b Δ ↑	Convert it to * and move right
* * a b b a b Δ ↑	Move right upto Δ in search of b
* * a b b a b Δ ↑	Move left, if left symbol is b convert it to Δ
* * a b b a Δ ↑	Move left till *
* * a b b a Δ ↑	Goto right

* * a b b a Δ ↑	Replace *a* by * and move right, upto Δ
* * * b b a Δ ↑	We will move left and check if it is *a*, then replace it by Δ
* * * b b a Δ ↑	It is *a* so replace by Δ
* * * b b Δ ↑	Move to left till *
* * * b b Δ ↑	Move right
* * * b b Δ ↑	Replace it by *
* * * * b Δ ↑	Move right upto Δ
* * * * b Δ ↑	Move left to see if it is b, if so then replace it by Δ
* * * * Δ ↑	Move left till *
* * * * Δ ↑	Move right and check whether it is Δ, if so
* * * * Δ ↑	Goto HALT state

Let us draw TM for it,

If you observe the turing machine drawn for palindrome, you will find it very logically constructed. Even there is no need to insert any special symbol in between to separate out two halves.

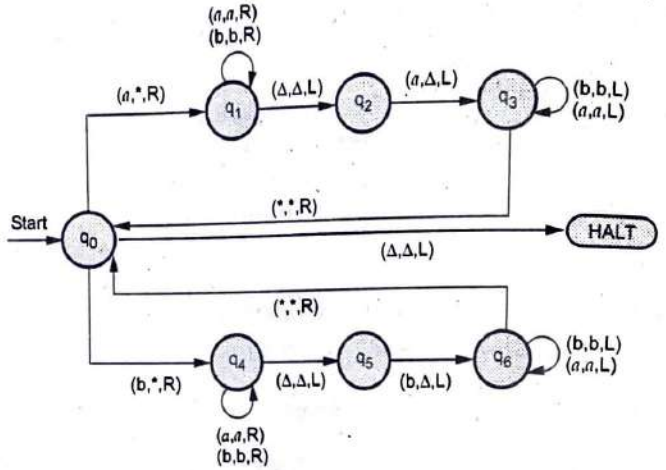


Fig. Q.5.1

Q.6 Construct a TM for checking the palindrome of a string of odd palindrome for $\Sigma = \{0, 1\}$. **☞[SPPU : May-14, Marks 7]**

Ans. : The TM will be very much similar to previous one but with a slight difference.

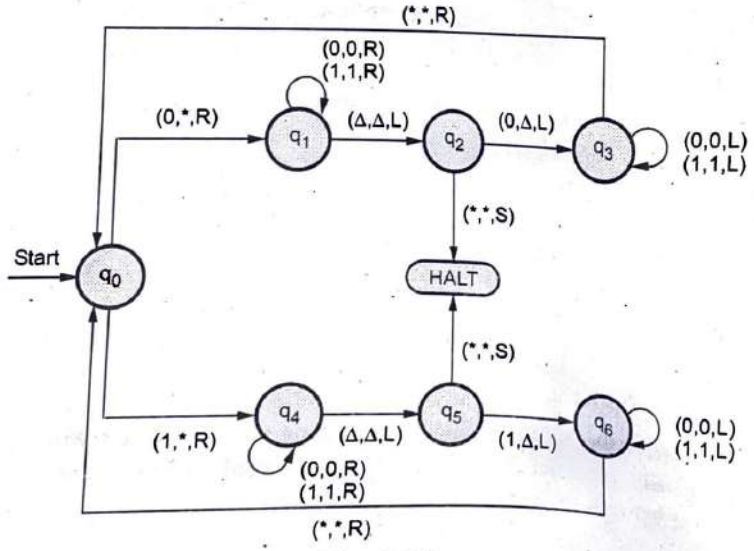


Fig. Q.6.1

For example : If the string is

0 0 1 0 0 Δ ↑	Move right by making 0 to *
* 0 1 0 0 Δ ↑	Move right upto Δ
* 0 1 0 0 Δ ↑	Move left, replace 0 by Δ
* 0 1 0 Δ Δ ↑	Move left
* 0 1 0 Δ ↑	Move left upto *
* 0 1 0 Δ ↑	Move right convert 0 to * and then move right
* * 1 0 Δ ↑	Move right upto Δ
* * 1 0 Δ ↑	Move left and replace 0 by Δ
* * 1 Δ Δ ↑	Move left till *
* * 1 Δ Δ ↑	Move right and convert 1 to *
* * * Δ Δ ↑	Move right

* * * Δ Δ — Move left

↑

* * * Δ Δ — Since it is *, goto HALT state

↑

Q.7 Construct turing machine that recognizes the language

1) $L = \{0^n 1^m : n, m >= 0\}$

2) $L = \{x \in \{0,1\}^* \mid x \text{ ends in } 00\}$ **☞[SPPU : May-06, Marks 10]**

Ans. : i) Let, $\{L = 0^n 1^m \mid n, m > = 0\}$

That means any number of 0's and 1's are allowed in a string but all the 0's must appear before all the 1's. Hence the TM can be

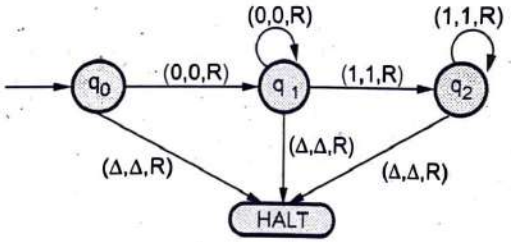


Fig. Q.7.1

ii) Let, $L = \{x \in \{0,1\}^* \mid x \text{ end in } 00\}$ be the given language. We will first draw FA for this language.

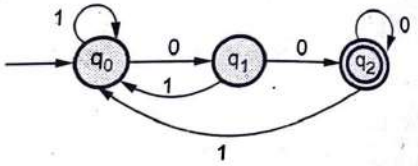


Fig. Q.7.2

The TM can then be as follows

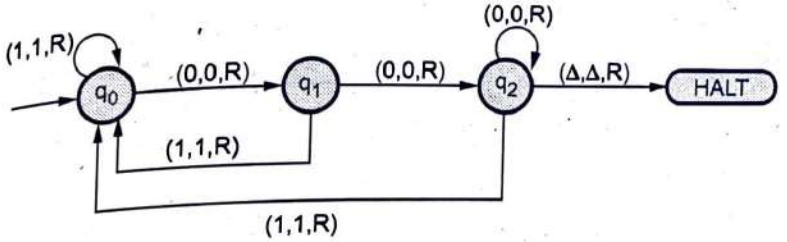


Fig. Q.7.3

Q.8 Design a turing machine to replace string 110 by 101 in binary input string.

☞[SPPU : Dec.-06, Marks 8, Dec.-14, End Sem., Marks 9]

Ans. : The TM can then be

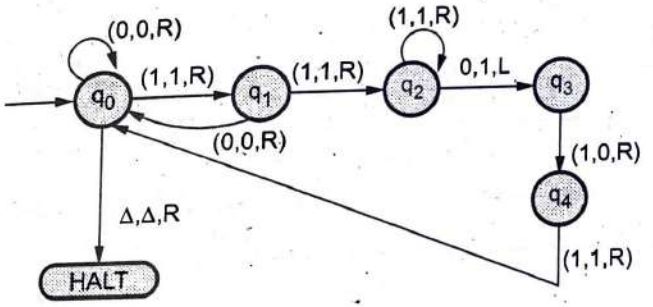


Fig. Q.8.1

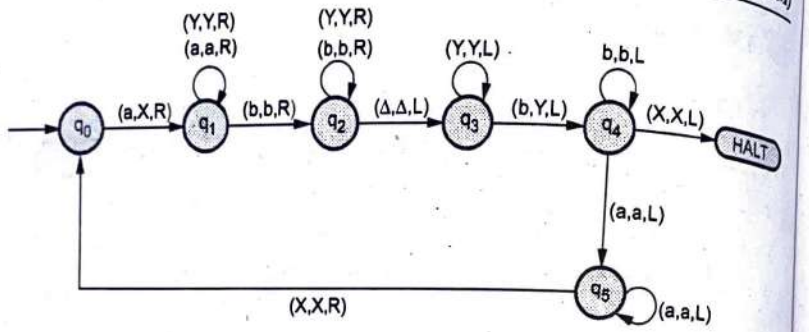
Q.9 Draw a transitions table for turing machine accepting each of the following languages.

a) $\{a^i b^j / i < j\}$

b) The language of balanced strings of parentheses.

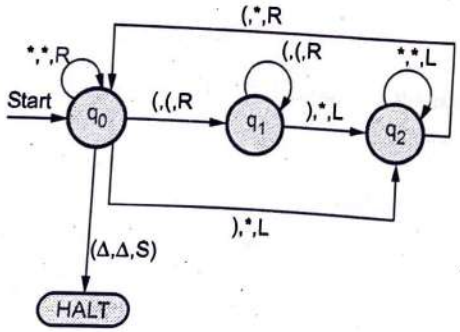
☞[SPPU : Dec.-07, 09, May-09, Marks 8, Dec.-14, End Sem., Marks 9]

Ans. : a) The required TM can be drawn as follows -



The transition table can be

Input / State	a	b	X	Y	Δ
q_0	(q_1, X, R)	–	–	–	–
q_1	(q_1, a, R)	(q_2, b, R)	–	–	–
q_2	–	(q_2, b, R)	–	(q_2, Y, R)	(q_3, Δ, L)
q_3	–	(q_4, Y, L)	–	(q_3, Y, L)	–
q_4	(q_5, a, L)	(q_4, b, L)	(HALT, X, L)	–	–
q_5	(q_5, a, L)	–	(q_0, X, R)	–	–



The transition table for this TM will be

State	(	)	*	Δ
q_0	$(q_1, (, R)$	$(q_2, *, L)$	$(q_0, *, R)$	(HALT, Δ, S)
q_1	$(q_1, (, R)$	$(q_2, *, L)$	–	–
q_2	$(q_2, *, R)$	–	$(q_2, *, L)$	–
HALT	–	–	–	–

Q.10 Design a turing machine M to recognize the language $\{1^n 2^n 3^n \mid n \geq 1\}$. **☞[SPPU : Dec.-11, Marks 8; Dec.-13, Marks 6]**

Ans. :

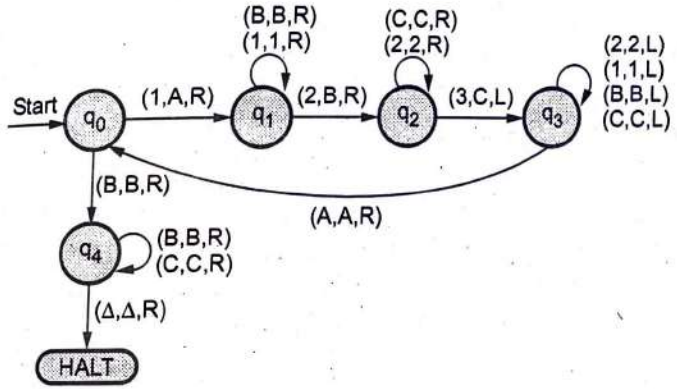


Fig. Q.10.1

Q.11 Construct a TM for a language having equal number of a's and b's in it over the input set $\Sigma = \{a, b\}$.

☞[SPPU : Dec.-08, Marks 6; May-12, Marks 8, Dec.-17,18, Marks 8; May-14, Marks 7]

Ans. : The input can be placed as follows.

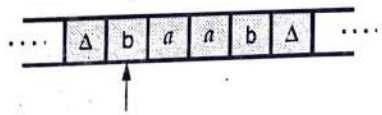


Fig. Q.11.1

We will have a simple logic as if we get *a* we will replace it by A and move right in search of b, if we get b we will replace it by B and move left. We will move to the leftmost end (i.e. Δ) and again repeat the same

process of marking *a* by A and b by B. We may have string starting with *a* or b, so from initial state only we will have two separate paths shown for *a* and b. Remember we have to check both the ends, to check whether the string is completely scanned or not. Let us understand the logic first and then move on for TM.

Δ b *a* *a* b Δ — Convert b to B and move right in search of *a*

Δ B *a* *a* b Δ — *a* has to be converted to A and move left

Δ B A *a* b Δ — Skip B and move left

Δ B A *a* b Δ — Keep Δ as it is and move right

Δ B A *a* b Δ — Move right

Δ B A *a* b Δ — Move right

Δ B A *a* b Δ — Convert *a* to A and move right

Δ B A A b Δ — Convert b to B move left

Δ B A A B Δ — Move left

Δ B A A B Δ — Move left

Δ B A A B Δ — Move left

Δ B A A B Δ — If Δ comes move to right by ignoring all A's and B's

Δ B A A B Δ — Move right

Δ B A A B Δ — Move right

Δ B A A B Δ — Move right

Δ B A A B Δ — Move right

Δ B A A B Δ — Δ is reached, goto HALT state

The TM can be

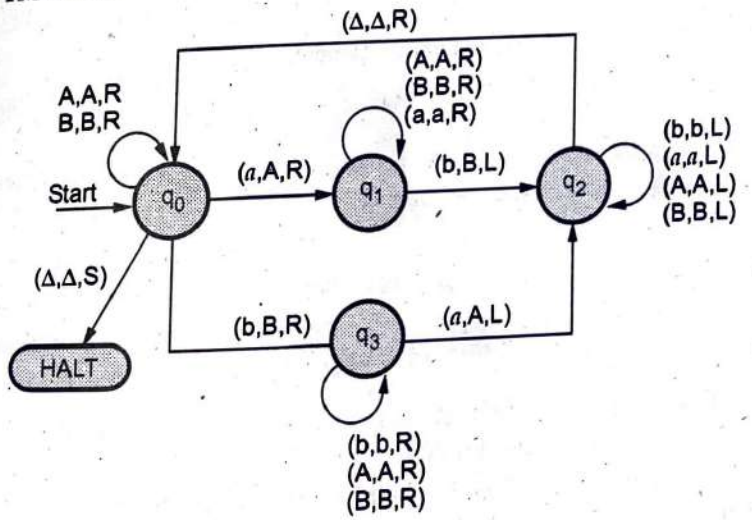


Fig. Q.11.2

Similarly if the string is *abab*

Δ *a* b *a* b Δ — Convert *a* to A move right

Δ A b *a* b Δ — Convert b to B move left

Δ A B *a* b Δ ↑	Skip A and move left
Δ A B *a* b Δ ↑	Move right, ignore all A's and B's
Δ A B *a* b Δ ↑	Convert *a* to A, move right
Δ A B A b Δ ↑	Convert b to B, move left
Δ A B A B Δ ↑	Skip all A's and B's and move left
Δ A B A B Δ ↑	Move right by skipping all A's and B's
Δ A B A B Δ ↑	Since both the ends are checked and all the *a*'s and b's are encountered as equal. So TM HALTs here.

