UNIT-IV: NORMAL FORMS AND TURING MACHINES 4.1-4.75

4

PROPERTIES OF CONTEXT FREE LANGUAGES

4.1 NORMAL FORMS OF CONTEXT-FREE GRAMMARS

If the language is a CFL, then it should have a grammar in some specific form. In order to obtain the form, we need to simplify the context-free grammars.

Every CFL is generated by a CFG in which all productions are of the form

$$A \rightarrow BC$$

or $$A \rightarrow a$$

where A, B, C - variables

a - terminal

This form of CFG is called as Chomsky Normal Form.

In order to find the CNF, we need to perform the following operations.

1. Eliminate useless symbols *i.e.,* symbols or terminals which do not appear in any derivation of a terminal string from the start symbol.
2. Eliminate ε-production which are of the form $A \rightarrow \varepsilon$ for some variable A.
3. Eliminate unit productions which are of the form $A \rightarrow B$ for variables A and B.

4.1.1 ELIMINATION OF USELESS PRODUCTIONS/SYMBOLS

Let G = (V, T, P, S) be a grammar

A symbol X is *useful* if there is a derivation $S \overset{*}{\Rightarrow} \alpha X \beta \overset{*}{\Rightarrow} w$, for some α, β and w where w is in T*. Otherwise it is *useless.*

There are two ways to usefulness

1. Some terminal string must be derivable from X.
2. X must occur in some string derivable from S.

A useful symbol will be both generating and reachable.

Theorem:4.1

Let G = (V, T, P, S) be a CFG with L(G) ≠ ϕ, we can effectively find an equivalent CFG G' = (V', T, P', S) such that for each A in V' there is some w in T* for which $A \overset{*}{\Rightarrow} w$.

Proof:

1. First eliminate non-generating symbols and all productions involving one or more of those symbols. Assume L(G) has atleast one string, so S has not been eliminated.
2. Eliminate all symbols that are not reachable in G'.

Then L(G) = L(G')

Each variable A with production A → w in P clearly belongs to V.

If A → $x_1 x_2 \ldots x_n$ is a production where each x_i is either a terminal or a variable already placed in V'. The set V' can be computed by an iterative algorithm given below.

begin

1. $V_1 := \phi$
2. $V_2 := \{A \mid A \rightarrow w \text{ for some } w \text{ in } T^*\}$
3. while $V_1 \neq V_2$ do

begin

4. $V_1 := V_2$
5. $V_2 := V_1 \cup \{A \mid A \rightarrow \alpha \text{ for some } \alpha \text{ in } (T \cup V_1)^*\}$

end

6. $V' := V_2$

Fig.4.1. Algorithm to find V'

The theorem is proved by induction on the length of the derivation $A \overset{*}{\Rightarrow} w$

Basis:

If the length is one, then A → w is a production and A is added to V_2 in step (2).

Induction:

Let $A \Rightarrow X_1 X_2 \ldots X_n$

$\overset{*}{\Rightarrow}$ w [in k steps]

$w = w_1 w_2 \ldots w_n$ where $X_i \overset{*}{\Rightarrow} w_i$ for $1 \le i \le n$

By the hypothesis, those X_i that are variables are eventually added to V_2.

By using the algorithm, we can find P^1 to be all productions whose symbols are in $V^1 \cup T$.

Surely $G^1 = (V^1, T, P^1, S)$ satisfies the property that 'If A is in V^1', then $A \overset{*}{\Rightarrow} w$ for some w.

As every derivation in G^1 is a derivation of G.

$\therefore L(G^1) \subseteq L(G)$

But if there is some w in L(G) not in $L(G^1)$ then any derivation of w in G must involve a variable in $V - V^1$, or a production in $P - P^1$. But there is a variable in $V - V^1$ that derives a terminal string which is not so actually. So it is a contradiction.

$\therefore L(G^1) = L(G)$

Theorem:4.2

Given a CFG $G = (V, T, P, S)$ the equivalent CFG $G^1 = (V^1, T^1, P^1, S)$ such that for each X in $V^1 \cup T^1$ there exist α and β in $(V^1 \cup T^1)^*$ is given by $S \overset{*}{\Rightarrow} \alpha X \beta$.

Proof:

The set $V^1 \cup T^1$ is constructed by an algorithm 1. Place S in V^1. If A is placed in V^1 and $A \to \alpha_1 \mid \alpha_2 \mid \ldots \mid \alpha_n \mid$ then add all variables of $\alpha_1, \alpha_2, \ldots \alpha_n$ to the set V^1 and all terminals of $\alpha_1, \alpha_2, \ldots \alpha_n$ to T^1. P^1 is the set of productions of P containing the symbols of $V^1 \cup T^1$.

Note

In order to find a grammar with no useless symbols, we need to first apply Theorem 4.1 and Theorem 4.2 next.

Example:4.1 ***Consider the grammar:***

$S \to AB/a$

$A \to b$

Eliminate useless symbols.

☺ ***Solution:***

By applying Theorem 1, first find the useful variables

$V_2 = \{A\}$

Since no string is derivable from B, eliminate B and its production $S \rightarrow AB$.

$\therefore$ we have

$S \rightarrow a$

$A \rightarrow a$

By applying Theorem 2 then,

It is clear that S and *a* appear in sentential forms

$\therefore$ The equivalent grammar is given by

$S \rightarrow a$

$\therefore \quad G = (\{S\}, \{a\}, \{S \rightarrow a\}, S)$

Computing the Generating and Reachable Symbols

- ✓ X is generating if $X \overset{*}{\Rightarrow} w$ for some terminal string w.
- ✓ X is reachable if there is a derivation $S \overset{*}{\Rightarrow} \alpha X \beta$ for some α and β.

A. Generating Symbols

Let G = (V, T, P, S) be a grammar.

To compute the generating symbols of G, we need to follow the below steps.

- ✓ Every symbol of T is generating therefore it generates itself.
- ✓ If $A \rightarrow \alpha$, then A is also generating for $\alpha \in T$ or $\alpha = \varepsilon$.

Example:4.2 ***Consider the grammar***

$E \rightarrow eB|e$

$A \rightarrow a$

Compute generating symbols.

☺ ***Solution:***

The set of generating symbols are

(i) All terminals are generating by itself, so {e, a}

(ii) Since E → eB has no generating symbols for B. So B has to be eliminated. Then E has one more production E → e which is having generating symbol e and like so for A.

∴ The generating symbols are {e, a, E, A}

B. Reachable Symbols

Let G = (V, T, P, S) be a grammar. The following steps are used to find the reachable symbols.

(i) S is reachable [S-Start symbol]

(ii) If A is reachable, then all productions with A in the head, all symbols of those productions are also reachable.

Example:4.3 ***Consider the grammar***

$E \rightarrow AB|a$

$A \rightarrow a$

Find the reachable symbols.

☺ ***Solution:***

Since E is the start symbol, it has to be included. Then E has production bodies AB and a. So E, A, B and a are reachable.

∴ The reachable set = {E, A, B, a}

4.1.2 ELIMINATING ε-PRODUCTIONS

A production which is of the form A → ε is called ε-production. If ε is in L(G), it is not possible to eliminate all ε-productions from G. The same is possible if ε is not in L(G).

For each variable A, if $A \overset{*}{\Rightarrow} \varepsilon$, then A is called as nullable variable. We need to check whether the variable is nullable or not.

If $B \rightarrow C_1 C_2 \ldots C_n$ where each C_i is nullable, then B is nullable.

Theorem:4.3

If L = L(G) for some CFG G = (V, T, P, S) then L-{ε} is L(G¹) for a CFG G¹ with no useless symbols or ε-productions.

Proof:

1. Find the nullable symbols by the method given here →.
2. P^1 is constructed as follows

If $A \to X_1 X_2 \dots X_n$ is in P, then add all productions $A \to \alpha_1 \alpha_2 \dots \alpha_n$ to P^1 where

(i) $\alpha_i = X_i$

(ii) $\alpha_i = \varepsilon$

(iii) not all α_i S are ε

To prove:

$$A \underset{G^1}{\overset{*}{\Rightarrow}} w \text{ if and only if } A \underset{G}{\overset{*}{\Rightarrow}} w \text{ and } w \neq \varepsilon$$

If part:

$A \underset{a}{\overset{i}{\Rightarrow}} w$ and $w \neq \varepsilon$

If $i = 1$, then

$A \to w$ must be a production in P. [$\because w \neq \varepsilon$]

$\therefore A \underset{G^1}{\overset{*}{\Rightarrow}} w$

If $i > 1$, then

$$A \underset{G}{\Rightarrow} X_1 X_2 \dots X_n \underset{G}{\overset{i-1}{\Rightarrow}} w$$

Break w into $w_1 w_2 \dots w_m$ such that for each j, $X_j \overset{*}{\Rightarrow} w_j$ in fewer than i steps if $w_j \neq \varepsilon$, then

$A \to Y_1 Y_2 \dots Y_n$ is a production in P^1

where $Y_j = X_j$

Hence we have

$$A \Rightarrow Y_1 Y_2 \dots Y_n \overset{*}{\Rightarrow} w_1 Y_2 Y_3 \dots Y_n \overset{*}{\Rightarrow} w_1 w_2 Y_3 \dots Y_n$$
$$\overset{*}{\Rightarrow} \dots \overset{*}{\Rightarrow} w_1 w_2 \dots w_n = w$$

$$\therefore \quad A \underset{G''}{\overset{*}{\Rightarrow}} w$$

Only if:

$$A \underset{G''}{\overset{i}{\Rightarrow}} w$$

If i = 1, then there is a production $A \to w$ in G^1. There must be a production $A \to \alpha$ in P whereby deleting certain nullable symbols from α.

$$A \underset{G}{\overset{*}{\Rightarrow}} \alpha \underset{G}{\overset{*}{\Rightarrow}} w$$

which involves deriving ε from the nullable symbols of α.

If i > 1, then,

$$A \underset{G''}{\Rightarrow} X_1 X_2 \ldots X_n \underset{G}{\overset{i-1}{\Rightarrow}} w$$

There must be some $A \to Y$ in P such that $x_1 x_2 \ldots x_n$ is found by striking out some nullable symbols from Y.