Q.12 Construct TM for obtaining two's complement of a given binary number.

☞[SPPU : Dec.-05, Marks 10; Dec.-10, Marks 8, Dec.-19, Marks 6]

Ans. : The logic for computing two's complement is, we read the binary string from LSB to MSB. From LSB, we keep all the zero's as it is and move left till we do not get 1. After reading first 1 from LSB, we move left and thereafter we convert 0 to 1 and 1 to 0 and go on moving towards left. The process continues upto leftmost Δ.

For example : The binary number 0110 its two's complement can be computed as

0 1 1 0	
1 0 0 1	1's complement
+ 1	Adding 1
= 1 0 1 0	Two's complement of 0110

By our idea also we will get the same result.

Δ 0 1 1 0 Δ ↑	Move to rightmost end upto Δ
Δ 0 1 1 0 Δ ↑	Move left
Δ 0 1 1 0 Δ ↑	It is 0, so keep it as it is and move left
Δ 0 1 1 0 Δ ↑	First 1 from LSB has encountered so keep it as it is and move left
Δ 0 1 1 0 Δ ↑	Complement it to 0 and move left
Δ 0 0 1 0 Δ ↑	Convert it to 1 and move left
Δ 1 0 1 0 Δ ↑	Since Δ is reached, stop !

The TM could be

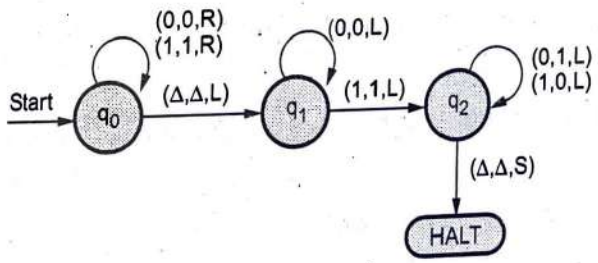


Fig. Q.12.1

Thus two way infinite tape brings significant power to turing machine.

Q.13 Construct TM for reversing a binary string on the input tape.

☞[SPPU : Dec.-09, May-10, Marks 10]

Ans. : Suppose on the input tape some binary string is placed, then we will read each character from left to right and copy it at the end of the tape.

For example

... Δ 1 0 1 0 0 1 Δ Δ Δ ... ↑	We will move upto the end of the input string
Δ 1 0 1 0 0 1 Δ Δ ... ↑	Move left
Δ 1 0 1 0 0 1 Δ Δ ... ↑	Mark it * and copy it after Δ
Δ 1 0 1 0 0 * Δ 1 ↑	Move left upto *
Δ 1 0 1 0 0 * Δ 1 Δ ↑	Move left, convert 0 to * and place it, at the end
Δ 1 0 1 0 * * Δ 1 0 Δ ↑	Move left upto *
Δ 1 0 1 0 * * Δ 1 0 Δ ↑	Move left, convert 0 to * and place it at the end
Δ 1 0 1 * * * Δ 1 0 0 Δ ↑	Move left upto *
Δ 1 0 1 * * * Δ 1 0 0 Δ ↑	Move left, convert 1 to * and place it at the end
Δ 1 0 * * * * Δ 1 0 0 1 Δ ↑	Move left upto *
Δ 1 0 * * * * Δ 1 0 0 1 Δ ↑	Move left, convert 0 to * and place it at the end
Δ 1 * * * * * Δ 1 0 0 1 0 Δ ↑	Move left skipping *

Δ 1 * * * * * Δ 1 0 0 1 0 Δ ↑	Copy 1 at the end of tape by converting 1 to *
Δ * * * * * * Δ 1 0 0 1 0 1 Δ ↑	Move left skipping all *
Δ * * * * * * Δ 1 0 0 1 0 1 Δ ↑	Left to * is a Δ character so we goto HALT state

The TM will be

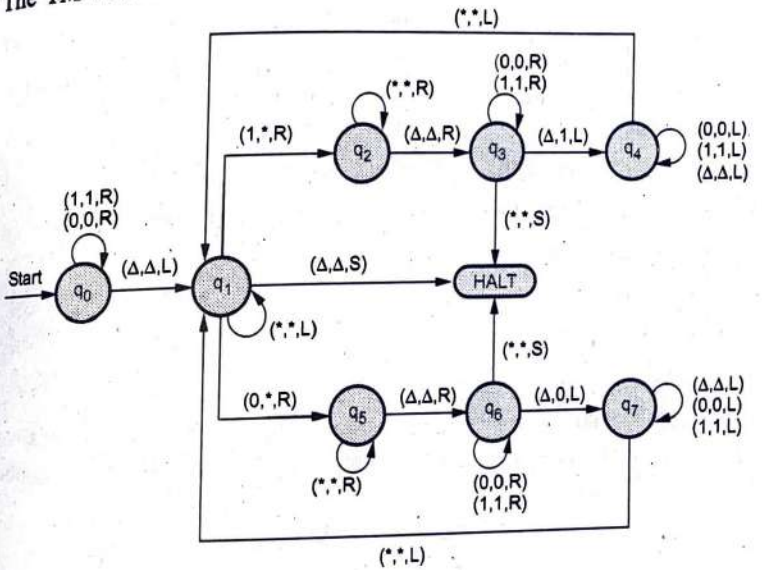


Fig. Q.13.1

The transition table for the above TM will be

	0	1	*	Δ
q_0	$(q_0, 0, R)$	$(q_0, 1, R)$	–	(q_2, Δ, L)
q_1	$(q_5, *, R)$	$(q_2, *, R)$	$(q_1, *, L)$	(HALT, Δ, S)
q_2	–	–	$(q_2, *, R)$	(q_3, Δ, R)
q_3	$(q_3, 0, R)$	$(q_3, 1, R)$	–	$(q_4, 1, L)$

q_4	$(q_4, 0, L)$	$(q_4, 1, L)$	$(q_1, *, L)$	(q_4, Δ, L)
q_5	–	–	$(q_5, *, R)$	(q_6, Δ, R)
q_6	$(q_6, 0, R)$	$(q_6, 1, R)$	–	$(q_7, 0, L)$
q_7	$(q_7, 0, L)$	$(q_7, 0, L)$	$(q_1, *, L)$	(q_7, Δ, L)
HALT	–	–	–	–

Thus TM reverses the input string.

Q.14 Design TM to recognize an arbitrary string divisible by 4 from $\Sigma = \{0, 1, 2\}$. **[SPPU : May-15, (End Sem), Marks 10]**

Ans. : As input string is $\Sigma = \{0, 1, 2\}$, any arbitrary string of input should be a ternary string and we have check whether such ternary input is divisible by 4 or not. For divisibility by 4 there are four case -

Remainder 0, Remainder 1, Remainder 2, Remainder 3

Now we will consider separate states for each of these cases -

q_0 - Remainder 0 state

q_1 - Remainder 1 state

q_2 - Remainder 2 state

q_3 - Remainder 3 state

HALT : is a final. When you get a valid string (string which is divisible by 4). The TM is using a triplet of (input, output, direction) during the transition from one state to another.

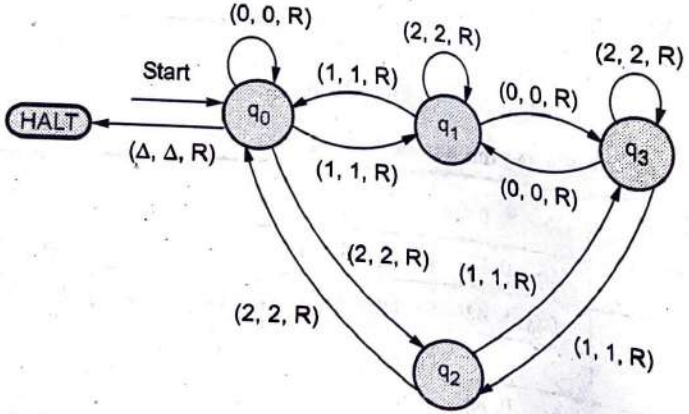


Fig. Q.14.1

1	2	1	Δ	.....

Input tape with sample input

For the sample input string 121Δ we will reach to halt state finally as $(121)_3 = (16)_{10}$ which is divisible by 4.

Q.15 Design a turing machine M to implement the function multiplication using the subroutine 'copy'.

Ans. : Initially the input is placed on the tape as follows.

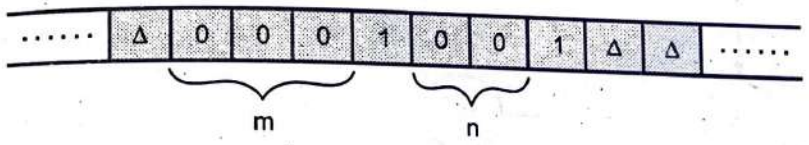


Fig. Q.15.1

The TM can be

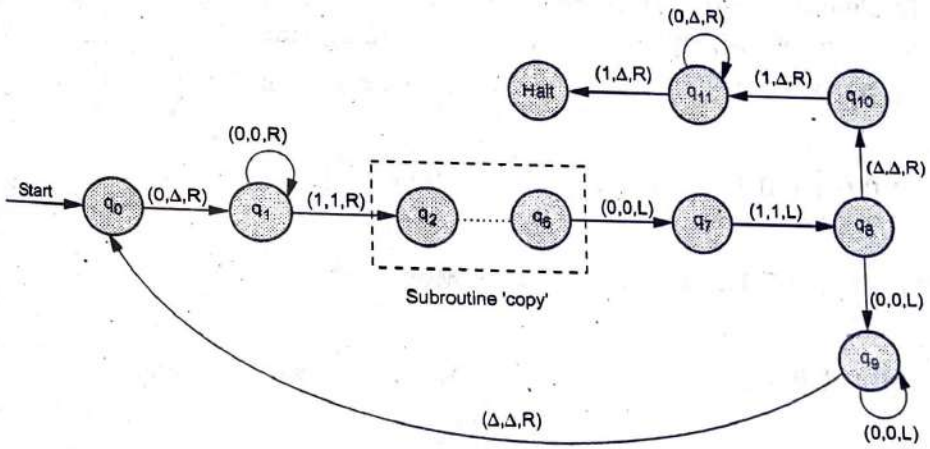


Fig. Q.15.2

The subroutine copy can be as shown below.

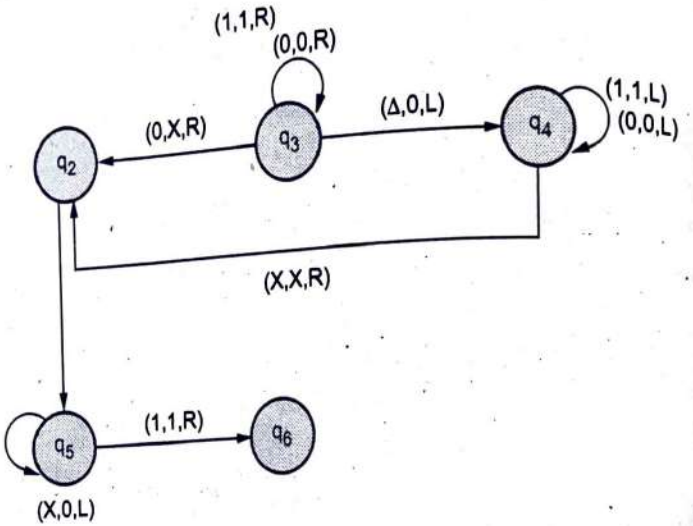


Fig. Q.15.3

To simulate above TM, let us take an example for multiplication of 3 and 2. The inputs are represented by unary 0 and separated by 1.

Δ 0 0 0 1 0 0 1 Δ ...	Convert to Δ and move right.
Δ Δ 0 0 1 0 0 1 Δ Δ Δ Δ Δ Δ ...	Move right.
Δ Δ 0 0 1 0 0 1 Δ Δ Δ...	Move right.
Δ Δ 0 0 1 0 0 1 Δ Δ Δ ...	Move right and call 'copy'.
Δ Δ 0 0 1 0 0 1 Δ Δ Δ ...	Convert 0 to X and move right.
Δ Δ 0 0 1 X 0 1 Δ Δ Δ ...	Move right.
Δ Δ 0 0 1 X 0 1 Δ Δ Δ ...	Move right.

Δ Δ 0 0 1 X 0 1 Δ ...	Convert to 0 and move left.
Δ Δ 0 0 1 X 0 1 0 Δ Δ ...	Move left.
Δ Δ 0 0 1 X 0 1 0 Δ Δ	Move left.
Δ Δ 0 0 1 X 0 1 0 Δ Δ	Move right.
Δ Δ 0 0 1 X X 1 0 Δ Δ	Convert 0 to X, move right.
Δ Δ 0 0 1 X X 1 0 Δ Δ	Move right, skip all 0's and 1's.
Δ Δ 0 0 1 X X 1 0 Δ	Convert Δ to 0 and move left.
Δ Δ 0 0 1 X X 1 0 Δ	Skip all 1's and 0's and reach at X by moving left.
Δ Δ 0 0 1 X X 1 0 0 Δ	Move right.
Δ Δ 0 0 1 X X 1 0 0 Δ	Move left and convert X to 0's.
Δ Δ 0 0 1 0 0 1 0 0 Δ	Move right.
Δ Δ 0 0 1 0 0 1 0 0 Δ	Move left, skip all 0's and 1's and reach at Δ by moving left.
Δ Δ 0 0 1 0 0 1 0 0 Δ	Move right convert 0 to Δ .
Δ Δ Δ 0 1 0 0 1 0 0 Δ	

Now again perform copy operation as illustrated above. The tape will then consist of

Δ Δ Δ 0 1 0 0 1 0 0 0 0 Δ Move right convert 0 to Δ and repeat the above illustrated steps.

↑

Δ Δ Δ Δ 1 0 0 1 0 0 0 0 0 0

↑

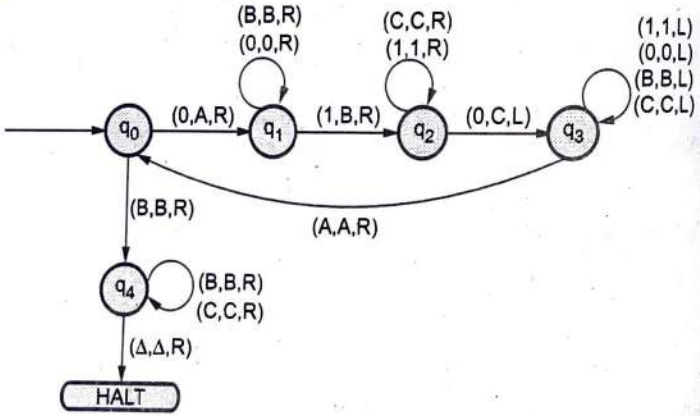
As we have read out the value of m from m × n we will convert remaining symbols to Δ leaving only multiplication result on the tape.

Thus finally Δ Δ Δ Δ Δ Δ Δ 0 0 0 0 0 0 0 Δ HALT

Q.16 Design turing machine that accepts a language
$L = \{0^n 1^n 0^n \mid n >= 1\}$.

☞[SPPU : May-15, (End Sem), Dec.-18, Marks 8]

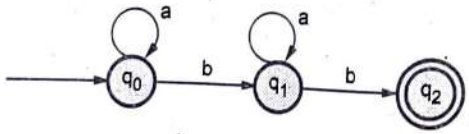
Ans. :



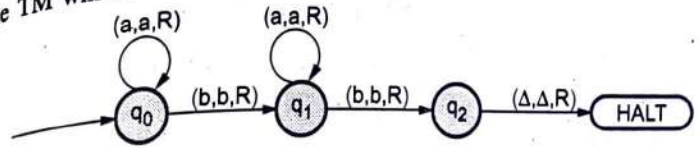
Q.17 Construct a TM that accepts a language L, a^*ba^*b.

☞[SPPU : May-15, (End Sem), Dec.-18, Marks 6]

Ans. : We will construct FA for given L i.e. a*ba*b

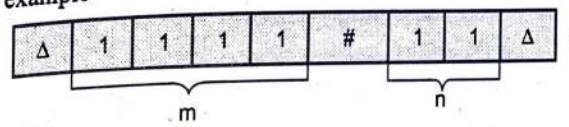


The TM will be



Q.18 Design turing machines for each of the following problems : i) Given two unary numbers, m and n, display, 'G', if m > n, 'E', if m = n, 'L', if m < n **☞[SPPU : Dec. 2015, (End Sem), Marks 10]**

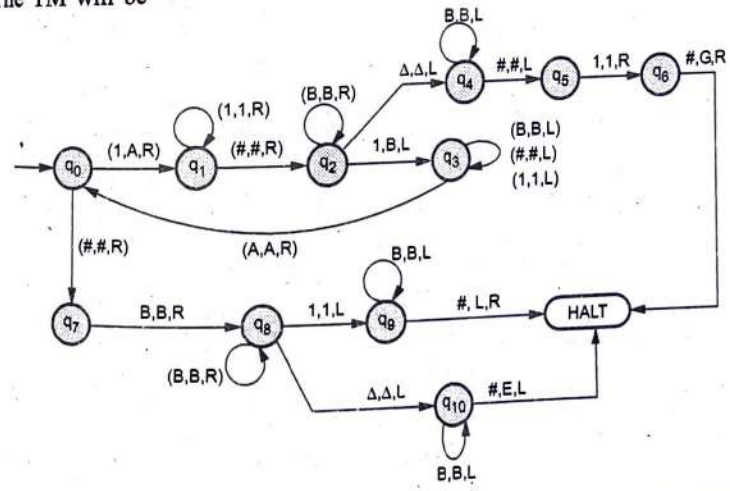
Ans. : i) The two unary numbers are placed on the tape and are separated by #. For example



The logic can be as follows :

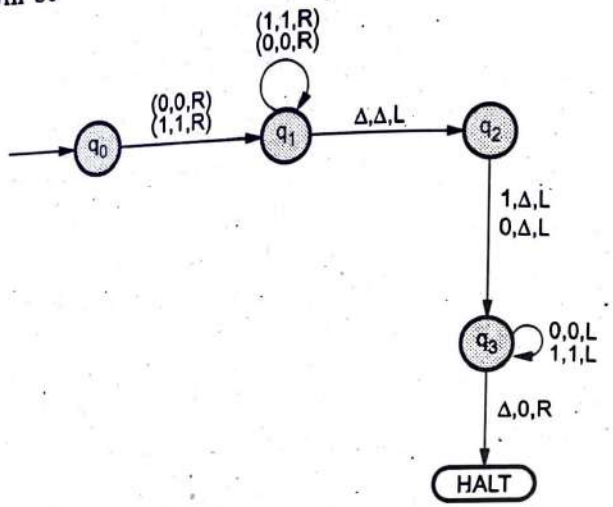
m > n case	m < n case	m = n case
Δ 1111#11 Δ	Δ 11 # 1111 Δ	Δ11 # 11 Δ
Δ A111#B1 Δ	Δ A1# B111 Δ	ΔA1 # B1 Δ
Δ AA11#BB Δ	Δ AA # BB11 Δ	ΔAA # BB Δ
Δ AA11GBB Δ	Δ AA L BB11 Δ	Δ AA E BB Δ

The TM will be -



Q.19 Design a turing machine to perform right shift operation on a binary number. **[SPPU : Dec. 2015, (End Sem), Marks 6]**

Ans. : In right shift operation the LSB bit is discarded and at MSB the padding 0 is attached. The right shift operation also means division by 2. The TM will be



Q.20 Construct a two tape TM to convert an input W into WW^R.

[SPPU : May-16, End Sem, Marks 6]

Ans. :

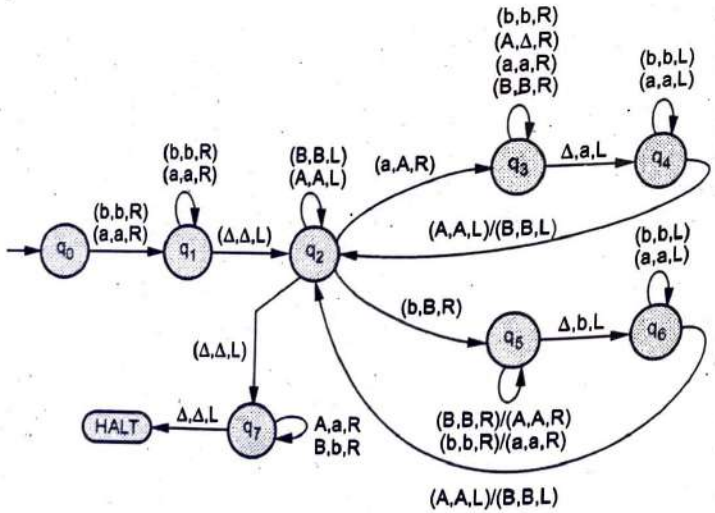


Fig. Q.20.1

Simulation of string Δ a b a b Δ

Δ a b a b Δ ... ↑	Read a, move right towards Δ.
Δ a b a b Δ ↑	Read Δ, keep Δ as it is and move left.
Δ a b a b Δ ↑	Read b, convert it to B, move right.
Δ a b a B Δ ↑	Convert Δ to be and move left.
Δ a b a B b ↑	Read B as it is and move left.
Δ a b a B b Δ ↑	Read a, convert it to A move right.
Δ a b a B b Δ ↑	Move towards Δ.
Δ a b A B b Δ ↑	Convert Δ to a and move left.
Δ a b A B b a Δ ↑	Keep moving to left by skipping a's or b's.
Δ a b A B b a Δ ↑	Read B, move left. Skip all capital letters and move towards left.
Δ a b A B b a Δ ↑	Read b, convert it to B and move right.
Δ a B A B b a Δ ↑	Go on moving right towards Δ.
Δ a B A B b a Δ ↑	Convert Δ to b and move left.

ΔaBABbabΔ Keep moving left towards a.
↑

ΔaBABbabΔ Convert a to A. Keep moving right towards Δ.
↑

ΔABABbabΔ Convert Δ to a
↑

ΔABABbabaΔ Keep moving left towards A.
↑

ΔABABbabaΔ Move left.
↑

ΔABABbabaΔ Move right convert all capitals to small letters.
↑

ΔababbabaΔ HALT.
↑

Q.21 Construct a TM for the subroutine
f (a, b) = a * b
where a and b are unary numbers. ☞ **[SPPU : May-19, Marks 8]**

Ans. : The unary number are represented by 1's. That if $a = 2$ and b = 3 then for this subroutine we will perform a copy operation. That is marking first 1 of a and copying it after Δ at b number of times, similarly marking second 1 of a and copying it at the rightmost end (i.e. after previously copied 1's) for b number of times. This makes the multiplication function complete.

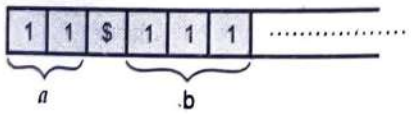


Fig. Q.21.1

			1 ↑	1	$	1	1	1	Δ	
			X	1 ↑	$	1	1	1	Δ	Moving to right upto $
			X	1	$	1 ↑	1	1	Δ	Copy this 1 after Δ by marking it as Y.
	X	1	$	Y	1	1	Δ	1 ↑	Δ	Now move to left upto Y
	X	1	$	Y ↑	1	1	Δ	1	Δ	
	X	1	$	Y	1 ↑	1	Δ	1	Δ	Mark this 1 as Y and copy this 1 at the rightmost end.
X	1	$	Y	Y	1	Δ	1	1 ↑	Δ	Now move to left upto Y
X	1	$	Y	Y ↑	1	Δ	1	1	Δ	
X	1	$	Y	Y	1 ↑	Δ	1	1	Δ	Mark this 1 as Y and copy this 1 at the rightmost end.
X	1	$	Y	Y	Y	Δ	1	1	1	Now move to left upto Y
X	1	$	Y	Y	Y ↑	Δ	1	1	1	

The head is pointing to Y next Y is Δ that means all the 1's are over we will convert these Y to 1's and repeat the same procedure for the 1 which is left to $.

	X	1	$	1	1	1	Δ	1	1	1	
	X	X	$	1	1	1	Δ	1	1	1	Convert this 1 to X and move to right upto after the $ symbol.
		↑									
	X	X	$	1	1	1	Δ	1	1	1	Mark it Y and copy this 1 to rightmost end
				↑							
X	X	$	Y	1	1	Δ	1	1	1	1	Move to left upto Y
										↑	
X	X	$	Y	1	1	Δ	1	1	1	1	Convert this 1 to Y and move to right, copy this 1 to rightmost end
				↑							
X	X	$	Y	Y	1	Δ	1	1	1	1	Move to left upto Y
										↑	

The above steps will be repeated and all the 1's between $ and Δ are copied to the rightmost end. This will be like this

X X $ Y Y Y Δ 1 1 1 1 1 1

Now we will move to left by converting all Y's and X's to 1's. Finally input tape will have

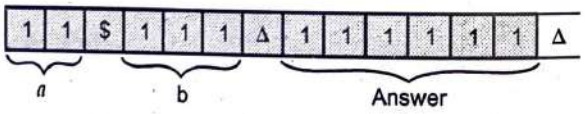


Fig. Q.21.2

We put these actions together and draw the transition graph and model out the turing machine.

The HALT state is a final state in the above TM.

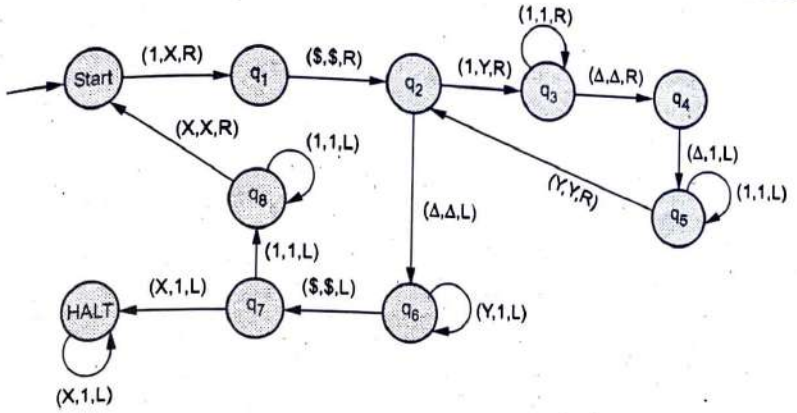


Fig. Q.21.3 Turing machine for multiplication subroutine

Q.22 Construct a Turing Machine to accept the language of even number of 1's and even number 0's over Σ = {0, 1}.

☞[SPPU : Dec.-17, End Sem, Marks 8]

Ans. :

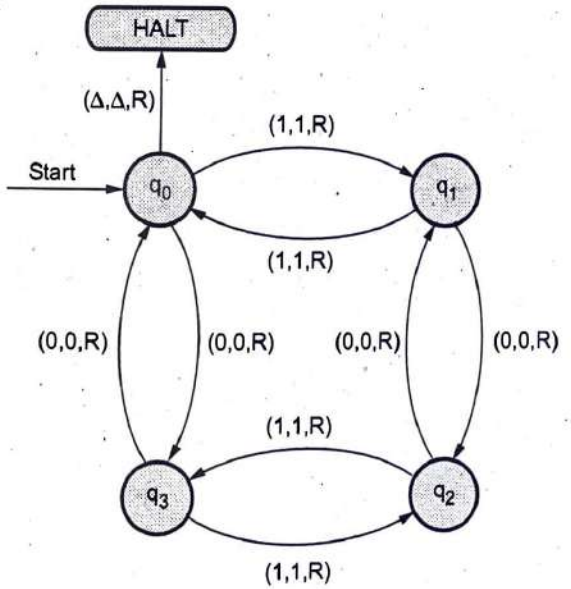


Fig. Q.22.1

Q.23 Design a Turing Machine to add two unary numbers.

☞[SPPU : Dec.-17, Marks 6]

Ans. :

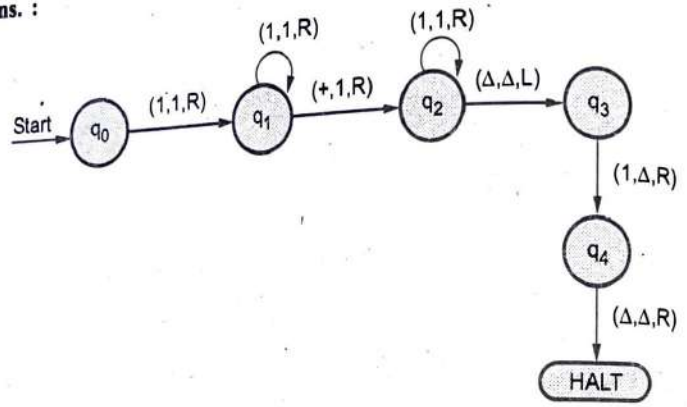


Fig. Q.23.1

Q.24 Construct TM for 1's complement of binary number.

☞[SPPU : May-18 (End Sem), Marks 6]

Ans. :

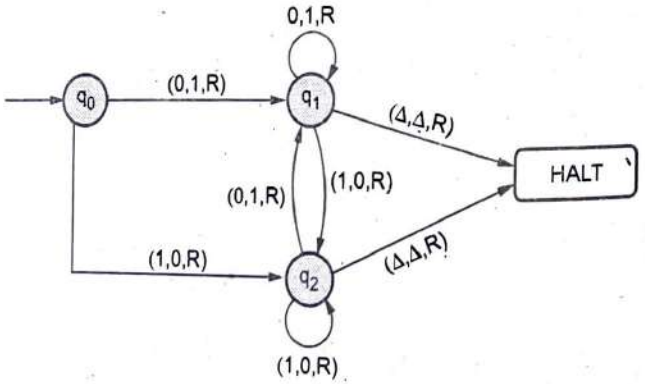


Fig. Q.24.1

Q.25 Design a Turing Machine to accept the language. L = {w/w ∈ (0 + 1) *} containing the substring 001.