Thus $A \underset{G}{\overset{*}{\Rightarrow}} X_1 X_2 \ldots X_n$

write $w_1 w_2 \ldots w_n$ such that for all j

$X_j \underset{G''}{\overset{*}{\Rightarrow}} w_j$ by fewer than i steps

$X_j \underset{G}{\overset{*}{\Rightarrow}} w_j$ if X_j is a variable

Like that if x_j is a terminal then

$w_j = X_j$ then

$X_j \underset{G}{\overset{*}{\Rightarrow}} w_j$

$\therefore \quad A \underset{G''}{\Rightarrow} X_1 X_2 \ldots X_k \underset{G''}{\overset{*}{\Rightarrow}} w$

So w is in $L(G^1)$ if and only if $S \underset{G}{\overset{*}{\Rightarrow}} w$. Thus w is in L(G') if and only if w is in L(G) and $w \neq \varepsilon$.

Example:4.4 ***Consider the grammar***

$S \to AB$

$A \to aAA \mid \varepsilon$

$B \to bBB \mid \varepsilon$

Eliminate ε-productions

☺ ***Solution:***

(i) Find the nullable symbols

Since $A \to \varepsilon$, $B \to \varepsilon$, A and B are nullable symbols.

(ii) $S \rightarrow AB$ has a body consisting of nullable symbols only. Thus S, A, B all are nullable symbols.

(iii) Construct the productions of grammar G^1.

Take $S \rightarrow AB$

There are four ways of arranging this AB with and without ε.

$$S \rightarrow AB \mid A \mid B$$

Take $A \rightarrow aAA$

Again there are four ways of arranging these variables AA

$$A \rightarrow aAA \mid aA \mid aA \mid a$$

Eliminate one aA, we get

$$A \rightarrow aAA \mid aA \mid a$$

Like for the other production

$$B \rightarrow bBB \mid bB \mid aA \mid b$$

∴ The resultant grammar after eliminating ε productions are G':

$$S \rightarrow AB \mid A \mid B$$

$$A \rightarrow aAA \mid aA \mid a$$

$$B \rightarrow bBB \mid bB \mid b$$

4.1.3 ELIMINATING UNIT PRODUCTIONS

A unit production is a production which is of the form $A \rightarrow B$, where both A and B are variables.

Unit pair:

If the sequence of derivation steps are

$$A \Rightarrow B_1 \Rightarrow B_2 \Rightarrow \ldots B_n \Rightarrow \alpha$$

then these unit productions are replaced by a non-unit productions

$B_n \rightarrow \alpha$ directly from A

i.e., $A \rightarrow \alpha$

therefore (A, B) such that $A \overset{*}{\Rightarrow} B$ is called an unit pair.

How to eliminate unit productions:

Given a CFG G = (V, T, P, S) with unit productions, construct a new CFG

Let, $G_1 = (V, T, P_1, S)$

1. Find all the unit pairs of G
2. For each unit pair (A, B), if there is a production $A \rightarrow B$, replace it with $A \rightarrow \alpha$ provided $B \rightarrow \alpha$ a is a production in G.

Example:4.5 ***Given a CFG G = (V, T, P, S) where P is given by***

$E \rightarrow T \mid E + T$

$T \rightarrow F \mid T * F$

$F \rightarrow id$

Eliminate unit productions

☺ ***Solution:***

It contains two unit productions like

$E \rightarrow T$

and $T \rightarrow F$

Replace both the productions by finding the unit pairs. They are

(E, F) [$\because E \Rightarrow T \Rightarrow F \Rightarrow id$]

(T, F) [$\because T \Rightarrow F \Rightarrow id$]

The equivalent grammar without unit production is given by

$E \rightarrow id \mid E + T$

$T \rightarrow id \mid T * F$

$F \rightarrow id$

A CFG G is converted into an equivalent CFG that has no useless symbols, ε-productions or unit productions.

A safe way to obtain an equivalent CFG is by following the safe order

- Eliminate ε-productions
- Eliminate unit productions
- Eliminate useless symbols

4.2 CHOMSKY NORMAL FORM

Theorem:4.5

Any context-free language without ε is generated by a grammar in which all productions are of the form $A \rightarrow BC$ or $A \rightarrow \alpha$ where A, B and C are variables and a is a terminal.

Proof:

To convert a grammar to CNF, check whether the grammar has no ε-productions, unit productions or useless symbols.

Thus find a grammar $G_1 = (V, T, P, S)$ such that P contains no unit productions or ε productions or useless symbols.

If a production is of the form $A \rightarrow a$ *i.e.*, a single symbol on the right, then the production is already in an acceptable form.

If a production is of the form $A \rightarrow x_1 x_2 \ldots x_m$ where $m \geq 2$.

If x_i is a terminal, then introduce a new variable C_a and a production $C_a \rightarrow a$, which is an allowable form.

If the length of the right side part of the production is greater than 2, then we need to do the following tasks.

(i) Arrange the right side part which has length 2 or more containing only variables.

(ii) Break the body of length 3 or more into a cascade of production, each consisting of two variables.

So every production has a body that is either a single terminal or atleast two variables and no terminals.

Then break the productions $A \rightarrow B_1 B_2 \ldots B_k$ for $k \geq 3$ into a group of productions with two variables in each body. Create new variables $D_1 D_2 \ldots D_{m-2}$ and replace $A \rightarrow B_1 B_2 \ldots B_m$ by the set of productions.

$A \rightarrow B_1 D_1, D_1 \rightarrow B_2 D_2 \ldots$

$D_{m-3} \rightarrow B_{m-2} D_{m-2}$

$D_{m-2} \rightarrow B_{m-1} B_m$

Example:4.6 ***Consider the grammar ({S, A, B}, {a, b}, P, S) has the production:***

$S \rightarrow bA \mid aB$

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

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

Convert it into CNF.

☺ ***Solution:***

(i) Find the productions which are already in CNF.

$A \rightarrow a$

$B \rightarrow b$

(ii) Replace the terminals on the right by new variables

$S \rightarrow bA, S \rightarrow aB, A \rightarrow aS, B \rightarrow bS$

Change this as follows

$S \rightarrow bA$

$S \rightarrow C_b A$

$C_b \rightarrow b$

$S \rightarrow aB$

$S \rightarrow C_a B$

$C_a \rightarrow a$

$A \rightarrow aS$

$A \rightarrow C_a S$

$C_a \rightarrow a$

$B \rightarrow bS$

$B \rightarrow C_b S$

$C_b \rightarrow b$

$A \rightarrow bAA$

$A \rightarrow C_b AA$

$B \rightarrow aBB$

$B \rightarrow C_a BB$

(iii) According to CNF theorem, the right hand side body should contain only two variables.

$\mathbf{A \rightarrow C_bAA}$

$A \rightarrow C_b D_1$

$D_1 \rightarrow AA$

$\mathbf{B \rightarrow C_aBB}$

$B \rightarrow C_a D_2$

$D_2 \rightarrow BB$

Thus the resultant productions are

$S \rightarrow C_b A \mid C_a B$

$A \rightarrow C_a S \mid C_b D_1 \mid a$

$B \rightarrow C_b S \mid C_a D_2 \mid b$

$D_1 \rightarrow AA$

$D_2 \rightarrow BB$

$C_a \rightarrow a$

$C_b \rightarrow b$

Example:4.7 ***Convert the CFG into CNF***

$S \rightarrow AB \mid Aa$

$A \rightarrow aAA \mid a$

$B \rightarrow bBB \mid b$

☺ ***Solution:***

The productions which are already in CNF

$S \rightarrow AB$

$A \rightarrow a$

$B \rightarrow b$

Convert the other productions into CNF

$\mathbf{S \rightarrow Aa}$

$S \rightarrow AD_a$

$D_a \rightarrow a$

$A \rightarrow aAA$

$A \rightarrow D_a AA$

$A \rightarrow D_a D_1$

$D_1 \rightarrow AA$

$B \rightarrow bBB$

$B \rightarrow D_b BB$

$B \rightarrow D_b D_2$

$D_2 \rightarrow BB$

Thus the resultant productions are

$S \rightarrow AB \mid AD_a$

$A \rightarrow D_a D_1 \mid a$

$B \rightarrow D_b D_2 \mid b$

$D_a \rightarrow a$

$D_1 \rightarrow AA$

$D_2 \rightarrow BB$

4.3 GREIBACH NORMAL FORM

Theorem:4.5

Every context-free language L without ε can be generated by a grammar for which every production is of the form $A \rightarrow a\alpha$, where A is a variable, *a* is a terminal and α is a string of variables.

Proof:

Let G = (V, T, P, S) be a Chomsky normal form grammar generating the CFL L.

Assume that $V = \{A_1, A_2, \ldots A_m\}$

(*i*) Modify the production which is of the form $A_i \rightarrow A_j \gamma$ where $j < i$

(*a*) If $A_k \rightarrow A_j \gamma$ where $j < k$, then substitute for A_j with its productions.

Suppose $A_j \rightarrow A_k \mid A_L$ then the new set of productions are

$A_k \rightarrow A_k \gamma \mid A_L \gamma$

(*b*) If we get the productions like $A_i \rightarrow A_j \gamma$ where $j = i$ even after substituting with A_j's productions. Then do the following steps

Introduce a new variable B_i,

Then

$B_i \rightarrow \gamma$

$B_i \rightarrow \gamma B_i$

and remove the production $A_i \rightarrow A_i \gamma$

(c) For each production $A_i \rightarrow \beta$ where β does not begin with A_i, then add the production.

$A_i \rightarrow \beta B_i$

After converting all these things, we have only productions of the forms

1. $A_i \rightarrow A_j \gamma, j > i$
2. $A_i \rightarrow a\gamma, a \in T$
3. $B_i \rightarrow \gamma, \gamma \in (V \cup \{B_1, B_2, \ldots B_{i-1}\})^{\alpha}$

This is shown in the following diagrams.