☞[SPPU : May-18 (End Sem), Marks 8]

Ans. :

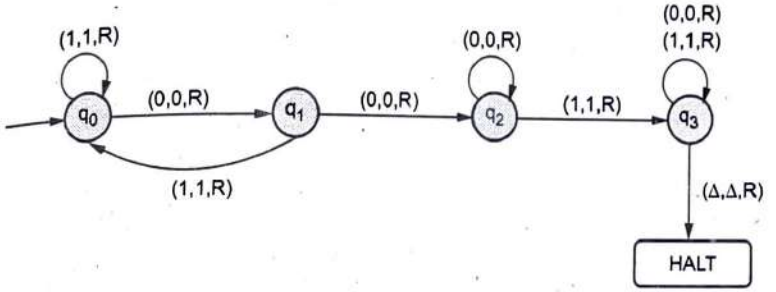


Fig. Q.25.1

Q.26 Construct a Turing Machine for R = (a + b)*bb.

☞[SPPU : Dec.-18 (End Sem), Marks 6]

Ans. :

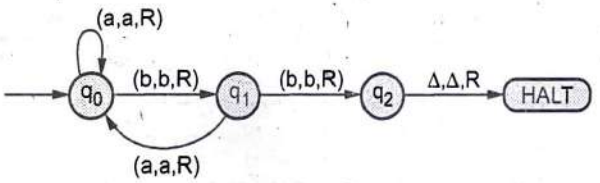


Fig. Q.26.1

Q.27 Construct a TM to compute L = $\{a^n b^{2n} | n \geq 0\}$ Write simulation for the string.

i) abb ii) aabbbb **☞[SPPU : May-19 (End Sem), Marks 10]**

Ans. : The Turing machine is,

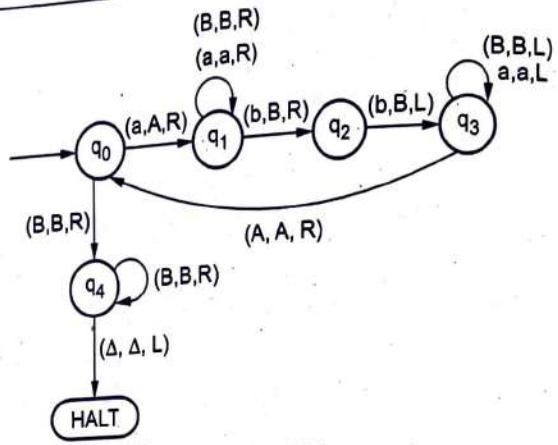


Fig. Q.27.1

Simulation :

i) **abb**

Input	Action taken
abbΔ ↑	Convert a to A and move right.
AbbΔ ↑	Convert b to B and move right.
ABbΔ ↑	Convert b to B and move left
ABBΔ ↑	Move left upto A
ABBΔ ↑	Move right
ABBΔ ↑	Move right
ABBΔ ↑	Move right
ABBΔ ↑	ACCEPT

ii) **aabbbb**

Step	Input	Action taken
1.	aabbbbΔ ↑	Convert a to A and move right.
2.	Aabbbb Δ ↑	Move right upto b.
3.	AaBbbbΔ ↑	Move rith convert b to B.
4.	AaBBbbΔ ↑	Move left upto A
5.	AaBBbbΔ ↑	Move Right. Then repeat step 1 to 4.
6.	AABBBBΔ ↑	Move right. Move right upto Δ by skipping B's.
7.	AABBBBΔ ↑	ACCEPT

Q.28 Design TM for the language $L = \{0^{2n}\}$ over $\sum = \{0,1\}$.

☞[SPPU : May-19 (End Sem), Marks 8]

Ans. : We will design finite automata for

$L = \{0^{2n}\}$ over $\{0, 1\}$

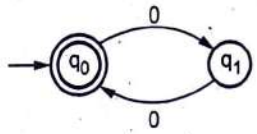


Fig. Q.28.1

From this FA we can design TM as follows -

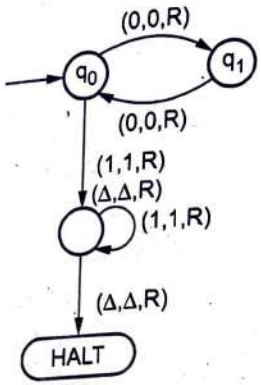


Fig. Q.28.2

Q.29 Design TM to accept the set L of all strings formed with 0 and 1 and having substring '000'. ☞ **[SPPU : May 19(End Sem), Marks 8]**

Ans. : The FA can be designed as

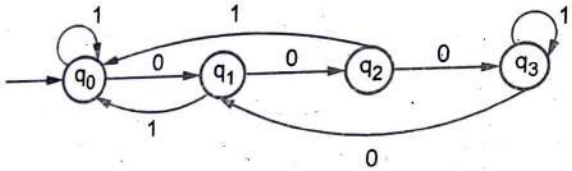


Fig. Q.29.1

From this FA we can design TM as

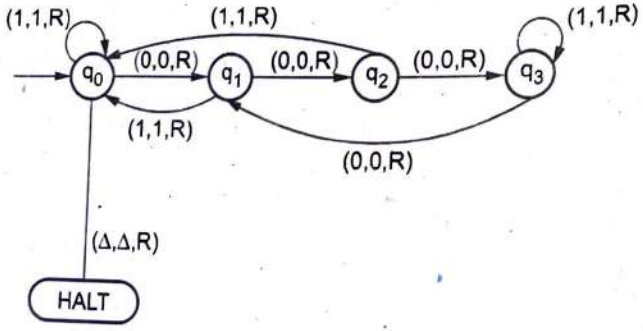


Fig. Q.29.2

Q.30 Construct a Turing Machine which accepts odd length palindrome over the $\Sigma = \{a, b\}$ ☞ **[SPPU : Dec.-19, Marks 6]**

Ans. :

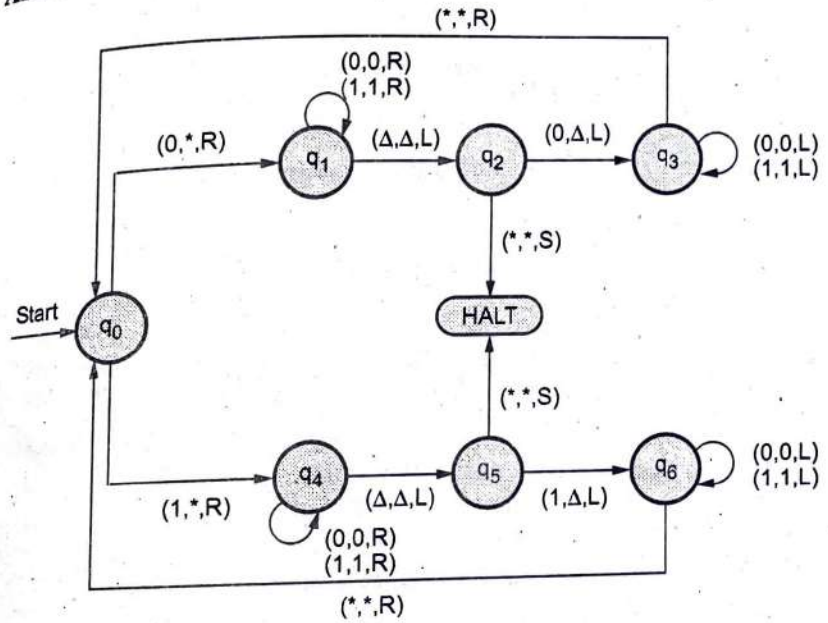


Fig. Q.30.1

For example : If the string is

0 0 1 0 0 Δ ↑	Move right by making 0 to *
* 0 1 0 0 Δ ↑	Move right upto Δ
* 0 1 0 0 Δ ↑	Move left, replace 0 by Δ
* 0 1 0 Δ Δ ↑	Move left

* 0 1 0 Δ ↑	Move left upto *
* 0 1 0 Δ ↑	Move right convert 0 to * and then move right
* * 1 0 Δ ↑	Move right upto Δ
* * 1 0 Δ ↑	Move left and replace 0 by Δ
* * 1 Δ Δ ↑	Move left till *
* * 1 Δ Δ ↑	Move right and convert 1 to *
* * * Δ Δ ↑	Move right
* * * Δ Δ ↑	Move left
* * * Δ Δ ↑	Since it is *, goto HALT state

Q.31 What is a Turing machine ? Give formal definition of TM. Design a TM to compute multiplication of two unary numbers.
(Refer Q.1 and Q.21) **☞[SPPU : May-17, End Sem, Marks 9]**

Q.32 What are the different ways for extension of TM ? Explain. Construct a two tape TM to convert an input W into WW^R.

☞[SPPU : May-17, End Sem, Marks 9]

Ans. : Following are the extension of TM -

1. Two Way Infinite Tape Turing Machine : This is a type of Turing machine in which the input is placed on a tape which is infinite in both

the directions. The Turing Tape head can move towards left as well as towards right.

2. Multiple Track Turing Machine : If the input tape is divided into multiple tracks then the input tape will be as follows -

For example :

As shown in Fig. Q.32.1 the input tape has multiple tracks on the first track. The input which is placed is surrounded by # and $. The unary number equivalent to 5 is placed on the input tape, on the first track. On the second track unary 2 is placed. If we construct a TM which subtracts 2 from 5 we get the answer on the third track and that is 3, in unary form.

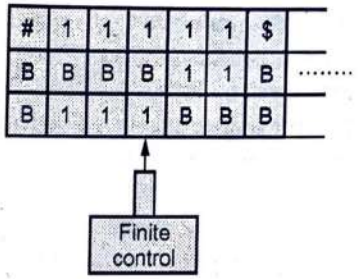


Fig. Q.32.1 Multiple tracks

3.Multiple Tape Turing Machine The multitape turing machine is a type of turing machine in which there are more than one input tapes. Each tape is divided into cells and each cell can hold any symbol of finite tape alphabet. The multitape turing machine is as shown in the Fig. Q.32.2.

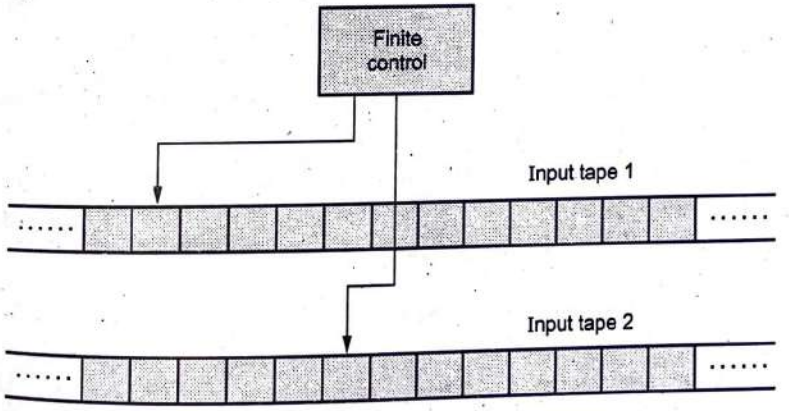


Fig. Q.32.2 A multitape turing machine

This TM is more powerful than the basic turing machine. Because finite control reads more than one input tape and more symbols can be scanned at a time.

Two Tape TM to convert an input W to WW^R - Refer Q.20.

5.4 : Variants of Turing Machines

Q.33 Explain the concept of deterministic and Non deterministic Turing Machine. **☞ [SPPU : May-18, End sem]**

Ans. : Deterministic Turing Machine : The deterministic Turing machine is a kind of turing machine in which the set of rules denote the single specific action on reading particular input in current specific state. It is just similar to deterministic finite automata.

Non Deterministic Turing Machine : The non deterministic turing machine is a kind of turing machine in which the set of rules denote more than one specific action on reading particular input in current specific state. This kind of TM just similar to NFA.

For example -

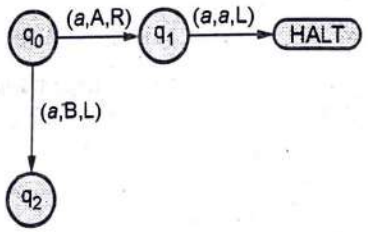


Fig. Q.33.1

Any language accepted by non deterministic turing machine can also be accepted by deterministic turing machine.

Q.34 Write short note on - i) Universal Turing Machine ii) Multitape Turing Machine **☞[SPPU : Dec.-17, Marks 4]**

Ans. : i) Universal Turing Machine

- The universal language L_u is a set of binary strings which can be modeled by a turing machine.
- The universal language can be represented by a **pair (M, w)** where M is a TM that accepts this languages and w is a binary string in$(0 + 1)^*$ such that w belongs to L(M). Thus we can say that any binary string belongs to a universal language.
- The **universal language** can be represented by $L_u = L(U)$ where U is a universal Turing Machine.

- In fact U is a binary string. This binary string represents various codes of many Turing machines.
- Thus the universal turing machine is a Turing machine which accepts many Turing machines.

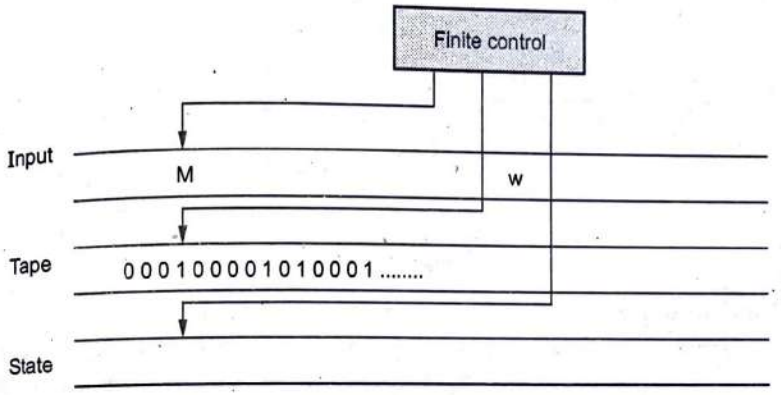


Fig. Q.34.1

ii) Multitape Turing Machine : Refer Q.32.

5.5 : Halting Problem of TM

Q.35 Write a short note on Halting problem.

☞[SPPU : May-16, End Sem, Marks 8, May-17, End Sem, Dec.-19, Marks 4, May-18, Marks 2]

OR Define Undecidability. Let $HALT_{TM}$ = {<M,w> where M is a TM and M halts on input w}. Prove that $HALT_{TM}$ is undecidable.

☞[SPPU : Dec.-18, May-19, Marks 8]

Ans. : Halting Problem can be stated as follows - Given any functional matrix, input data tape and initial configuration, then it is decide whether the process will run forever or will eventually stop."

The halting problem is unsolvable.

Proof : Let, there exists a TM M_1 which decides whether or not any computation by a TM T will ever halt when a description d_T of T and

tape t of T is given [That means the input to machine M_1 will be (machine, tape) pair]. Then for every input (t, d_T) to M_1 if T halt for input t, M_1 also halts which is called accept halt. Similarly if T does not halt for input t then the M_1 will halt which is called reject halt. This is shown in Fig. Q.35.1.

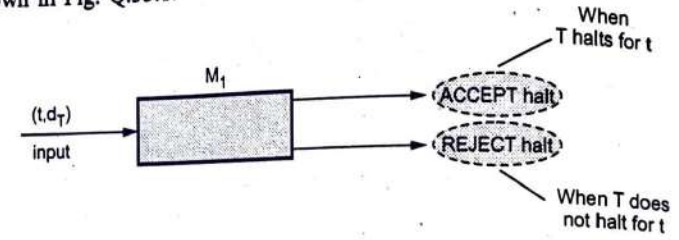


Fig. Q.35.1

Now we will consider another Turing Machine M_2 which takes an input d_T. It first copies d_T and duplicates d_T on its tape and then this duplicated tape information is given as input to machine M_1. But machine M_1 is a modified machine with the modification that whenever M_1 is supposed to reach an accept halt, M_2 loops forever. Hence behavior of M_2 is as given. It loops if T halts for input t = d_T and halts if T does not halt for T = d_T. The T is any arbitrary turing machine.

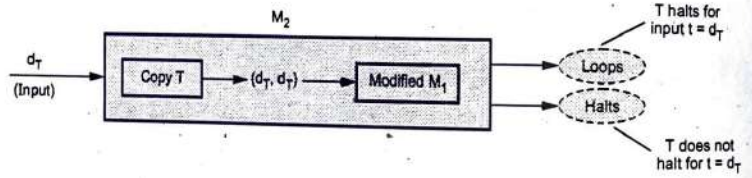


Fig. Q.35.2

As M_2 itself is one turing machine we will take M_2 = T. That means we will replace T by M_2 from above given machine.

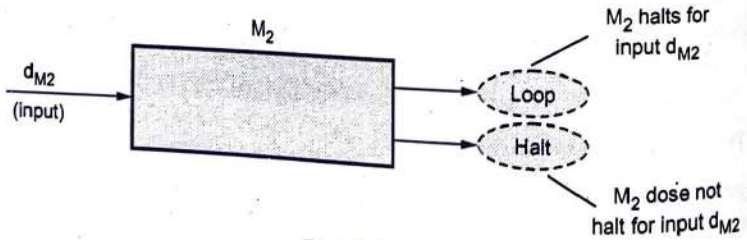


Fig. Q.35.3

Thus machine M_2 halts for input d_{M2} if M_2 does not halt for input d_{M2}. This is a contradiction. That means a machine M_1 which can tell whether any other TM will halt on particular input does not exist. Hence halting problem is unsolvable.

Q.36 Differentiate between FA and TM.

☞ **[SPPU : May-19, End Sem, Marks 2]**

Ans. :

Sr. No.	FA	TM
1.	It accepts regular languages.	It accepts recursive and recursively enumerable languages.
2.	It can not be programmed.	It can be programmed.
3.	It accepts small class of language.	It accepts large class of languages.

5.6 : Halting Vs Looping

Q.37 Explain Halting Vs. Looping.

Ans. : Halting problem is denoted by following question - "Given a program or algorithm will ever halt or not.

Halting means that the program on certain input will either **accept it and halt** or **reject it and halt.** It would never go into an infinite loop. Basically halting means terminating. So can we have an algorithm that will tell that the given program will halt or not.

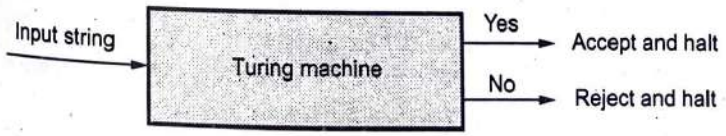


Fig. Q.37.1 Halting

Looping means the program on certain input will either accept it and enter it in infinite loop or reject it and halt.

The halting and looping both are undecidable problems.

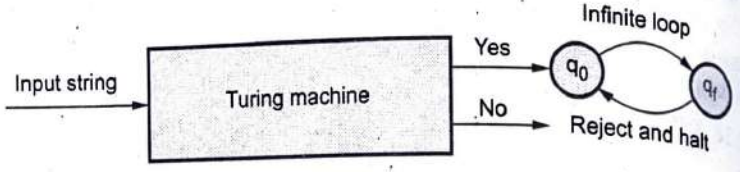


Fig. Q.37.2 Looping

5.7 : A Turing-unrecognizable language

Q.38 Define and explain recursive and recursively enumerable languages. **☞[SPPU : May-14,17, End Sem, Marks 4]**

Ans. : Recursive Languages :

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

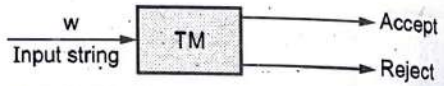


Fig. Q.38.1 TM Accepting Recursive Language

Recursively Enumerable Languages :

A language is said to be recursive if there exists a Turing machine that **accepts** every string belonging to that language. And if the string does not belong to that language then it can cause a Turing machine to enter in an **infinite loop**.

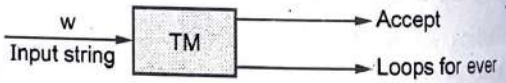


Fig. Q.38.2 TM Accepting RE language

5.8 : Reducibility

Q.39 Explain the concept of reducibility.

Ans. : Reducibility

- Reducibility is problem solving technique in mathematics.
- Let A and B are two problems and A is reduced to B. If we solve B, we solve A as well. If we can't solve A, we can't solve B.

- For example - If we solve the Eulerian cycle problem, we solve the Eulerian path problem.
- To decide if a problem is solvable or not we use mapping reducibility technique.
- If there is a mapping reduction from language A to language B, we say that language A is mapping reducible to language B.
- Notation used for mapping reducibility is $A \leq M^B$ iff language A is mapping reducible to language B.
- If A is reduced to B and B is decidable, then A is decidable.
- If A is undecidable and reducible to B then B is also undecidable.

5.9 : Recursion Theorem

Q.40 What is recursive function ? Explain initial function.

Ans. : Recursive function is a class of functions those are Turing Computable.

Initial function : The initial functions are the elementary functions whose values are independent of their smaller arguments. The following functions comprise the class of recursive functions.

The zero function : $z(x) = 0$

The successor function : $s(x)$ = successor of x (roughly , "x+1")

The identity function : $id(x) = x$

The zero functions returns zero regardless of its argument.

The successor functions returns the successor of its argument. Since successorship is a more primitive notion.

The zero and successor functions take only one argument each. But the identity functions is designed to take any number of arguments. When it takes one argument (as above) it returns its argument as its value. When it takes more than one argument, it returns one of them. That means,

$$id(x, y) = x$$

$$id(x, y) = y$$

Q.41 Explain the class of recursive functions.

Ans. : Class of Recursive Functions

The classification of recursive functions is as shown in Fig. Q.41.1.

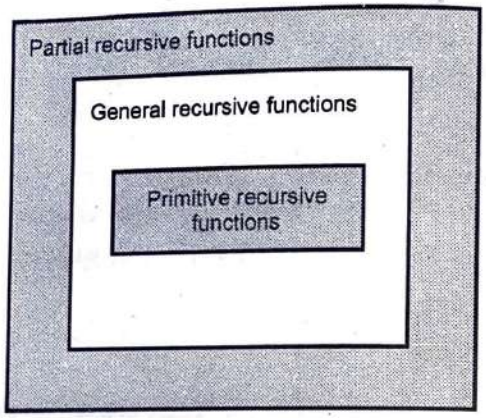


Fig. Q.41.1 Classification of recursive functions

1. Partial Recursive Function

The function is called **partial recursive function** if it can be obtained by applying composition, primitive recursion and minimization as building operations.

2. General Recursive Function

The function is called **general recursive** if it is obtained by applying composition, primitive recursive and an unbounded minimization that happen to terminate. The general recursive function is a larger class than partial recursive functions.

3. Primitive Recursive Function

The function is called **primitive recursive** if it is obtained by applying composition, primitive recursion and unbounded minimization that does not terminate. The set of general recursive function is the same as the set of turing computable functions. The example of general recursive function is an Ackermann's function. The Ackermann's function can be defined as follows -

$$A(0, y) = y + 1$$

$$A(x+1, 0) = A(x, 1)$$

$$A(x+1, y+1) = A(x, A(x+1, y))$$

Then we can calculate A(x, y) for every pair of (x, y).

Q.42 Compute A(1, 1) A(1, 2), A(2, 1), A(2, 2) using Ackermann's function.

Ans. : Let,

$$A(0, y) = y + 1 \quad \ldots (1)$$

$$A(x+1, 0) = A(x, 1) \quad \ldots (2)$$

$$A(x+1, y+1) = A(x, A(x+1, y)) \quad \ldots (3)$$

By the Ackermann's function,

To compute A(1, 1) put x = 0, y = 0, then

$$A(1, 1) = A(0 + 1, 0 + 1) \quad \text{...Using equation (3)}$$

$$= A(0, A(0 + 1, 0))$$

$$= A(0, A(0, 1)) \quad \text{...Using equation (2)}$$

$$= A(A(0, 1)) \quad \text{... Using equation (1)}$$

$$= A(0, 2) \quad \text{...Using equation (1)}$$

Hence A(1, 1) = 3

To compute A(1, 2) put x = 0 and y = 1

$$A(1, 2) = A(0 + 1, 1 + 1) \quad \text{...Using equation (3)}$$

$$= A(0, A(1, 1))$$

$$= A(0, 3) \quad \text{...Using equation (1)}$$

$$= 4$$

To compute A(2, 1) put x = 1 and y = 0

$$A(2, 1) = A(1 + 1, 0 + 1)$$

$$= A(1, A(2, 0))$$

$$= A(1, A(1, 1))$$

$$= A(1, 3)$$

$$= A(0 + 1, 2 + 1)$$

$$= A(0, A(1, 2))$$

$$= A(0, 4)$$

$= 5$

To compute A(2, 2) put x = 1 and y = 1

$$A(2, 2) = A(1 + 1, 1 + 1)$$
$$= A(1, A(2, 1))$$
$$= A(1, 5)$$

Now we will compute A(1, 5) where in x = 0 and y = 4.

$$A(1, 5) = A(0 + 1, 4 + 1)$$
$$= A(0, A(1, 4))$$
$$= A(0, (A(0 + 1, 3 + 1)))$$
$$= A(0,(A(0,A(1, 3))))$$
$$= A(0,(A(0,A(0 + 1,(2 + 1)))))$$
$$= A(0, (A(0,A(0,A(1, 2)))))$$
$$= A(0,(A(0,A(0,(4))))$$
$$= A(0,(A(0, 5)))$$
$$= A(0, 6)$$
$$A(1, 5) = 7$$

Hence $A(2, 2) = A(1, 5)$

$$A(2, 2) = 7$$

5.10 : The Model of Linear Bounded Automata

Q.43 Explain the concept of linear bounded automata and context sensitive language.

Ans. : Concept : A linear bounded automaton is a multitrack Turing machine which has only one tape and this tape is exactly of same length as that of input.

- In LBA computation is restricted to an area bounded by length of the input. This is very much similar to programming environment where size of variable is bounded by its data type.
- The LBA is powerful than Non deterministic Push Down Automata(NPDA) but less powerful than Turing machine.

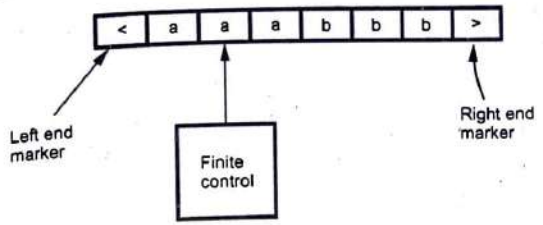


Fig. Q.43.1 Linear bound automaton

A linear bounded automata can be **formally defined as :**

It is 7 - tuple non deterministic Turing machine with

$$M = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject}) \text{ having}$$

1. Two extra symbols of left end marker and right end marker which are not elements of Γ.
2. The input lies between these end markers.
3. The TM cannot replace < or > with anything else nor move the tape head left of < or right of >.

Context Sensitive Languages (C.S. Languages)

The context sensitive languages are the languages which are accepted by linear bounded automata. These type of language are defined by context sensitive grammar. In this grammar more than one terminal or non terminal symbol may appear on the left hand side of the production rule. Along with it, the context sensitive grammar follows following rules -

i) The number of symbols on the left hand side must not exceed number of symbols on the right hand side.

ii) The rule of the form $A \rightarrow \varepsilon$ is not allowed unless A is a start symbol. It does not occur on the right hand side of any rule.

The classic example of context sensitive language is $L = \{a^n b^n c^n | n \geq 1\}$. The context sensitive grammar can be written as -

$$S \rightarrow aBC$$
$$S \rightarrow SABC$$
$$CA \rightarrow AC$$
$$BA \rightarrow AB$$

$CB \rightarrow BC$

$aA \rightarrow aa$

$aB \rightarrow ab$

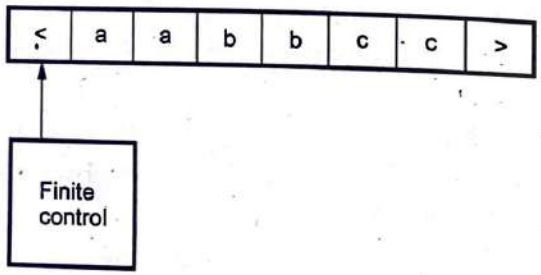
$bB \rightarrow bb$

$bC \rightarrow bc$

$cC \rightarrow cc$

Q.44 Construct a language $L = \{a^n\ b^n\ c^n \mid n \geq 1\}$ using LBA.