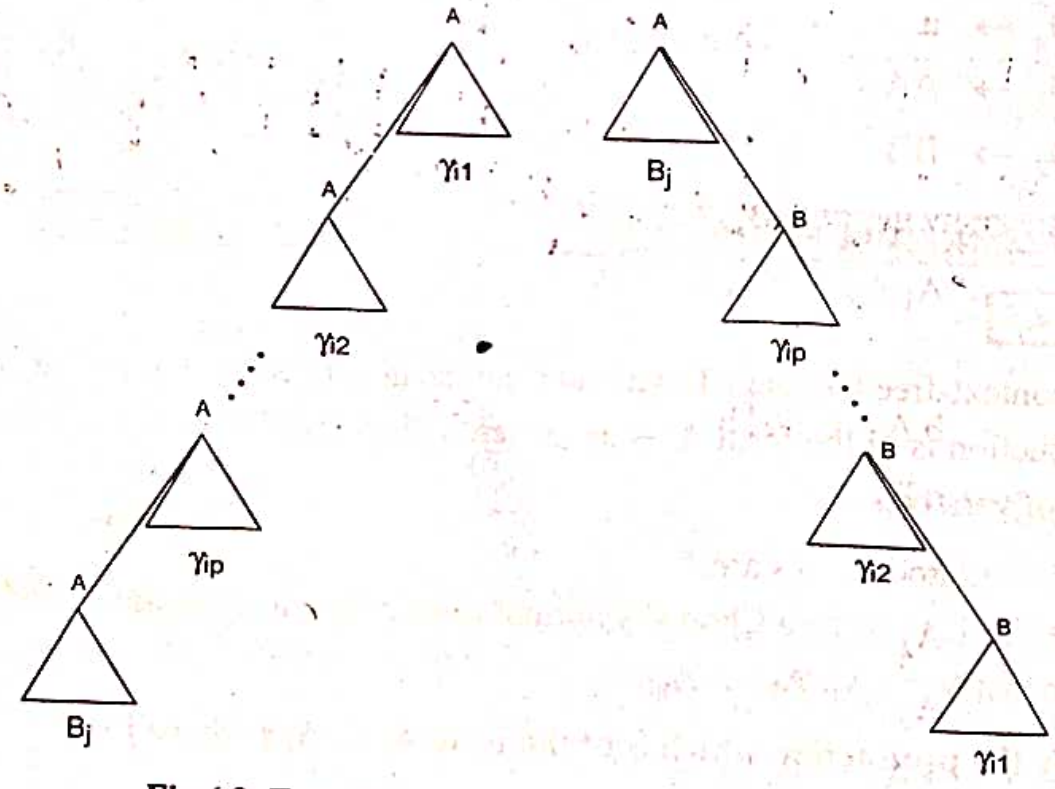


Fig.4.2. Transformation of the production $A_i \rightarrow A_i \gamma$

SOLVED PROBLEMS

Example:4.8 *Convert the following grammar to GNF*

$A_1 \rightarrow A_2 A_3$ — (1)

$A_2 \rightarrow A_3 A_1 \mid b$ — (2)

$A_3 \rightarrow A_1 A_2 \mid a$ — (3)

(or)

$S \rightarrow AB$

$A \rightarrow BS \mid b$

$B \rightarrow SA \mid a$

☺ ***Solution:***

The first two productions (1) and (2) start with terminals or higher numbered variables, we begin with the production.

$A_3 \rightarrow A_1 A_2$

Substitute the string $A_2 A_3$ for A_1

$A_1 \rightarrow A_2 A_3$

$A_2 \rightarrow A_3 A_1 \mid b$

$A_3 \rightarrow A_2 A_3 A_2 \mid a$

Then substitute the string $A_3 A_1$ for A_2

$A_1 \rightarrow A_2 A_3$

$A_2 \rightarrow A_3 A_1 \mid b$

$\underset{i}{\underline{A_3 \rightarrow}} \quad \underset{j}{\underline{A_3}} \quad \underset{\gamma}{\underline{A_1 A_3 A_2}} \mid b A_3 A_2 \mid a$

Here i = j

Introduce a new variable B_3

$B_3 \rightarrow A_1 A_3 A_2$ and $B_3 \rightarrow A_1 A_3 A_2 B_3$

Then

$A_3 \rightarrow b A_3 A_2 B_3 \mid b A_3 A_2$

$A_3 \rightarrow a B_3 \mid a$

The resultant productions are

$A_1 \rightarrow A_2 A_3$ ... (1)

$A_2 \rightarrow A_3 A_1 \mid b$ ... (2)

$A_3 \rightarrow b A_3 A_2 B_3 \mid a B_3 \mid b A_3 A_2 \mid a$ ... (3)

$B_3 \rightarrow A_1 A_3 A_2 \mid A_1 A_3 A_2 B_3$ ... (4)

It is must that all the productions on the right should start with terminals.

Substitute the values for A_3 and A_2 and A_1 in $A_2 A_1 B_3$

$A_2 \rightarrow b A_3 A_2 B_3 A_1 \mid a B_3 A_1 \mid b A_3 A_2 A_1 \mid a A_1 \mid b$

$A_1 \rightarrow b A_3 A_2 B_3 A_1 A_3 \mid a B_3 A_1 A_3 \mid b A_3 A_2 A_1 A_3 \mid a A_1 A_3 \mid b A_3$

$B_3 \rightarrow b\,A_3\,A_2\,B_3\,A_1\,A_3\,A_3\,A_2 \mid a\,B_3\,A_1\,A_3\,A_3\,A_2$

$b\,A_3\,A_2\,A_1\,A_3\,A_3\,A_2 \mid a\,A_1\,A_3\,A_3\,A_2 \mid b\,A_3\,A_3\,A_2 \mid$

$b\,A_3\,A_2\,B_3\,A_1\,A_3\,A_3\,B_3 \mid a\,B_3\,A_1\,A_3\,A_3\,A_2\,B_3 \mid$

$b\,A_3\,A_2\,A_1\,A_3\,A_3\,A_2\,B_3 \mid a\,A_1\,A_3\,A_3\,A_2\,B_3 \mid b\,A_3\,A_3\,A_2\,B_3$

$\therefore$ The final set of productions are

A_1:

$A_1 \rightarrow b\,A_3\,A_2\,B_3\,A_1\,A_3$

$A_1 \rightarrow a\,B_3\,A_1\,A_3$

$A_1 \rightarrow b\,A_3\,A_2\,A_1\,A_3$

$A_1 \rightarrow a\,A_1\,A_3$

$A_1 \rightarrow b\,A_3$

A_2:

$A_2 \rightarrow b\,A_3\,A_2\,B_3\,A_1$

$A_2 \rightarrow a\,B_3\,A_1$

$A_2 \rightarrow b\,A_3\,A_2\,A_1$

$A_2 \rightarrow a\,A_1$

$A_2 \rightarrow b$

A_3:

$A_3 \rightarrow b\,A_3\,A_2\,B_3$

$A_3 \rightarrow a\,B_3$

$A_3 \rightarrow b\,A_3\,A_2$

$A_3 \rightarrow a$

B_3

$B_3 \rightarrow b\,A_3\,A_2\,B_3\,A_1\,A_3\,A_3\,A_2$

$B_3 \rightarrow a\,B_3\,A_1\,A_3\,A_3\,A_2$

$B_3 \rightarrow b\,A_3\,A_2\,A_1\,A_3\,A_3\,A_2$

$B_3 \rightarrow a\,A_1\,A_3\,A_3\,A_2$

$B_3 \rightarrow b\,A_3\,A_3\,A_2$

$B_3 \rightarrow b A_3 A_2 B_3 A_1 A_3 A_3 A_2 B_3$

$B_3 \rightarrow a B_3 A_1 A_3 A_3 A_2 B_3$

$B_3 \rightarrow b A_3 A_2 A_1 A_3 A_3 A_2 B_3$

$B_3 \rightarrow a A_1 A_3 A_3 A_2 B_3$

$B_3 \rightarrow b A_3 A_3 A_2 B_3$

Example:4.9 ***Convert the following productions into GNF.***

$S \rightarrow AA \mid a$

$A \rightarrow SS \mid b$

☺ ***Solution:***

The given productions are in CNF

Rename S and A as A_1 and A_2 respectively

$A_1 \rightarrow A_2 A_2 \mid a$ ... (1)

$A_2 \rightarrow A_1 A_1 \mid b$ ... (2)

Step 1:

In (1), j > k no need to change the production

In (2), j < k, substitute A_1 productions in (2)

$A_2 \rightarrow A_2 A_2 A_1 \mid a A_1 \mid b$

The resultant productions are

$A_1 \rightarrow A_2 A_2 \mid a$

$A_2 \rightarrow A_2 A_2 A_1 \mid a A_1 \mid b$

Step 2:

Take $A_2 \rightarrow \dfrac{A_2}{A_k} \dfrac{A_2 A_1}{\alpha}$ $[\because A_k \rightarrow A_k \alpha]$

Introduce a new variable B_2

$B_2 \rightarrow A_2 A_1 \mid A_2 A_1 B_2$

and $A_2 \rightarrow a A_1 \mid b$ has been changed to

$A_2 \rightarrow a A_1 \mid a A_1 B_2 \mid b \mid 1 B_2$

The resultant productions are

$A_1 \rightarrow A_2 A_2 \mid a$ ... (1)

and $A_2 \rightarrow a A_1 \mid a A_1 B_2 \mid b \mid 1 B_2$ ... (2)

$B_2 \rightarrow A_2 A_1 \mid A_2 A_1 B_2$ ... (3)

Step 3:

Substitute A_2 in (1)

$$A_1 \rightarrow A_2 A_2$$

$$\Rightarrow \quad A_1 \rightarrow aA_1 A_2 \mid a A_1 B_2 A_2 \mid b A_2 \mid b B_2 A_2 \mid a$$

Substitute A_2 in (3)

$$B_2 \rightarrow a A_1 A_1 \mid a A_1 A_1 B_2 \mid a A_1 B_2 A_1 \mid b A_1 \mid b A_1 B_2 \mid a A_1 B_2 A_1 B_2$$

$\therefore$ The equivalent GNF productions are given by

$$A_1 \rightarrow a A_1 A_2 \mid a A_1 B_2 A_2 \mid b A_2 \mid b B_2 A_2 \mid a$$

$$A_2 \rightarrow a A_1 \mid a A_1 B_2 \mid b B_2 \mid b$$

$$A_3 \rightarrow a A_1 A_1 \mid a A_1 A_1 B_2 \mid a A_1 B_2 A_1 \mid a A_1 B_2 A_1 B_2 \mid b B_2 A_1 \mid b B_2 A_1 B_2 \mid b A_1 \mid b A_1 B_2$$