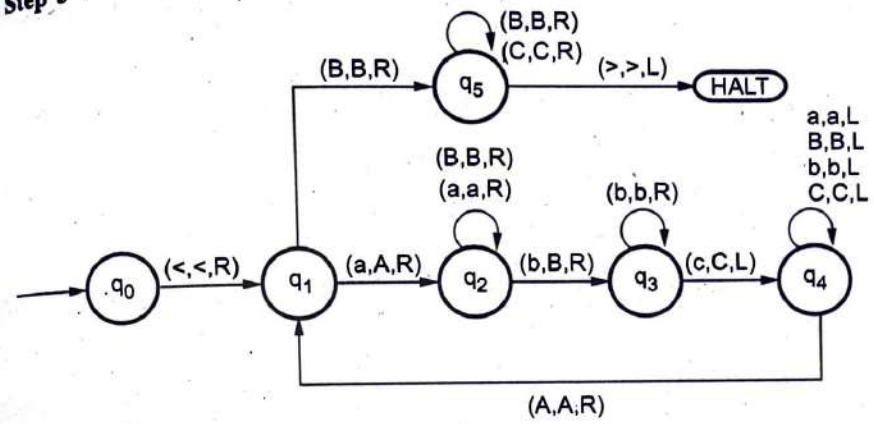
Ans. :



Step 1 : The input is placed on the input tape which is enclosed within left end marker and right end marker.

Step 2 : We will apply the simple logic as : when we read 'a' convert it to A then move right by skipping all a's. On encountering first 'b' we will convert it to B. Then move right by skipping all b's. On receiving first c convert it to C. Move in left direction unless you get A. Repeat the above procedure and convert equal number of a's, b's, and c's to corresponding A's, B's and C's.- Finally move completely to the rightmost symbol if it is '>' a right end marker, then HALT. The machine will be -

Step 3 :



Simulation : Consider input **aabbcc**

< aabbcc > ↑	Move right.
< aabbcc > ↑	Convert to A, move right.
< Aabbcc > ↑	Move right.
< Aabbcc > ↑	Convert to B, move right.
< AaBbcc > ↑	Move right.
< AaBbcc > ↑	Convert to C, move left.
< AaBbCc > ↑	Move left.
< AaBbCc > ↑	Move left.

< AaBbCc > ↑	Move left.
< AaBbCc > ↑	Move right.
< AaBbCc > ↑	Convert to A, Move right.
< AABbCc > ↑	Move right.
< AAB bCc > ↑	Convert to B, move right.
< AABBCc > ↑	Move right.
< AABBCc > ↑	Convert to C, move left.
< AABBCC > ↑	Move left continuously by skipping B's.
< AABBCC > ↑	Move right
< AABBCC > ↑	If we get B, we will move right to check whether all b's and c's are converted to B and C.
< AABBCC > ↑	If we get right end marker '>' then we HALT by accepting the input aabbcc.

Thus in LBA the length of tape exactly equal to the input string and tape head can not move left of '<' and right of '>'.

Q.45 Write short notes on :
i) Universal Turing Machine ii) Multi-tape Turing Machine
iii) Limitation of Turing Machine

☞[SPPU : Dec.-17, End Sem, Marks 6]

Ans. : (1) Universal Turing Machine

- The universal language L_u is a set of binary strings which can be modeled by a turing machine.
- The universal language can be represented by a **pair (M, w)** where M is a TM that accepts this languages and w is a binary string in$(0 + 1)^*$ such that w belongs to L(M). Thus we can say that any binary string belongs to a universal language.
- The **universal language** can be represented by $L_u = L(U)$ where U is a universal Turing Machine.
- In fact U is a binary string. This binary string represents various codes of many Turing machines.

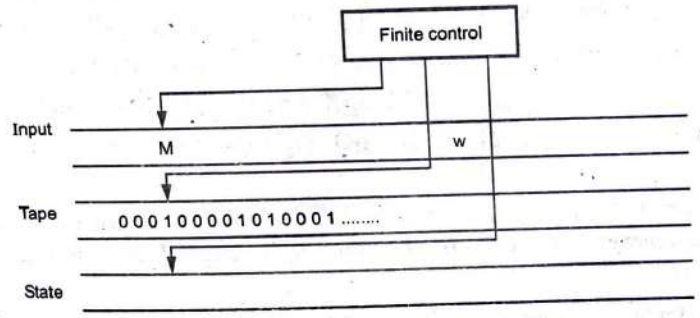


Fig. Q.45.1

- Thus the universal turing machine is a Turing machine which accepts many Turing machines.

(2) Multi-tape Turing Machine : The multitape turing machine is a type of turing machine in which there are more than one input tapes. Each tape is divided into cells and each cell can hold any symbol of finite tape alphabet. The multitape turing machine is as shown in the Fig. Q.45.2.

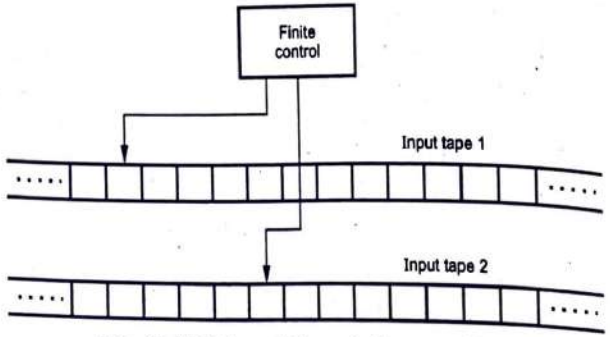


Fig. Q.45.2 A multitape turing machine

(3) Limitations of Turing Machine

- Turing machine is a model which can solve any problem which has some algorithm. But not all the problems be solved by TM. There are some problems which cannot be solved by TM. For example : Halting problem cannot be solved by TM. The Halting problem can be stated as follows - "It is not possible to determine whether a TM will ever halt on particular input". Such a problem is called unsolvable problem. There some more problems that are not solvable by TM -

i) Deciding whether two context free grammars are equivalent.

ii) Whether a given program will loop forever on particular input.

iii) If two CFGS are given G_1 and G_2 then $L(G_1) \cap L(G_2) = \phi$. is undecidable.

iv) Whether the language accepted by TM is empty it is not possible for a TM to answer "yes" or "no" for above problems. Hence these problems are unsolvable by TM.

Q.46 Differentiate between FA and TM. **☞[SPPU : May-19, Marks 2]**

Ans. :

Sr. No.	FA	TM
1.	It accepts regular languages.	It accepts recursive and recursively enumerable languages.
2.	It can not be programmed.	It can be programmed.
3.	It accepts small class of language	It accepts large class of languages.

END...

Unit VI

6 Computability and Complexity Theory

6.1 Decidable Problems and Un-decidable Problems

Q.1 Explain the decidable and un-decidable problems.

Ans. : • If a language is recursive then it is called decidable language and if the language is not recursive then such a language is called undecidable language.

- The class of decidable problems are also called as solvable problems and the class of undecidable problems is called unsolvable problems.

Q.2 Prove that following decision problems are recursive :
(i) Two DFA's are equivalent or Not (ii) NFA Accepts a word or not.

☞[SPPU : May-15 End Sem, Marks 10, May-16, End Sem, Marks 5]

Ans. : Proof : (i)

- To prove that the given decision problem is recursive. We require a TM T(M) which simulates the DFA.
- Let, M_1 and M_2 are two DFAs. Consider the Turing machines $T(M_1)$ and $T(M_2)$ Obtain set of strings accepted by $T(M_1)$ and rejected by $T(M_2)$. That is

$$S_1 = T(M_1) \cap \overline{T(M_2)}.$$

- Similarly obtain the set of strings which $T(M_2)$ accepts and $T(M_1)$ rejects. That is

$$S_2 = T(M_2) \cap \overline{T(M_1)}.$$

- If these two sets are empty then it shows that $L(M_1)$ and $L(M_2)$ are equivalent and can be simulated by TM for the set of languages as.

$L(M) = L(M_1) \cap \overline{L(M_2)} \cup \overline{L(M_1)} \cap L(M_2)$. The $L(M) = \phi$

- This shows that two DFA's are equivalent or not is a recursive problem.

(II) Let, M be the NFA and input word w which decides if M accepts w. Convert this NFA to DFA. Then run the algorithm for DFA on Turing Machine T(M). The TM T(M) accepts w if and only if M reaches a final state on w. This shows that NFA accepts a word or not is recursive.

Q.3 Prove that CFG G generates the string w or not.

☞[SPPU : May-15, End Sem, Marks 5, May-16, End Sem, Marks 5]

Ans. : Convert CFG G to G″ in chomsky normal form. Now the string $w \in L(G'')$ iff w can be derived in $2|w|-1$ steps where none of the intermediate string is of length more than |w|. If these derived step will derive w, the T(M) accepting L(G″) will accept otherwise rejects. This shows CFG G generates string w or not is recursive.

Q.4 Justify - "It is undecidable whether a CFG is ambiguous".

☞[SPPU : Dec.-13, Marks 6]

Ans. : Let $A = (x_1, x_2 \ldots x_k)$ and $B = (y_1, y_2, y_3 \ldots y_k)$ are two strings.

The G_1 and G_2 are two context free grammars. The production rules for these grammars are as follows -

$S_1 \rightarrow a_i S_1 x_i$ for $i = 1, 2, 3, \ldots k$

$S_1 \rightarrow a_i x_i$

$S_2 \rightarrow a_i S_2 y_i$ for $i = 1, 2, 3, \ldots k$

$S_2 \rightarrow a_i y_i$

The grammar G_1 generates the sentence of the form

$a_{in}\ a_{in-1} \cdots a_{i1}\ x_{i1}\ x_{i2} \cdots x_{in}$

The grammar G_2 generates the sentence of the form

$a_{in}\ a_{in-1} \cdots a_{i1} y_{i1} y_{i2} \cdots y_{in}$

We can combine the languages from both the grammar. The new production which is added here is -

$S \rightarrow S_1 \mid S_2$

If grammar G_{12} is ambiguous then PCP with pair (A, B) has a solution.

Similarly if G_{12} is unambiguous then PCP with pair (A, B) has no solution.

But since having solution to PCP is undecidable and there is algorithm for decidability of PCP, there is no algorithm deciding whether given CFG is ambiguous or not. Hence whether CFG is ambiguous or not is undecidable.

6.2 Church-Turing Thesis

Q.5 Write short note on - Chruch- Turing Thesis.

Ans. : Hypothesis means proposing certain facts. The Church's hypothesis or Church's Turing thesis can be stated as,

> "The assumption that the intuitive notion of computable functions can be identified with partial recursive functions".

However, this hypothesis cannot be proved. The computability of recursive functions is based on following assumptions -

1) Each elementary function is computable.
2) Let f be the computable function and g be the another function which can be obtained by applying an elementary operation to f, then g becomes a computable function.
3) Any function becomes computable if it is obtained by rule 1 and rule 2.

6.3 Undecidable Problems that is Recursively Enumerable

Q.6 What is recursively enumerable language ?

Ans. : A language is called **recursively enumerable** if and only if the language is accepted by some Turing machine M.

Q.7 What is universal language L_u?

Ans. : The universal language L_u is a set of binary strings which can be modeled by a turing machine. The universal language can be represented by a pair (M, w) where M is a TM that accepts this languages and w is a binary string in $(0 + 1)^*$ such that w belongs to L(M).

Q.8 Show that the language L_u is recursively enumerable but not recursive.

Ans. : Consider that the language L_u is recursively enumerable. We can construct a TM accepting L_u. The TM U is a multitape turing machine which can be shown below.

The L_u can be represented by pair <M, w> where M is a TM and w be the binary string (0+1)* such that w ∈ L(M). The universal language L_u can be represented by L(U), where U is a universal turing machine. In fact U is a binary string. Thus binary string represents the code for many Turing machines. Thus as the L_u is acceptable by some turing machine M we can prove that L_u is recursively enumerable.

Now, to prove whether L_u is recursive or not consider L_d the diagonalization language.

Construct a list of strings over $\Sigma = \{0, 1\}^*$ in canonical order.

The i^{th} word is represented by w_i and j^{th} turing machine is represented by M_j.

The language L_d is constructed by diagonalization entries.

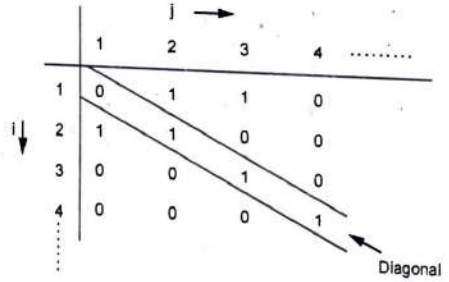


Fig. Q.8.1

When we get entry as 0 at (i, j) that means w_i is not in $L(M_j)$ and when we get entry as 1 at (i, j) that means string w_i is in $L(M_j)$.

Also $L_d = \{w_i | M_i \text{ does not accept } w_i\}$. If we assume L_d as RE then that means there exists a turing machine M_j that accepts L_d.

There arises two cases.

Case 1 : $w_j \in L_d$ - When (j, j) entry comes out to be '0' then only $w_j \in L_d$ i.e. M_j does not accept w_j.

But we have assumed w_j is acceptable by M_j. This is a contradiction.

Case 2 : $w_j \notin L_d$ - This means that (j, j) entry is 1. That means M_j accepts w_j. But L_d is defined as : $\{w_j | M_j \text{ does not accept } w_j\}$. This is contradiction. This means L_d is not RE hence it is not recursive. By reduction method we say that as L_d is not recursive then L'_d is also not recursive. By reducing L'_d to L_u we can prove that L_u is not recursive.

Q.9 Define undecidability. **☞[SPPU : May 2013, Marks 2]**

Ans. : The class of problems which are said to be unsolvable or can be answered as no are called undecidable problems

Examples of various undecidable problems are -

1. It is undecidable whether a function performs some arithmetic operation or print some message or calls some another function.
2. It is undecidable whether a CFG is ambiguous.
3. If G_1 and G_2 are context free grammars and R be a regular expression then following are undecidable -
   a. Is intersection of two context free languages empty ?
   b. Is language denoted by G_1 and G_2 both are equal ?

Q.10 For following decision problem, determine whether it is decidable or undecidable, prove the same.
1) Given a TM T, does it ever reach a state other than its initial state if it starts with a blank tape ?
2) Given a TM T, and a non halting state 'q' of T, does T ever enter state 'q' when it begins with a blank tape ?

☞[SPPU : May-11, Marks 8]

Ans. : 1) This problem is decidable. Following can be the situations that are possible with given scenario -

Case 1 : The TM T changes the state. That means it reaches to the state other than initial state (Answer to given problem is Yes).

Case 2 : The TM T simply changes the state and crashes. That means it reaches to the state other than the initial state (Answer to given problem is Yes).

Case 3 : The TM T simply moves its tape head to the right without changing the state (Answer to given problem is no i.e. TM remains in initial state only).

Case 4 : The TM T simply writes symbol on the currently pointed cell without changing the state (Answer to given problem is no). Thus we get either yes or no answers. Hence the given problem is **decidable.**

2) This problem is undecidable. To know the reason behind it, we will reduce this problem to - "Whether or not a turing machine accepts empty string".

This problem is **undecidable** problem.

This reduction is as follows -

Given a TM T (which accepts empty string) We construct a TM T′ which contains all the states of T and one more state i.e. q. T′ makes same moves as T but when T halts, T′ moves to state q and then in the next move it halts. That means T accepts empty string if and only if T′ enters the state q.

Q.11 Prove that if a language L and its complement L' both are RE then L is a recursive language. **☞[SPPU : Dec.-05, Marks 6]**

Ans. : Consider a turing machine M made up of two turing machines M1 and M2. The machine M2 is complement of machine M1. We can also denote that L(M) = L(M1) and L(M2). Both M1 and M2 are simulated in parallel by machine M. Machine M is a two tape TM, which can be made one tape TM for the ease of simulation. This One tape then will consist of tape of machine M1 and machine M2. The states of M consists of all the states of machine M1 and all the states of machine M2. The machine M made up of M1 and M2 is as shown in Fig. Q.11.1.

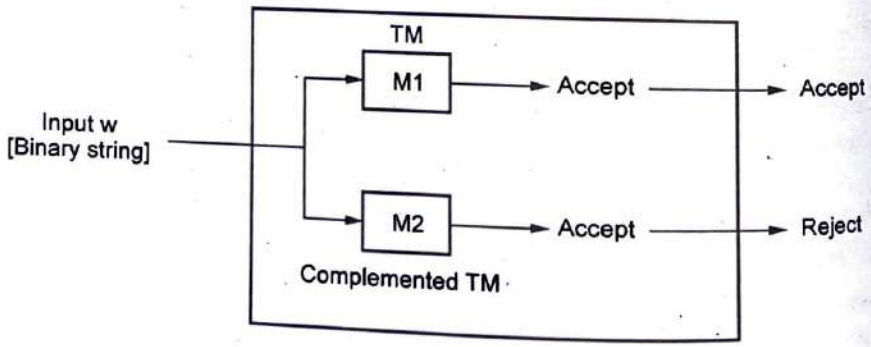


Fig. Q.11.1

If the input w of language L is given to M then M1 will eventually accept and therefore M will accept L and halt. If w is not in L, then it is in L'. So M2 will accept and therefore M will half without accepting. Thus on all inputs, M halts. Thus L(M) is exactly L. Since M always halts we can conclude that L(M) mean L is a recursive language. Thus a recursive language can be recursively enumerable but a recursively enumerable language is not necessarily be recursive.

Q.12 Prove that if L_1 and L_2 are recursive languages then $L_1 \cup L_2$ and $L_1 \cap L_2$ also recursive.

☞[SPPU : May-08, 10, Marks 8, Dec.-16, End Sem, Marks 6]

Ans. : Let

L_1 is a recursive language.

L_2 is a recursive language.

As L_1 and L_2 are recursive languages there exists a machine M_1 that accepts L_1 as well as machine M_1 that accepts L_2. Now, we have to simulate a machine (algorithm) M that accepts the language L such that $L = L_1 \cup L_2$. Then construct machine M which accepts if M_1 accepts. The construction of M is as shown in following Fig. Q.12.1.

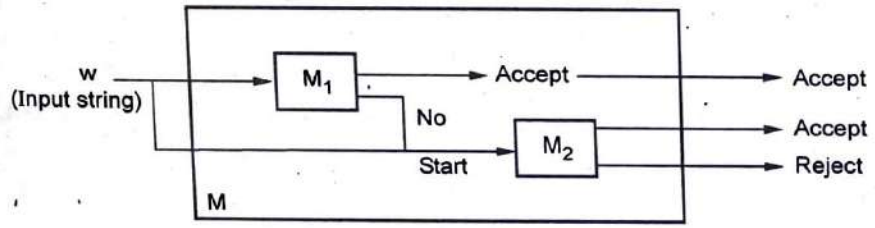
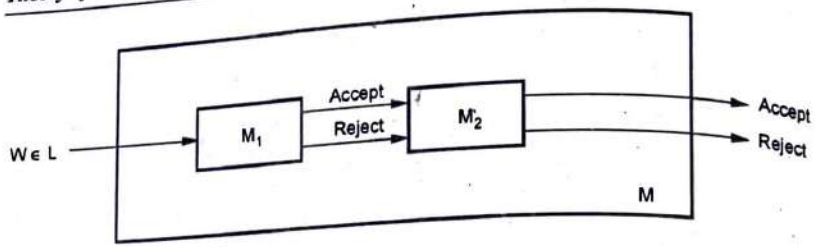


Fig. Q.12.1

If machine M_1 does not accept then M_2 simulates M. That means if M_2 accepts then M accepts, if M_2 rejects M also rejects. Thus M accepts the language $L = L_1 \cup L_2$ which is recursive.

To prove $L_1 \cap L_2$ as a recursive language consider machine M that accepts the language L such that $L = L_1 \cap L_2$.

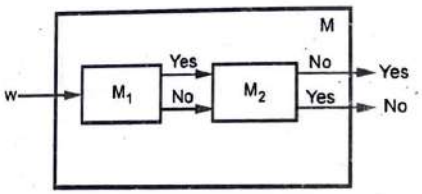
M_1 is a TM which accepts L_1 and M_2 is a TM which accepts L_2. The TM M is simulated which halts on accepting $w \in L$ such that $L = L_1 \cap L_2$. Hence intersection of two recursive languages is also recursive.



Q.13 Prove that if L1 and L2 are two recursive languages and if L is defined as : L = {w| w is in L1 and not in L2 and not in L1}. Prove or disprove that L is recursive. **☞[SPPU : Dec.-08, Marks 8]**

Ans. : Let L1 is a recursive language which is accepted by M1 and L2 is a recursive language which is accepted by M2. Languages L1 and L2 are such that if $w \in L1$ then $w \notin L2$. Similarly if $w \notin L1$ then $w \in L2$.

Find $L = L1 \cup L2$. We can then design a TM M which accepts the language L. Such a TM can be drawn as follows -



As the language L is accepted by TM M, we can say that the language L is recursive language.

Q.14 Prove that the set of languages L over {0, 1} so that neither L nor L' is recursively enumerable in uncountable.

☞[SPPU : Dec.-07, May-09, 10, Marks 6]

Ans. : If there is any language L which is recursively enumerable then there exists a TM which semidecides it (either accepts and halt or loops for ever).

Every TM has description of finite length. Hence number of TM and number of recursively enumerable languages is countably. infinite, because power set of every countable set is countably infinite. Now, for the language L its complement L' is RE. Then for L' also there exists TM which semidecides it. But there are uncountable number of languages. Hence there may be language L and L' which are not recursively

enumerable. And there may be uncountable number of such languages. Hence neither L nor L' is RE in uncountable.

Q.15 Prove that "Let L be a language accepted by deterministic PDA, then the complement of L,can also be accepted by, deterministic PDA. **☞[SPPU : Dec.-19, Marks 4]**

Ans. : • Since every language accepted by a PDA is context-free, it must be the case that no PDA exists that will accept a non-context-free language.

- Context-free languages are not closed under complement. That also means if L is a context free language, then L' may not be context free.
- If the L' is non context free language then it is not accepted by PDA. But if L' is a context free language then only it is accepted by PDA.

6.4 Simple Un-decidable Problem

Q.16 What is post correspondence problem ? Explain with example.

☞[SPPU : Dec.-10, Marks 8, Dec.-19, Marks 6]

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

Example - Refer Q.17.

Q.17 Obtain the solution for the following correspondence system.
A = {ba, ab, a, baa, b}, B = {bab, baa, ba, a, aba}.
The input set {a, b}.

Ans. : To obtain the corresponding system the one sequence can be chosen. Hence we get $w_1w_5w_2w_3w_4w_4w_3w_4 = x_1x_5x_2x_3x_4x_4w_3w_4$. This solution gives a unique string babababaabaaabaa = babababaabaaabaa. Hence solution is 15234434.

Q.18 Obtain the solution for the following system of posts correspondence problem.
A = {100, 0, 1}, B = {1, 100, 00}

Ans. : The solution is 1 3 1 1 3 2 2. The string is

$$A1A3A1A1A1A3A2A2 = 100 + 1 + 100 + 100 + 1 + 0 + 0$$
$$= 1001100100100$$

$$B1B3B1B1B3B2B2 = 1 + 00 + 1 + 1 + 00 + 100 + 100$$
$$= 1001100100100$$

Q.19 Obtain the solution for the following system of posts correspondence problem
A = {ba, abb, bab} B = {bab, bb, abb}.

Ans. : Now to consider 1, 3, 2 the string bababbb from set A and bababbbb from set B thus the two strings obtained are not equal. As we can try various combinations from both the sets to find the unique sequence but we could not get such a sequence. Hence there is no solution for this system.

Q.20 Does PCP with two lists x = (b, bab^3, ba) and y = (b^3, ba, a) have a solution ?

Ans. : Now we have to find out such a sequence that strings formed by x and y are identical. Such a sequence is 2, 1, 1, 3. Hence from x and y list

2	1	1	3		2	1	1	3
bab^3	b	b	ba	=	ba	b^3	b^3	a

which forms bab^3b^3a. Thus PCP has a solution.

6.5 Time and Space Measures

Q.21 Define the terms - time complexity and space complexity.

Ans. : • **Time complexity** of an algorithm means the amount of time taken by an algorithm to run. By computing time complexity we come to know whether the algorithm is slow or fast.

- **Space complexity** of an algorithm means the amount of space (memory) taken by an algorithm. By computing space complexity we can analyze whether an algorithm requires more or less space.

6.6 The Class P and NP

Q.22 Explain with example - Computational complexity.

☞[SPPU : Dec. 2015, End Sem, Marks 2]

Ans. : 1) The computational problems is an infinite collection of instances with a solution for every instance.

2) In computational complexity the solution to the problem can be "yes" or "no" type. Such type of problems are called **decision problem.** The computational problems can be **function problem**. The function problem is a computational problem where single output is expected for every input. The output of such type of problems is more complex than the decision problem.

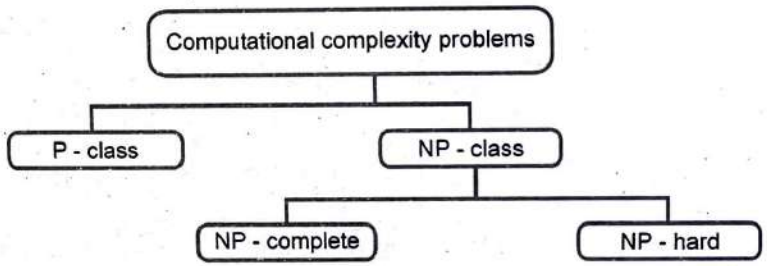


Fig. Q.22.1

3) The computational problems also consists of a class of problems, whose output can be obtained in polynomial time.

Q.23 Write a note on P and NP problems with examples.

☞[SPPU : Dec.-14, End Sem, Marks 4, Dec.-19, Marks 6]

Ans. : • **Definition of P** - Problems that can be solved in polynomial time. ("P" stands for polynomial). The polynomial time is nothing but the time expressed in terms of polnomial.

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

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

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

Q.24 Differentiate between P Class problems and NP class problems.

☞[SPPU : Dec.-15, 19, End Sem, Marks 4]

Ans. :

P Class	NP Class Problem
Problems that can be solved in polynomial time are called P-class problems.	Problems that can be solved in non-deterministically Polynomial time are called NP class problem.
These are simple to solve.	These are complex to solve.
Examples : Searching an element from unordered list, sorting the elements.	**Examples** : Traveling salesperson problem, knapsack problem.

6.7 Examples of Problems in P

Q.25 What is Kruskal's Algorithm ? How can we solve this problem using Turing Machine ?

☞[SPPU : May-16,17, End Sem, Dec.-17, Marks 8]

Ans. : Kruskal's algorithm : In Kruskal's algorithm the minimum weight is obtained. In this algorithm also the circuit should not be formed. Each time the edge of minimum weight has to be selected, from the graph. It is not necessary in this algorithm to have edges of minimum weights to be adjacent.

Example : Find the minimum spanning tree for the following figure using Kruskal's algorithm.

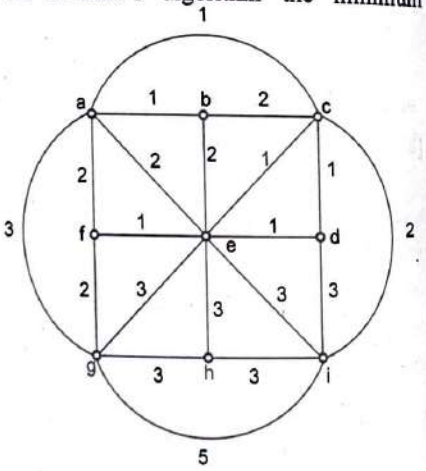


Fig. Q.25.1

In Kruskal's algorithm, we will start with some vertex and will cover all the vertices with minimum weight. The vertices need not be adjacent.

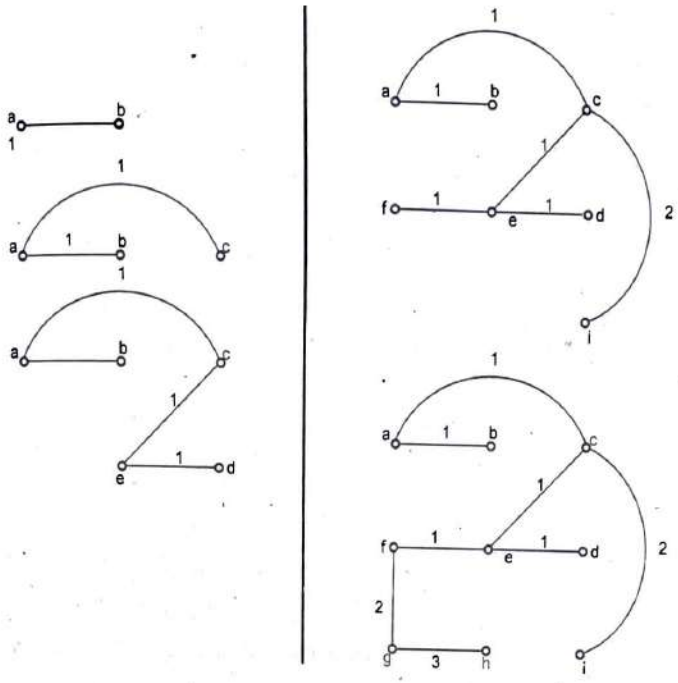


Fig. Q.25.2

6.8 Examples of Problems in NP

Q.26 Justify whether the traveling Salesman problem is a class P or class NP problem?

☞[SPPU : May-15, End Sem, Marks 8]

OR What do you mean by NP problem ? Justify why Traveling Salesman problem is a NP problem?

☞[SPPU : May-17, End Sem, Dec.-17, Marks 8]

Ans. : Travelling Salesman's Problem (TSP) : This problem can be stated as " Given a set of cities and cost to travel between each pair of cities, determine whether there is a path that visits every city once and returns to the first city. Such that the cost travelled is less".