Solved Problems related to CNF and GNF

Example:4.10 ***Find a grammar in Chomsky normal form equivalent to***

$$S \rightarrow aAbB$$

$$A \rightarrow aA \mid a$$

$$B \rightarrow bB \mid b$$

☺ ***Solution:***

The productions which are already in CNF

$$A \rightarrow a$$

$$B \rightarrow b$$

Apply the CNF rule to other productions

$$S \rightarrow aAbB$$

$$S \rightarrow C_a A C_b B \text{ where } C_a \rightarrow a, C_b \rightarrow b$$

$$S \rightarrow C_a D_1$$

$$D_1 \rightarrow A D_2$$

$$D_2 \rightarrow C_b B$$

Apply the rule $A \rightarrow aA$

$$A \rightarrow C_a A \text{ where } C_a \rightarrow a$$

Then $B \rightarrow bB$

$$B \rightarrow C_b B \text{ where } C_b \rightarrow b$$

Then the resultant productions are

$S \rightarrow C_a D_1$

$D_1 \rightarrow A D_2$ $\quad C_a \rightarrow a$

$D_2 \rightarrow C_b B$ $\quad C_b \rightarrow b$

$A \rightarrow C_a A$ $\quad A \rightarrow a$

$B \rightarrow C_b B$ $\quad B \rightarrow b$

Example:4.11 ***Find a grammar in Chomsky normal form equivalent to***

$S \rightarrow aAD$

$A \rightarrow aB \mid bAB$

$B \rightarrow b$

$D \rightarrow d$

☺ ***Solution:***

The productions which are already in CNF is

$B \rightarrow b$

$D \rightarrow d$

Then apply the CNF rule to other productions

$S \rightarrow aAD$

$S \rightarrow C_a AD$ $\quad Ca \rightarrow a$

$S \rightarrow C_a D_1$

$D_1 \rightarrow A D$

$A \rightarrow aB$

$A \rightarrow C_a B$ $\quad C_a \rightarrow a$

$A \rightarrow bAB$

$A \rightarrow C_b AB$

$A \rightarrow C_b D_2$ $\quad C_b \rightarrow b$

$D_2 \rightarrow A B$

∴ The resultant productions are

$S \rightarrow C_a D_1$

$D_1 \rightarrow A D$ $\quad B \rightarrow b$

$A \rightarrow C_a B$ $\quad D \rightarrow d$

$A \rightarrow C_b D_2$ $\quad C_a \rightarrow a$

$D_2 \rightarrow A B$ $\quad C_b \rightarrow b$

Example:4.12 ***Convert the following grammar into CNF.***

$S \rightarrow aAa \mid bBb \mid \varepsilon$

$A \rightarrow C \mid a$

$B \rightarrow C \mid b$

$C \rightarrow CDE \mid \varepsilon$

$D \rightarrow A \mid B \mid ab$

☺ ***Solution:***

(i) First eliminate ε-production

Nullable variables are = {S, C}

If there is a production, $A \rightarrow C_1\ C_2 \ldots C_k$ such that every C_i is variables and nullable, then A is nullable.

Thus, $A \rightarrow C$, $B \rightarrow C$ and then $D \rightarrow B$

$\therefore$ A, B, D are nullable variables.

$\therefore$ Nullable variables = {S, A, B, C, D}

After eliminating the ε-productions, the resultant productions are

$S \rightarrow aAa \mid aa \mid sbBb \mid bb$

$A \rightarrow C \mid a$

$B \rightarrow C \mid b$

$C \rightarrow CE \mid DE \mid CDE \mid E$

$D \rightarrow A \mid B \mid ab$

(ii) Next eliminate unit productions

$A \rightarrow C \mid a$

Replace C by its production

$A \rightarrow a \mid CE \mid DE \mid CDE$

$B \rightarrow C \mid b$

$B \rightarrow b \mid CE \mid DE \mid CDE$

$C \rightarrow CE \mid DE \mid CDE \mid E$

$C \rightarrow CE \mid DE \mid CDE$

$D \rightarrow A \mid B \mid ab$

$D \rightarrow a \mid b \mid ab [\because A \rightarrow a, B \rightarrow b]$

Thus the reduced grammar is

$S \rightarrow aAa \mid aa \mid bBb \mid bb$

$A \rightarrow a \mid CE \mid DE \mid CDE$

$B \rightarrow b \mid CE \mid DE \mid CDE$

$C \rightarrow CDE \mid CE \mid DE$

$D \rightarrow a \mid b \mid ab$

(iii) Eliminate useless productions

Since C does not produce a terminal string, C is a useless variable. The variable E also does not produce any terminal string. Eliminate C and E.

$\therefore S \rightarrow aAa \mid aa \mid bBb \mid bb$

$A \rightarrow a$

$B \rightarrow b$

$D \rightarrow a \mid b \mid ab$

Since the start variables S does not have any variable related to D. So eliminate the variable D.

The reduced grammar is

$S \rightarrow aAa \mid aa \mid bBb \mid bb$

$A \rightarrow a$

$B \rightarrow b$

Apply CNF rule to this grammar

$\therefore$ The resultant productions are

$S \rightarrow C_a D_1$

$D_1 \rightarrow A C_a \qquad C_a \rightarrow a$

$S \rightarrow C_b D_2 \qquad C_b \rightarrow b$

$D_2 \rightarrow B C_b$

$S \rightarrow C_a C_a \qquad A \rightarrow a$

$S \rightarrow C_b C_b \qquad B \rightarrow b$

Example:4.13 ***Find a context free grammar with no useless symbols equivalent to***

$S \rightarrow AB \mid CA$

$B \rightarrow BC \mid AB$

$A \rightarrow a$

$C \rightarrow aB \mid b$

☺ ***Solution:***

By using the algorithm given for eliminating useless symbols.

V = {S, A, B, C}

T = {a, b}

	T ∪ old V	old V	New V
1.		ϕ	{A, C}
2.	{a, b, A, C}	{A, C}	{A, C, S}
3.	{a, b, A, C, S}	{A, C, S}	{A, C, S}

The useless productions are

S → AB

B → BC

B → AB

C → aB

After eliminating the useless productions, the useful productions are

S → CA

A → a

C → b

Example:4.14 ***Convert the grammar into CNF***

$S \rightarrow aSaA \mid A$

$A \rightarrow abA \mid b$

☺ ***Solution:***

There is no ε- productions, the unit production in this grammar is

S → A

Replace this by its productions

$S \rightarrow aSaA \mid abA \mid b$

Then the grammar is

$S \rightarrow aSaA \mid abA \mid b$

$A \rightarrow abA \mid b$

There is no useless productions. Apply CNF to this grammar.

$S \rightarrow aSaA$

$\rightarrow C_a S C_a A$

$S \rightarrow C_a D_1$

$D_1 \rightarrow S D_2$

$D_2 \rightarrow C_a A$

$S \rightarrow abA$

$S \rightarrow C_a C_b A$

$S \rightarrow C_a D_3$

$D_3 \rightarrow C_b A$

$A \rightarrow abA$

$\rightarrow C_a C_b A$

$A \rightarrow C_a D_3$

$D_3 \rightarrow C_b A$

Then the resultant productions are

$S \rightarrow C_a D_1$

$D_1 \rightarrow S D_2$

$D_2 \rightarrow C_a A$

$S \rightarrow C_a D_3$

$D_3 \rightarrow C_b A$

$S \rightarrow b$

$A \rightarrow C_a D_3$

$D_3 \rightarrow C_b A$

$A \rightarrow b$

Example:4.15 ***Convert the grammar into GNF.***

$S \rightarrow aSb$

$S \rightarrow ab$

☺ ***Solution:***

The given grammar is free from ε-productions, unit productions. Then convert it into CNF.

$S \rightarrow C_a S C_b \quad C_a \rightarrow a$

$S \rightarrow C_a C_b \quad C_b \rightarrow b$

Apply GNF rule to this grammar

Replace

S by A_1

C_a by A_2

C_b by A_3

The above grammar is written as

$A_1 \rightarrow A_2 A_1 A_3$ ... (1)

$A_2 \rightarrow a$ ... (2)

$A_1 \rightarrow A_2 A_3$ ... (3)

$A_3 \rightarrow b$ ... (4)

All the above productions are in the prescribed form of GNF.

$[A_i \rightarrow A_j$ where $i < j]$

In order to change the production (1) and (3) GNF (*i.e.*, $A \rightarrow \alpha\alpha$ or $A \rightarrow a$), replace A_2 by its productions.

∴ The resultant productions are

$A_1 \rightarrow a A_1 A_3$

$A_1 \rightarrow a A_3$

$A_2 \rightarrow a$

$A_3 \rightarrow b$

4.4 PUMPING LEMMA FOR CFL

Lemma

Let L be any CFL. Then there exists a constant n, depending only on L such that if z is in L and $|z| \geq n$, then we can write z = uvwxy such that

(i) $|vx| \geq 1$

(ii) $|vwx| \leq n$

(iii) for all $i \geq 0$, $uv^i wx^i y \in L$

Proof:

The first step is to find a Chomsky-Normal form grammar G for L.

Let G = (V, T, P, S) such that L(G) = L – {ε}. There are M variables in G. Choose $n = 2^M$. Suppose z in L is of length atleast n. By theorem, any parse tree whose longest path is of length M or less must have a yield of length $2^{M-1} = n/2$ or less. Such a parse tree cannot have yield z, because z is too long. Thus any parse tree with yield z has a path of length atleast M + 1. But such a path has atleast M + 2 vertices. Thus there must be two vertices v_1 and v_2 on the path. It is possible to divide the tree as in Fig.4.3, string w is the yield of the subtree whose root is A_j.

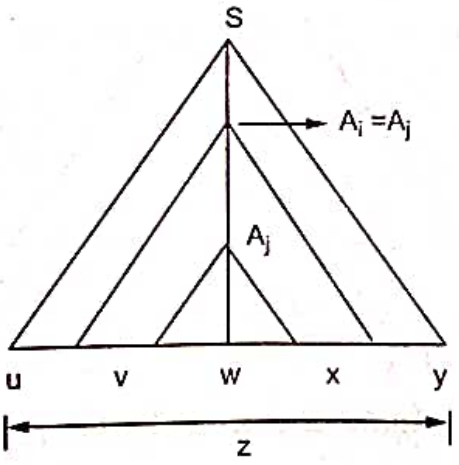


Fig.4.3.Dividing the string w

For the yield w, at A_j string v and x are the strings to the left and right. There will be no unit productions in this grammar, so v and x could not both be ε, but one could be. Then at A_i, strings u and y are to its left and right.

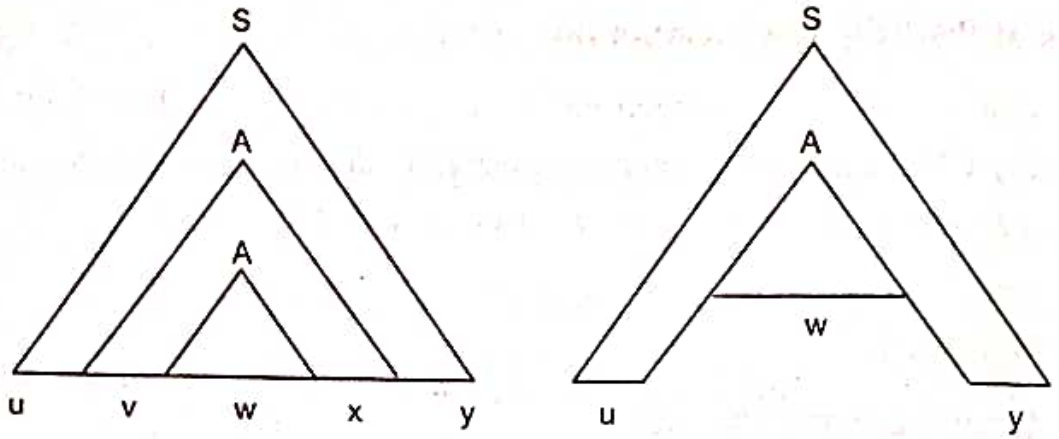


Fig.4.4.

Suppose $A_i = A_j = A$ construct a new parse tree Fig.4.4 (*a*) replacing the root A_i and A_j by A, which has yield w. the resultant tree is given in Fig.4.4(*b*), which has yield uwy and corresponds to the case i = 0 in the pattern of strings $uv^i wx^i y$.

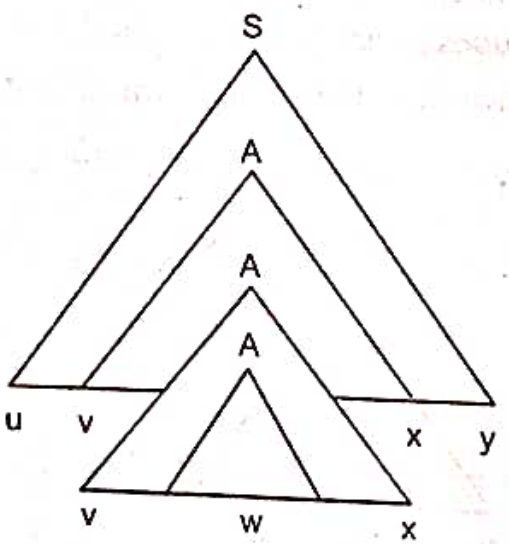


Fig.4.5. Replacing A_j and A_i

Suppose A_j is replaced by the subtree A_i.

The yield of this tree is uv^2wx^2y. Likewise we can expand the tree for any number of times. Thus there are parse trees in G for all strings of the form $uv^i wx^i y$.

Then $|vwx| \leq n$, for this chose A_i to be the bottom of the tree *i.e.*, $K - i \leq M$. Thus the longest path in the subtree rooted at A_i is no greater than M + 1. According to the theorem, the subtree rooted at A_i has a yield whose length is not greater than $2^M = n$.

Thus the lemma is proved.

Application of the Pumping Lemma for CFL's

To show that a language L is not context-free, we need to use the following steps:

(i) Assume L is context-free. Let n be the natural number obtained by using the pumping lemma.

(ii) Choose z Σ L so that | z | ≥ n. Write z = uvwxy using the lemma.

(iii) Find a suitable integer i such that $uv^i wx^i y \notin L$. This is a contradiction and so L is not context free.

4.4.1 PROBLEMS BASED ON PUMPING LEMMA

Example:4.16 ***Show that L = {a^n b^n c^n | n ≥ 1} is not context free.***

☺ ***Solution:***

(*i*) Assume L is context-free. By pumping lemma let *n* be the natural number.

(*ii*) Let $z = a^n b^n c^n$

$|z| = \{ a^n b^n c^n \} = 3n > n.$

Then z = uvwxy where | vx | ≥ 1.

(*iii*) uvwxy = $a^n b^n c^n$

As 1| ≤ | vx | ≤ n, v or x cannot contain all the three symbols a, b, c.

So v or x is of the form.

Case (*i*)

1. a's
2. b's
3. c's

Case (*ii*)

1. $a^i b^j$ for some i, j
2. $b^i c^j$ for some i, j

Case (*i*)

When both v and x are formed by the repetition of a single symbol,

For eg:

$u = a^i$

$v = b^j$

Then $uv^i wx^i y$

Let i = 0, $z = uv^0 wx^0 y$

$= uwy$

$= a^j (wy)$

uwy contains atmost $a^j b^{n-j} c^n$

which is not of the form $uv^i wx^i y$.

$\therefore uwy \notin L$

Likewise for $v = a^j, c^i$

Case (*ii*)

V or x has $a^j b^j$

If i = 0, $uv^i wx^i y = uwy$

Uwy contains at most $a^{n-i} b^n c^n$ which is not of the form $uv^i wx^i y \notin L$

Like so for $b^j c^j$

Therefore L is not context-free.

Example:4.17 ***Show that the language $L = \{a^P \mid P$ is a prime$\}$ is not a context-free language.***

☺ ***Solution:***

1. Assume L is context-free (contradiction). Let n be the natural number.
2. Let $z = a^p$ where $p > n$.
3. z = uvwxy

By theorem, $uv^i wx^i y \notin L$ and $|vx| \geq 1$

Let $|uwy| = q$ a prime number

And $|vx| = S$

Then, $|uv^q wx^q y| = |uwy| + q|vx|$

$= q + qS = q(1 + S)$

But q(1 + S) cannot be prime.

$\therefore uv^q wx^q y \notin L$ (contradiction)

So L is not context-free.

Example:4.18 ***Prove that $L = \{0^{2^i} \mid i \geq 1\}$ is not context-free.***

☺ ***Solution:***

1. Assume L is context free. Let n be the natural number.
2. Let z = uvwxy and $|vwx| \leq n$ and $|vx| \neq 0$

3. Take $v = 0^r$ and $x = 0^s$ and $x + s \geq 1$

$|uv^i wx^i y| = 2^n$

and $v = 0^r$ then $v^i = 0^{ir}$

$x = 0^s$ then $x^i = 0^{is}$

Take $|uwy| = 2^j$

$\therefore 2^n = 2^j + i(r + s)$

If $i = 1, r = 1, S = 0$ then

$2^j + 1$ cannot be written as a power of two.

It's a contradiction.

$\therefore$ The given language L is not context free.

Example:4.19 ***Show that the language $L = \{a^i b^j c^i d^j \mid i, j \geq 1\}$ is not context free.***

☺ ***Solution:***

(*i*) Assume that L is context-free.

(*ii*) By pumping lemma, there exists a constant n such that there is a string z of length atleast n such that $z = uvwxy$, where

(*a*) $vx \neq \varepsilon$, (*b*) $|vwx| \leq n$, (*c*) $uv^i wx^i y \Sigma L$ for all $i \geq 0$

(*iii*) Consider $z = a^n b^n c^n d^n$ where $|z| \geq n$

Since $|vwx| \leq n$ contains maximum two symbols only, in the given order.

(*iv*) Possible cases are vwx, contains

(*a*) a's and b's

(*b*) b's and c's

(*c*) c's and d's

(*d*) only a's

(*e*) only b's

(*f*) only c's

(*g*) only d's

Let vwx contains only a's and b's (*i.e.,* case 1)

Since $vx \neq \varepsilon$, vx contains atleast one 'a' or one 'b'. Thus uwy does not contain atleast one 'a' or one 'b'. uwy contains atmost n or 3 different symbols and less than n of the 4^{th} symbol.

(v) But there should be equal number of a's and c's and equal number of b's and d's. So uwy cannot be in the given language L.

∴ L is not context-free

This is same for the other cases.

4.5 CLOSURE PROPERTIES OF CFL

The operations are useful not only in constructing or proving that certain language are context free, but also in providing certain languages not to be context free. Here we see some of the operations like substitutions, homomorphisms, inverse homomorphism, inverse homomorphism, intersection, difference, etc.

4.5.1 SUBSTITUTIONS

Let Σ be a alphabet. For each symbol a in Σ, choose a language L_a and S(a) defines the substitution function.

If $w = a_1, a_2 \ldots a_n$ is a string in Σ*, then s(w) is the language of all strings $x_1, x_2, \ldots x_n$ such that string x_i is in the language $S(a_i)$ for i = 1, 2, ...n. S(L) is the union of S(w) for all strings w in L.

Example:

Let $S(0) = \{0^n1^n\}\ n \geq 1\}$ then

$S(1) = \{00, 11\}$

It constrains only two strings 00 and 11. But S(0) contains one or more a's followed by an equal number of b's.

Let w = 10, then S(w) is the concatenation of S(1) and S(0).

∴ $S(w) = S(1).\ S(0) = 000^n1^n, 110^n1^n$.

Theorem:4.6

If L is a context-free langrage over alphabet Σ, and S is a substituting on Σ such that S(a) is a CFL for each a in S, then S(L) is a CFL.

Proof:

The idea here is that for a CFG, replace each terminal a by the start symbol for language S(a). The result is a single CFG that generates S(L).

Let G = (V, S, P, S) be a grammar for L

and G_a = (V_a, T_a, P_a, S_a) be a grammar for each a in Σ.

construct a grammar $G^1 = (V^1, T^1, P^1, S)$ for S(L).

where ✓ V^1 is he union of V and V_a [for all a in Σ]

- ✓ T^1 is the union of all T_a
- ✓ P^1 is given by
  - P_a for a in Σ
  - P where each terminal is replaced by S_a.

Thus all parse trees in grammar G^1 start out with parse trees in G but all nodes have labels that are S_a for some a in Σ. Then generation of each such node produces a parse tree of G_a whose yield belongs to S(a).

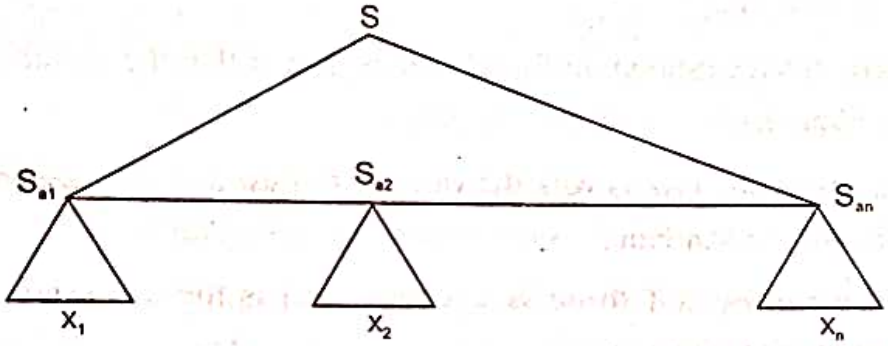


Fig.4.6.Purse tree of G1 due to substitution

To prove: A string w is in $L(G^1)$is and only if w is in S(L).