For example :

The tour path will be **a-b-d-e-c-a** and total cost of tour will be **16.**

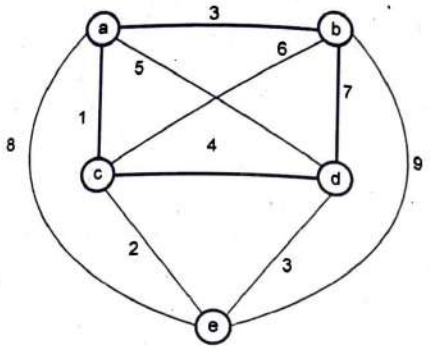


Fig. Q.26.1

This problem is NP problem as there may exist some path with shortest distance between the cities. If you get the solution by applying certain algorithm then Travelling Salesman problem is **NPComplete Problem**. If we get no solution at all by applying an algorithm then the travelling salesman problem belongs to NP **hard class.**

6.9 NP-completeness and NP-hard Problems

Q.27 Why do we need to reduce existing problems to NP Complete problems ? Explain with example.

☞[SPPU : Dec.-14, May-16, May-17, End Sem, Marks 8]

OR Explain in detail, the polynomial time reduction approach for proving that the problem is NP Complete.

☞[SPPU : May-15, End Sem, Marks 8]

OR What do you mean by polynomial time reductions ? Describe any problem in detail that is solvable through polynomial time reductions. ☞[SPPU : Dec.-15, End Sem, Marks 8, Dec.-19, Marks 4]

Ans. : To prove whether particular problem is NP complete or not we use polynomial time reducibility. That means if

$A \xrightarrow{Poly} B$ and $B \xrightarrow{Poly} C$ then $A \xrightarrow{Poly} C$.

The **reduction** is an important task in NP completeness proofs. This can be illustrated by Fig. Q.27.1.

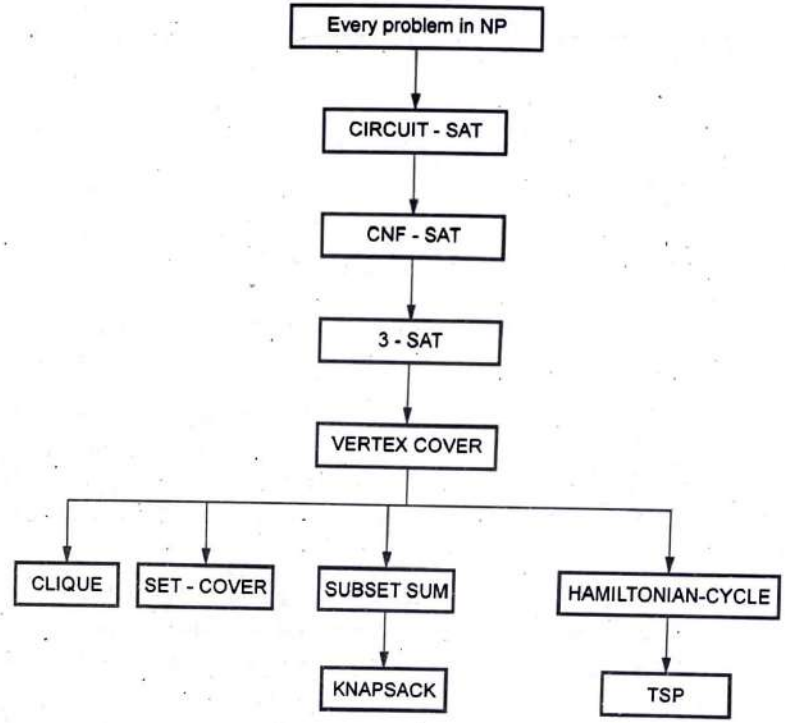


Fig. Q.27.1 Reduction in NP completeness

The reduction can be denoted as $A \leq_T {}^P B$.

Q.28 Explain the relationship between P,NP,NP complete and NP hard problems.

Ans. : • A problem A is NP-hard if there is an NP-complete problem B, such that B is reducible to A in polynomial time. NP-hard problems are as hard as NP-complete problems. NP-hard problem need not be in NP class.

- A NP problem such that, if it is in P, then NP = P. If a (not necessarily NP) problem has this same property then it is called **"NP-hard"**. Thus the class of NP-complete problem is the intersection of the NP and NP-hard classes.

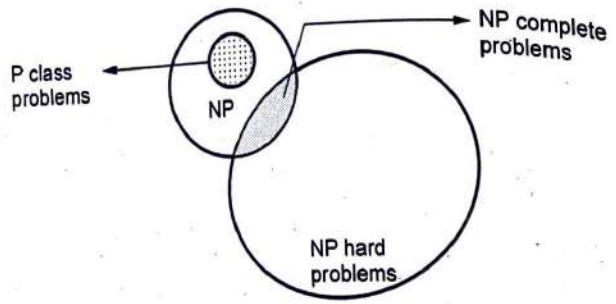


Fig. Q.28.1 Relationship between P, NP, NP-complete and NP-hard problems

- Normally the decision problems are NP-complete but optimization problems are NP-hard. However if problem L_1 is a decision problem and L_2 is optimization problem then it is possible that $L_1 \propto L_2$. For instance the Knapsack decision problem can be Knapsack optimization problem.
- There are some NP-hard problems that are not NP-complete. For example *halting problem*. The halting problem states that : "Is it possible to determine whether an algorithm will ever halt or enter in a loop on certain input ?"

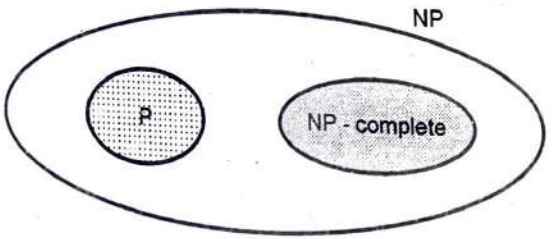


Fig. Q.28.2

Q.29 Give the difference betewen NP Complete and NP hard problem.

Ans. :

Sr. No.	NP hard	NP complete
1.	NP hard problem (Say A) can be solved if and only if there is NP complete problem (say B) can be reducible into A in **polynomial time.**	NP complete problems can be solved by deterministic algorithm in polynomial time.
2.	To solve the NP hard problem, it must be in NP class.	To solve NP complete problem, it must be in both NP and NP hard problems.
3.	It is not a decision problem.	It is a decision problem.
4.	For example - Halting problem, Hamiltonian cycle problem, vertex cover problem.	For example - Determine whether a graph has a Hamiltonian cycle, determine whether a Boolean formula is satisfiable or not, circuit-satisfiability problem.

Q.30 What is 3SAT problem ? Prove that 3-SAT is NP complete.

Ans. : The 3-SAT problem

A 3-SAT problem is a problem which takes a Boolean formula S with each clause having exactly **three literals** and check is S is satisfied or not.

Prove that 3-SAT is NP complete

Proof : The language 3-SAT is a restriction of SAT. We replace each clause C that represents the SAT problem to a function **f** by family of D_C of clauses that represent satisfiability.

For example say

$$C = a \vee b \vee c \vee d \vee e$$

One can simulate this by

$$D_C = (a \vee b \vee x) \wedge (\bar{x} \vee c \vee y) \wedge (\bar{y} \vee d \vee e)$$

where x and y are new variables.

Need to verify :

1) If C is FALSE, then D_C is FALSE; and

2) If C is TRUE, then one can make D_C TRUE.

If **f** is satisfiable then there is assignment where each clause C is TRUE. This can be extended to make D_C TRUE.

Further if **f** is evaluated to FALSE, then some clauses say C' must be FALSE and thus corresponding family $D_{C'}$ evaluates to FALSE.

This conversion process can be done in **polynomial time.** Thus we have shown that SAT reduces to 3-SAT in polynomial time. As we know that SAT is a NP complete problem, so we must say that 3-SAT is also NP complete problem.

END...

SOLVED MODEL QUESTION PAPER (In Sem)

Theory of Computation

T.E. (Computer) Semester - V [As Per 2019 Pattern]

Time : 1 Hour] **[Maximum Marks : 30**

N.B. : ***i) Attempt Q.1 or Q.2, Q.3 or Q.4.***

ii) Neat diagrams must be drawn wherever necessary.

iii) Figures to the right side indicate full marks.

iv) Assume suitable data, if necessary.

Q.1 a) *Define the term Finite Automata(FA).*
(Refer Q.4 of Chapter - 1) **[4]**

b) *Give the definition of regular language.*
(Refer Q.8 of Chapter - 1) **[3]**

c) *Construct a DFA over the alphabets {0, 1} for accepting the strings having number of 1's as multiple of 3.*
(Refer Q.17 of Chapter - 1) **[8]**

OR

Q.2 a) *Define Moore and Mealy machine.*
(Refer Q.38 and Q.39 of Chapter - 1) **[5]**

b) *Differentiate between NFA and DFA.*
(Refer Q.24 of Chapter - 1) **[4]**

c) *Convert following NFA into its equivalent DFA.*

Q \ Σ ->	0	1
-> P	P, Q	R
Q	R	R
R	S	Q
* S	S	S

(Refer Q.32 of Chapter - 1) **[6]**

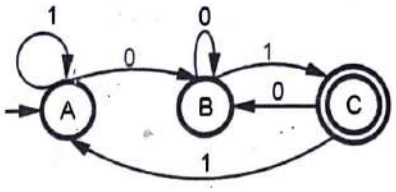
Q.3 a) *Define with example - Regular expression.*
(Refer Q.1 of Chapter - 2) **[3]**

b) *Give the regular expression for the following languages.*
i) The set of strings over the alphabet {a,b} starting with b and ending with odd number of a's or even number of b's.
ii) The set {10, 1010} **(Refer Q.11 of Chapter - 2)** **[4]**

c) *Construct DFA for the regular expression (a + b)* abb.*
(Refer Q.19 of Chapter - 2) **[8]**

OR

Q.4 a) *Convert the following finite automation into its equivalent regular expression using Arden's Theorem.*
(Refer Q.29 of Chapter - 2) **[5]**



b) *Explain the steps in Arden's method.*
(Refer Q.27 of Chapter - 2) **[3]**

c) *Show that L = {ww / w∈ {a, b}} is not regular.*
(Refer Q.35 of Chapter - 2) **[7]**

SOLVED MODEL QUESTION PAPER (End Sem)

Theory of Computation

T.E. (Computer) Semester - V [As Per 2019 Pattern]

Time : 2 $\frac{1}{2}$ Hours] **[Maximum Marks : 70**

N.B. : i) Attempt Q.1 or Q.2, Q.3 or Q.4, Q.5 or Q.6, Q.7 or Q.8.
ii) Neat diagrams must be drawn wherever necessary.
iii) Figures to the right side indicate full marks.
iv) Assume suitable data, if necessary.

Q.1 a) *In each case find context free grammar generating given language*
a) The set of odd length strings in {a, b} with middle symbol a.*
b) The set of even length strings {a, b} with the two middle symbols equal.* **(Refer Q.6 of Chapter - 3)** **[8]**

b) *Give an ambiguous grammar for if then else statement and then re-write an equivalent unambiguous grammar.*
(Refer Q.23 of Chapter - 3) **[10]**

OR

Q.2 a) *State and explain Cock-Younger-Kasami Algorithm with an illustrative example.* **(Refer Q.48 of Chapter - 3)** **[10]**

b) *Eliminate the useless symbols from following grammar.*
$S \rightarrow aA \mid a \mid Bb \mid cC$
$A \rightarrow aB$
$B \rightarrow a \mid Aa$
$C \rightarrow cCD$
$D \rightarrow ddd$ **(Refer Q.30 of Chapter - 3)** **[8]**

Q.3 a) *Design a PDA for accepting a language*
$\{L = a^n b^n \mid n \geq 1\}$ **(Refer Q.2 of Chapter - 4)** **[5]**

b) *Find the grammar G equivalent to the following PDA.*
$M = (\{q_0, q_1\}, \{0,1\}, \{X, Z_0\}, q_0, Z_0, \delta, \phi)$ *and* δ *is*
i) $\delta(q_0, 0, Z_0) = (q_0, XZ_0)$ *ii)* $\delta(q_0, 0, X) = (q_0, XX)$
iii) $\delta(q_0, 1, X) = (q_1, \Delta)$ *iv)* $\delta(q_1, 1, X) = (q_1, \Delta)$
v) $\delta(q_1, \Delta, X) = (q_1, \Delta)$ *vi)* $\delta(q_1, \Delta, Z_0) = (q_1, \Delta)$
(Refer Q.16 of Chapter - 4) **[12]**

OR

Q.4 a) *Construct NPDA that accepts the language generated by* $S = S + S \mid S * S \mid 4$. **(Refer Q.13 of Chapter - 4)** **[3]**

b) *Let* $L = \{d^j b^j c^k / i, j, k \geq$ *and* $i + j = k\}$
a) Find a PDA (which accepts via final state) that recognizes L
b) Find a PDA (which accepts via empty stack) that recognizes L.
(Refer Q.12 of Chapter - 4) **[8]**

c) *Design a PDA for the language*
$L = \{w \mid w \in (a + b)^*$ *and* $n_a(w) > n_b(w)\}$.
(Refer Q.5 of Chapter - 4) **[6]**

Q.5 a) *Give the formal definition of turing machine.* **[4]**
(Refer Q.1 of Chapter - 5)

b) *Construct a TM for* $L=\{a^n\ b^n\ c^n \mid n \geq 1\}$. **[8]**
(Refer Q.4 of Chapter - 5)

c) *Explain the concept of deterministic and non deterministic turing machine.* **(Refer Q.33 of Chapter - 5)** **[6]**

OR

Q.6 a) *Construct a TM for checking the palindrome of a string of odd palindrome for* $\Sigma = \{0,1\}$. **(Refer Q.6 of Chapter - 5)** **[8]**

b) *Construct TM for reversing a binary string on the input tape.* **[10]**
(Refer Q.13 of Chapter - 5)

Q.7 a) *What is post correspondence problem ? Explain with example.* **[6]**
(Refer Q.16 of Chapter - 6)

b) *Write a note on P and NP problems with examples.* **[6]**
(Refer Q.23 of Chapter - 6)

c) *What is 3SAT problem ? Prove that 3-SAT is NP complete.* **[5]**
(Refer Q.30 of Chapter - 6)

OR

Q.8 a) *Prove that if a language L and its complement L' both are RE then L is a recursive language.* **(Refer Q.11 of Chapter - 6)** **[6]**

b) *Show that the language* L_u *is recursively enumerable but not recursive.* **(Refer Q.8 of Chapter - 6)** **[8]**

c) *Write short note on - Chruch- Turing Thesis.* **[3]**
(Refer Q.5 of Chapter - 6)

END...