If part: w is in S(L).

Let x = a_1 a_2 . . . an in L and strings xi in $S(a_i)$ for i = 1, 2, ... n such that w = x_1 x_2 . . .x_n.

The portion of G^1 that comes from the productions of G with S_a substituted for each a will generate a string x. this string is S_{a1} S_{a2} . . . S_{an}. This part of the derivation of w is shown below.

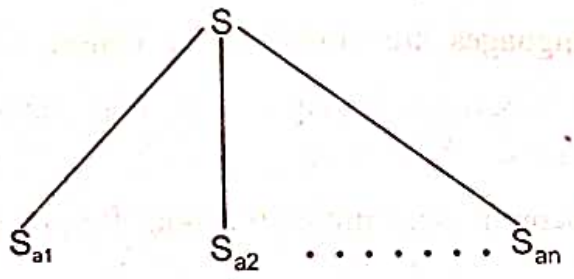


Fig.4.7. Derivation of w but Sa in place of a

The derivation of x_i from S_{ai} is also a derivation in G^1. The yield of G^1 is x_1 x_2 . . . x_n = w. So it is true that w is in $L(G^1)$.

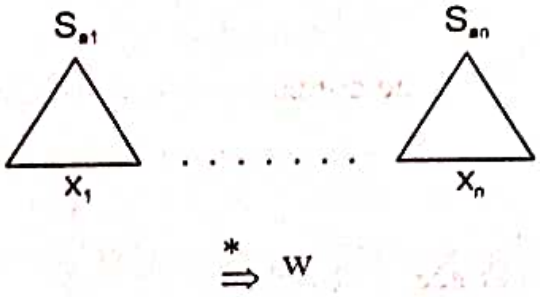


Fig.4.8.

Only-if part: w is in $L(G^1)$

The parse tree for w is shown in fig.4.6. the reason is that the variables of each of the grammars G and G_a for a in Σ are disjoint.

Thus, the top of the tree starts with the variable S must use only the productions of G till symbol S_a, and below that S_a, only productions of grammar G_a may be used.

So if there is a parse tree T for w, it is possible to identify a string $a_1\ a_2 \ldots a_n$ in L(G) and strings xi in language $S(a_i)$ where

(i) $w = x_1\ x_2 \ldots x_n$

(ii) $S = Sa_1\ Sa_2 \ldots Sa_n$

But the string $x_1\ x_2 \ldots x_n$ is in S(L) because it is formed by substituting string x_i for each of the a_i's. thuse w is in S(L) is true.

Thus the theorem is proved.

4.5.2 APPLICATIONS OF THE SUBSTITUTION THEOREM

Theorem:4.7

The context free languages are closed under Union, Concatenation, Closure, Homomorphism.

Proof:

Apply the pervious theorem (substitution theorem) for proving all these operation.

1. Union:

Let L_1 and L_2 be two CFL's. then the union of L_1 and L_2 is, $S(L) = L_1 \cup L_2$ where L is the language {1, 2} and S is the substitution given by $S(1) = L_1$ and $S(2) = L_2$.

2. Concatenation:

Let L_1 and L_2 be CFL's. Then the concatenation of two languages L_1 and L_2 is, $S(L) = L_1 \cdot L2$ where L is the language {12} and S is the substitution $S(1) = L_1$ and $S(2) = L_1$.

3. Closure and positive closure:

If L_1 is a CFL's, then the closure is given by

$S(L) = L_1^*$ where $S(1) = L_1$

The positive closure is given by

$S(L) = L_1^+$ where $L = \{1\}^+$.

4. Homomorphism:

Let L be a CFL over the alphabet Σ and h is a homomorphism on Σ.

Let S be the substitution that replaces each symbol a in Σ by the language consisting of one string h(a).

$S(a) = \{h(a)\}$ for all in Σ.

Thus $h(L) = S(L)$.

4.5.3 REVERSAL

Theorem:4.8

The CFL's are closed under reversal

(or)

If L is a CFL, then so is LR.

Proof:

Let G = (V, T, P, S) be a CFG and L(G) be its grammar.

Then $G^R = (V, T, P^R, S)$

where

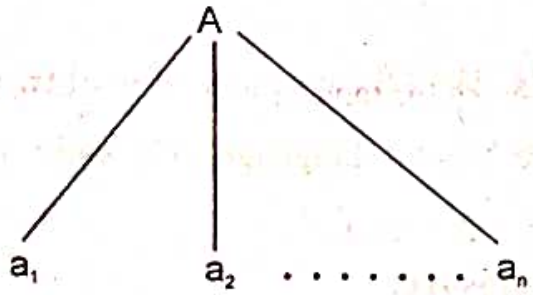
P^R is the reverse of each production in P

Construct P^R where

If $A \rightarrow \alpha$ is a production of G, then

$A \rightarrow \alpha^R$ is a production of G^R

Let $A \rightarrow a_1 a_2 \ldots$ an be a production in G.



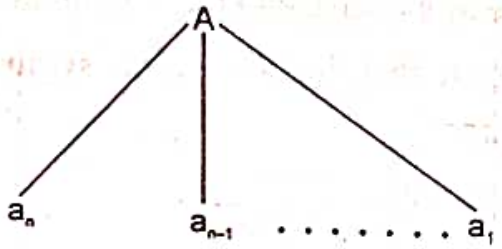
$\overset{*}{\Rightarrow} a_1 a_2 \ldots a_n$

Fig.4.9. Parse tree of grammar G_1

Then

$$A \rightarrow \alpha^R$$

$$\rightarrow a_n a_{n-1} \ldots a_2$$



$\overset{*}{\Rightarrow} a_n a_{n-1} \ldots a_2$

Fig.4.10. Parse tree of grammar G^R

Since all the sentential forms of GR are reverses of sentential forms of G and vice-versa.

∴ The CFL's are closed under reversal.

4.5.4 INTERSECTION

Theorem: 4.9

The CFL's are not closed under intersection.

Let L_1 and L_2 be CFL's. then $L_1 \cap L_2$ is the language I(L) where it satisfies both the properties of L_1 and L_2 which is not possible in CFL.

Example

$$\text{Let } L_1 = \{a^m b^n \mid m \geq 1, n \geq 1\}$$

$$L_2 = \{a^n b^m \mid n \geq 14, m \geq 1\}$$

$L = L_1 \cap L_2$ is not possible. Because L_1 requires that there be M number of a's and n number of b's but L_2 requires n number of a's and M number of b's.

So CFL's are not closed under intersection.

Theorem:4.10

If L is a CFL and R is a regular language then $L \cap R$ is a CFL.

Proof:

Construct a pushdown automaton which contains both the PDA and the Finite Automaton.

Let $P = (Q_P, \Sigma, \vdash, \delta_p, q_p, z_0, F_p)$ be a PDA that accepts L by reaching its final state.

Let $A = (Q_A, \Sigma, \delta_A, q_A, F_A)$ be a DFA for R.

Construct a new PDA produced due to the intersection of P and A.

$$P^1 = (Q_p \times Q_A, \Sigma, \vdash, (q_p, q_A)\, z_0, F_p \times F_A)$$

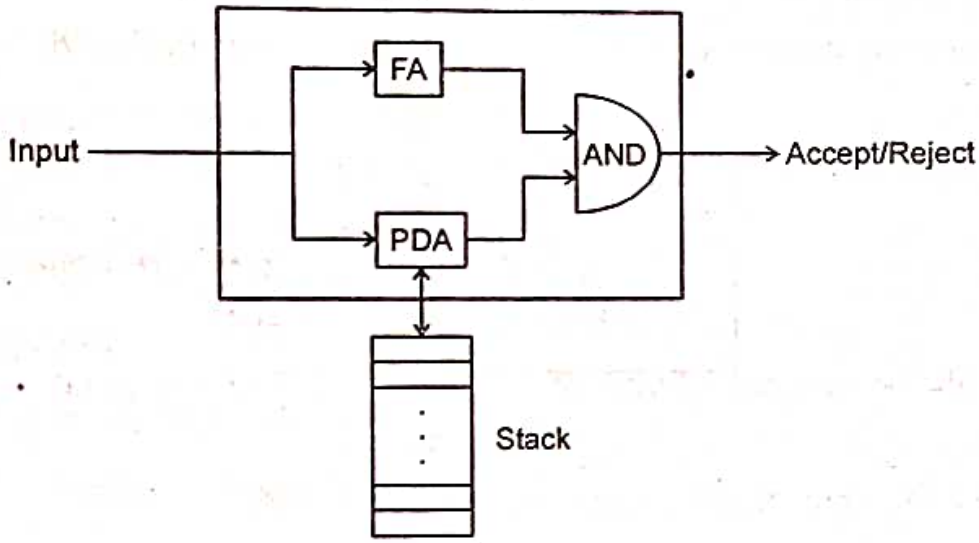


Fig.4.11.Intersection of a CFL and a regular language

where

$\delta((q, p), a, x)$ = set of all pairs of $((r,s), \gamma)$ such that

1. $S = \hat{\delta}_A(p,a)$
2. $\hat{\delta}_p(q,p,x) = \text{pair}(r,\gamma)$

For each move of PDA P and for each move of finite automata A, there should be a corresponding move in PDA P^1.

Note that $a\Sigma\Sigma$ where a is any symbol or ε.

If a = any value, then $\hat{\delta}(p,a) = \delta_A(p)$

If $a = \varepsilon$, then $\hat{\delta}(P,a) = P$

And the number of moves made by the PDA's that

$$(q_{p,}\ w, z_0) \overset{*}{\underset{P^1}{\vdash}} ((q, P)\ \varepsilon, \gamma)$$

where

$P = \hat{\delta}(P_A, w)$

$\therefore$ P^1 accepts w is and only if both P and A accepts w.

4.5.5 INVERSE HOMOMORPHISM

If h is a homomorphism, and L is any language, then $h^{-1}(L)$ is the set of strings w such that h(w) is in L.

The CFL's are closed under inverse homomorphism.

The following figure shows the inverse homomorphism of PDAS.

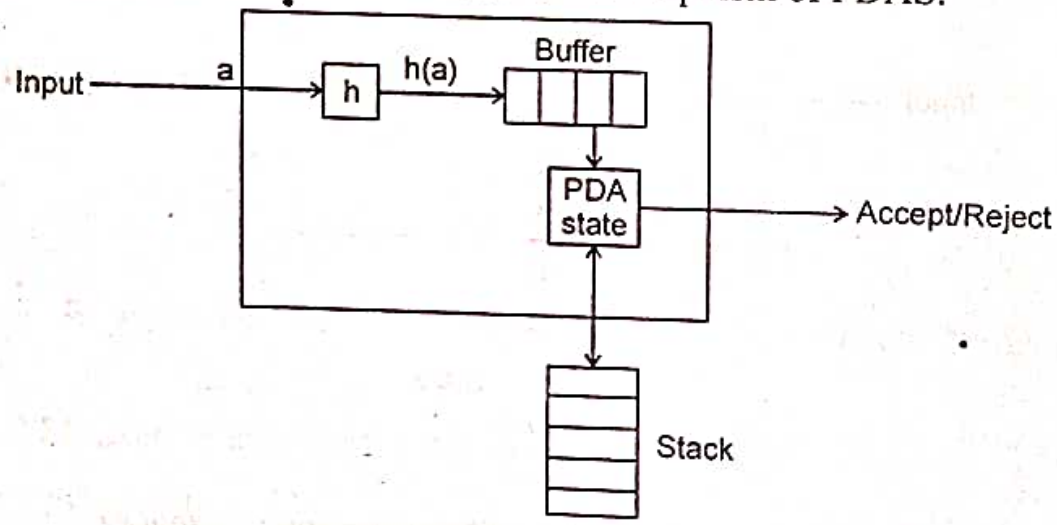


Fig. 4.12. Inverse homomorphism of a PDA

After getting the input a, h(a) is placed in buffer. The symbols of h(a) are used one at a time and fed to the PDA being simulated.

While applying homomorphism, the PDA checks whether the buffer is empty. If it is empty, then the PDA read the input symbols and apply the homomorphism.

4.6 TURING MACHINES

Alan Turing introduced a new mathematical model called Turing Machine during the year of 1936. It is mostly used to define languages and to compute integer functions.

The basic model has a finite control, an input tape that is divided into cells and a tape head that scans one cell of the tape at a time.

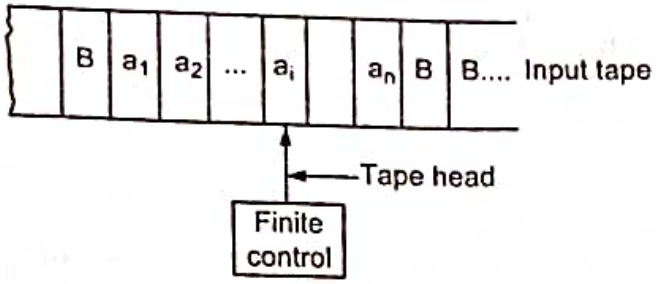


Fig.4.13.Turing machine model

- ✓ Each cell can hold one of a finite number of tape symbols.
- ✓ All other tape symbols extending infinitely to the left and right hold a special symbol called blank.
- ✓ Initially the tape head is pointing the leftmost cell that holds the input.

In one move of the Turing machine, depending upon the symbol scanned by the tape head and the state of finite control,

- ✓ Changes its state.
- ✓ Prints a symbol on the tape cell scanned, replacing what was written there.
- ✓ Moves its head one cell, to its left or right.

4.6.1 DEFINITION

A Turing machine M is a 7-tuple

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

where

Q - Finite set of states

Σ - Finite set of input symbols

Γ - Finite set of tape symbols

δ - Transition function mapping the states of finite automaton and tape symbols to states, tape symbols and movement of the head.

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

$q_0 \in Q$ is the initial state

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

$B \in \Gamma$ is the blank symbol

4.6.2 INSTANTANEOUS DESCRIPTION FOR TURING MACHINES

The ID of a Turing machine is defined in terms of the entire input string and the current state.

Definition

An ID of a Turing machine is a string $\alpha\beta\gamma$, where β is the present state of M, the entire input string is split as $\alpha\gamma$, the first symbol of γ is the current symbol 'a' under R/W head and γ has all the subsequent symbols of the input string and the string α is the substring of the input string formed by all the symbols to the left of a.

Eg. Construct the ID

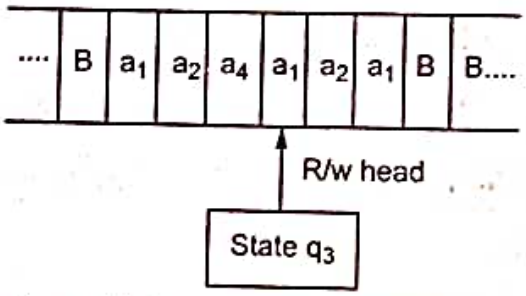


Fig.4.14.Instantaneous description

The present symbol under R/W head is a_1. The present state is q_3. So a_1 is written to the right of q_3. The non-blank symbols to the left of a_1 form the string $a_1a_2a_4$ which is written to the left of q_3.

The sequence of non-blank symbols to the right of a_1 is $a_2\ q_1$. Thus the ID is

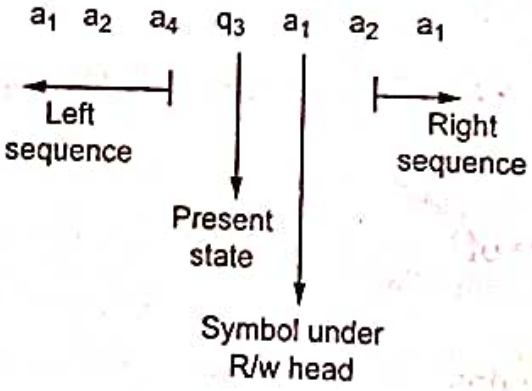


Fig.4.15.

4.6.3 MOVES IN A TURING MACHINE

Let M = (Q, Σ, Γ, δ, q_0, B, F) be a Turing machine. The symbol $\vdash$ is used to represent the move.

$\vdash$ - single move

$\vdash^*$ - zero, one or more moves

$\delta(q, x)$ causes a change in ID of the Turing machine. This is called as a move.

Suppose $\boxed{\delta(q, x_i) = (p, y, L)}$ and the input string to be processed is

$x_1\ x_2 \ldots x_n$

and the present symbol under R/w head is x_i.

Before processing

$x_1\ x_2 \ldots x_{i-1}\ q\ x_i \ldots x_n$

After processing

$x_1\ x_2 \ldots x_{i-2}\ p\ x_{i-1}\ y\ x_{i+1} \ldots x_n$

This change is represented by the move

$x_1\ x_2 \ldots x_{i-1}\ q\ x_i \ldots x_n \vdash x_i \ldots x_{i-2}\ p\ x_{i-1}\ y\ x_{i+1} \ldots x_n$

Suppose $\boxed{\delta(q, xi) = (p, y, R)}$ and the input string to be processed is

$x_1\ x_2 \ldots x_n$

$x_1\ x_2 \ldots x_{i-1}\ q\ x_i \ldots x_n \vdash x_1\ x_2 \ldots x_{i-1}\ yp\ x_{i+1} \ldots x_n$

4.6.4 LANGUAGE OF A TM

The language accepted by M denoted L(M) is the set of those words in Σ^* that cause M to enter a final state.

$L(M) = \{w \mid w \text{ in } \Sigma^* \text{ and } q_0 w \vdash^* \alpha_1 p\alpha_2 \text{ for some p in F and } \alpha_1, \alpha_2 \in \Gamma^*\}$

Example: 4.20 ***Design a TM to accept the language $L = \{0^n\ 1^n \mid n \geq 1\}$***

☺ ***Solution:***

Given a finite sequence of 0's and 1's on its tape. The Turing machine is designed using the following way.

(i) M replaces the leftmost 0 by x, moves right to the leftmost 1, replacing it by y.

(ii) Then M moves left to find the rightmost x and moves one cell right to the leftmost 0 and repeats the cycle.

(iii) While searching for a 1, if a blank is encountered, then M halts without accepting.

(iv) After changing a 1 to y, if M finds no more 0's then M checks that no more 1's remain, accepting the string else not.

Assume the set of states $Q = \{q_0, q_1, q_2, q_3, q_4\}$

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

$\Gamma = \{0, 1, x, y, B\}$

$F = \{q_4\}$

Let q_0 be the initial state and at state q_0, it replaces the leftmost 0 by x, and changes it to q_1. At q_1 M searches right for 1's, skipping over 0's and y's.

If M finds a 1, it changes it to y, entering state q_2. From q_2 it searches left for an x and moves right to change the state to q_0.

At q_0, if y is encountered, it goes to state q_3 and checks that no 1's remain. If the y's are followed by a B, state q_4 is entered and then accepted. And for all others, M rejects.

	0	1	x	y	B
→ q_0	(q_1, X, R)	-	-	(q_3, y, R)	-
q_1	$(q_1, 0, R)$	(q_2, y, L)	-	(q_1, y, R)	-
q_2	$(q_2, 0, L)$	-	(q_0, x, R)	(q_2, y, L)	-
q_3	-	-	-	(q_3, y, R)	(q_4, B, R)
* q_4	-	-	-	-	-

Eg:

(*i*) $q_0\,0011 \vdash x\,q_1\,011 \vdash x0\,q_1\,11 \vdash x\,q_2\,0\,y1 \vdash q_2x0y1 \vdash$
$x\,q_0\,0\,y1 \vdash xx\,q_1\,y1 \vdash xxy\,q_1\,1 \vdash xx\,q_2\,yy \vdash x\,q_2\,xyy \vdash$
$xx\,q_0\,yy \vdash xxy\,q_3y \vdash xxyy\,q_3 \vdash xxyy\,B\,q_4$

Accepted

(*ii*) $q_0\,011 \vdash x\,q_1\,11 \vdash q_1\,xy1 \vdash x\,q_0\,y1 \vdash xy\,q_3\,1$

Rejected

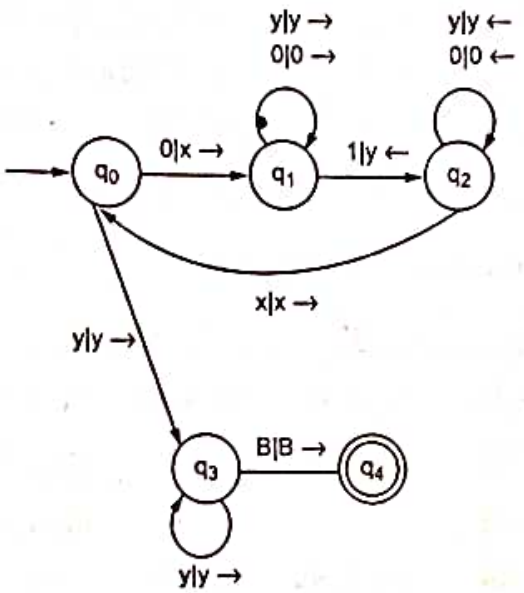


Fig.4.16.Transition diagram for 0^n 1^n

$\therefore$ M = $(Q, \Sigma, \Gamma, \delta, q_0, B, \{q_4\})$

where

$Q = \{q_0, q_1, q_2, q_3, q_4\}$

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

$\Gamma = \{0, 1, x, y, B\}$

q_0 = initial state

q_4 = final state

δ is given in the table

Example:4.21 ***Design a Turing machine for the language L: The set of strings with an equal number of 0's and 1's.***

☺ ***Solution:***

Assume that the input string may start with either 0 or 1, but it should have equal number of 0's and 1's. For eg 0101, 0110, 1001, ...

(i) Change all 0's to x's and all 1's to y's, whether the input symbol may be in any position till reaches the blank symbol.

(ii) Initially, the TM is at state q_0. At q_0 if it finds the leftmost symbol as '0', change it to x and enters q_1 then moves right. If it finds 1 by skipping 0's y's at q_1, change it to y and enters state q_2. At state q_2, the TM searches for the leftmost x by skipping 0's and y's and enters state q_0. Repeats the process till the TM finds blank symbol at state q_0.

(iii) At q_0, if it finds the leftmost symbol as 1, change it to y and enters state q_3. At q_3, if it finds 0's by skipping 1's and x's, change it to x and enters state q_4 by moving left. At q_4, it searches for the leftmost y. If it finds y at q_4, the TM enters state q_0. Repeat the process till it finds a blank symbol.

(iv) For all other state changes, the input is rejected.

(v) The transition table (or) δ is given by:

	0	1	x	y	B
→ q_0	(q_1, X, R)	(q_3, y, R)	(q_0, x, R)	(q_0, y, R)	(q_5, B, R)
q_1	$(q_1, 0, R)$	(q_2, y, L)	-	(q_1, y, R)	-
q_2	$(q_2, 0, L)$	-	(q_0, x, R)	(q_2, y, L)	-
q_3	(q_4, x, L)	$(q_3, 1, R)$	(q_3, x, R)	-	-
q_4	-	$(q_4, 1, L)$	(q_4, x, L)	(q_0, y, R)	-
* q_5	-	-	-	-	-

Eg:

(*i*) $q_0\,1001 \vdash y\,q_3\,001 \vdash q_4\,yx01 \vdash y\,q_0\,x\,01 \vdash yx\,q_0\,01 \vdash yxx\,q_1\,1 \vdash yx\,q_2xy \vdash yxx\,q_0\,y \vdash yxxyq_0B \vdash yxxy\,B\,q_5$

Accepted

(*ii*) $q_0\,0100 \vdash q_2\,xy00 \vdash x\,q_0y00 \vdash xy\,q_000 \vdash xyx\,q_10 \vdash xyx0\,q_1B$

Rejected

Transition diagram

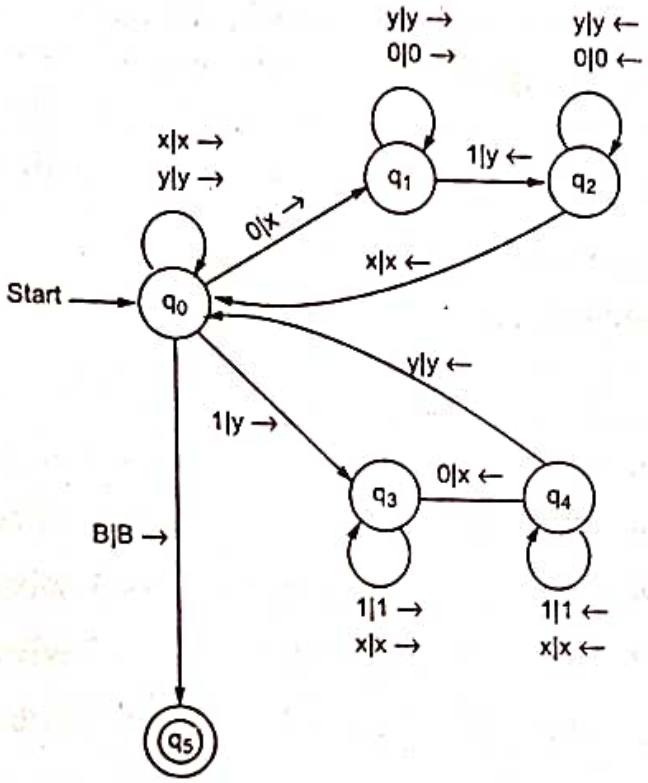


Fig.4.17.

$\therefore$ TM M = $(Q, \Sigma, \Gamma, \delta, q_0, B, \{q_5\})$

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

Σ = {0, 1}

Γ = (0, 1, x, y, B)

q_0 = initial state

q_5 = final state

δ is given in the table

Example:4.22 ***Design a Turing machine that accepts the language*** $L = \{a^n\ b^n\ c^n \mid n > 1\}$

☺ ***Solution:***

The construction is similar to the design of $0^n\ 1^n$.

Here we have to replace each a by x and b by y and c by z repeatedly.

(i) Initially the Turing machine is at state q_0. At q_0, if it finds a's, replace it by x's and moves right with state q_1. At q_1 if it finds b's, replace it by y's and moves right with state q_2. At state q_2 if it finds c's, replace it by z and enters state q_3 by moving left. At q_3, if it finds the leftmost x, by skipping z by, a, then it goes to state q_0. Repeat the process till at q_0, it finds y.

	a	b	c	x	y	z	B
→ q_0	(q_1, X, R)	-	-	-	(q_4, y, R)	-	-
q_1	(q_1, a, R)	(q_2, y, R)	-	-	(q_1, y, R)	-	-
q_2	-	(q_2, b, R)	(q_3, z, L)	-	-	(q_2, z, R)	
q_3	(q_3, a, L)	(q_3, b, L)	-	(q_0, z, R)	(q_3, y, L)	-	-
* q_4	-	-	-	-	(q_4, y, R)	(q_4, z, R)	(q_5, B, R)

Eg:

(*i*) $q_0$aabbcc ⊢ x $q_1$abbcc ⊢ xa q_1 bbcc ⊢
xay $q_2$bcc ⊢ xayb $q_2$cc ⊢ xay $q_3$bzc ⊢
xa $q_3$ybzc ⊢ x $q_3$aybzc ⊢ q_3 xaybzc ⊢
x q_0 aybzc ⊢ xx q_1 ybzc ⊢ xxy q_1 bzc ⊢
xxyy $q_2$zc ⊢ xxyyz q_2c ⊢ xxyy q_3 zz ⊢
xxy $q_3$yzz ⊢ xx q_3 yyzz ⊢ x q_3 xyyzz ⊢
xx q_0 yyzz ⊢ xxy $q_4$yzz ⊢ xxyy $q_4$zz ⊢
xxyyz q_4z ⊢ xxyyzz q_4B ⊢ xxyyzzB q_5 ⊢

Accepted

Transition diagram

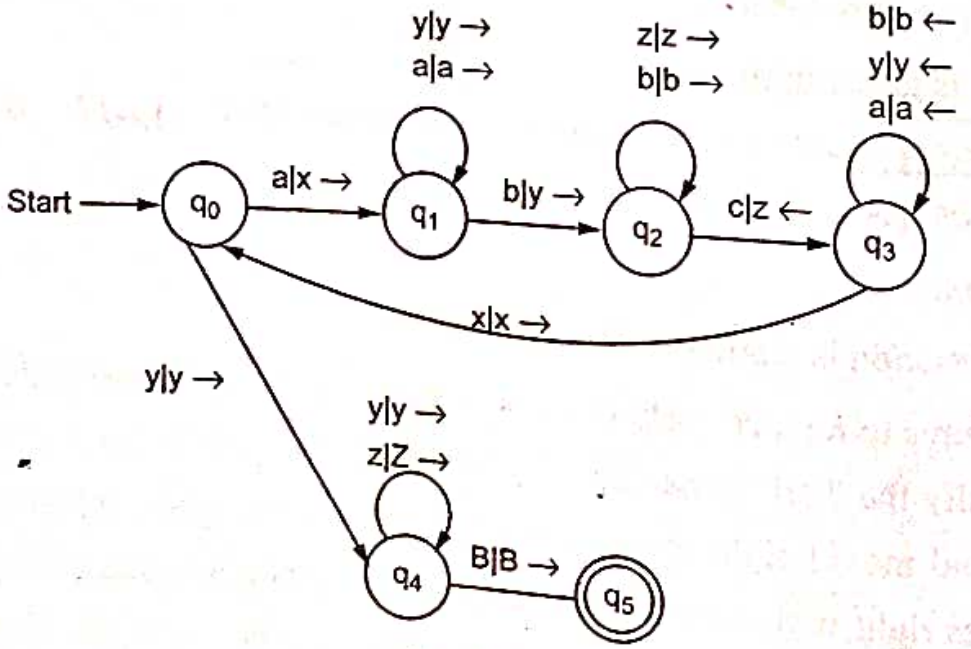


Fig. 4.18.

$M = (Q, \Sigma, \Gamma, \delta, q_0, z_0, B, \{q_5\})$

where

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

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

$\Gamma = \{a, b, c, B, x, y, z\}$

q_0 = initial state

q_5 = final state

4.6.5 TURING MACHINE AS A COMPUTER OF INTEGER FUNCTIONS

The Turing machine is also used as a computer of functions from integers to integers. The traditional approach is to represent integers in unary form. The integer $i \geq 0$ is represented by the string 0^i.

If a function has k arguments say $i_1, i_2, \ldots i_k$, then these integers are initially placed on the tape separated by 1's as $0^{i1}\ 1\ 0^{i2}\ 1 \ldots 10^{ik}$.

The Turing machine halts with a tape consisting of 0^m for some m when the given function is defined as $f(i_1, i_2, \ldots i_k) = m$.

Example:4.23 ***Construct a Turing machine that performs addition operation.***

☺ ***Solution:***

The function is defined *as* $f(x + y) = x + y$

x is given by 0^x

y is given by 0^y

The input is placed on the tape $0^x\ 10^y$. The sum of these two values are performed by replacing the intermediate 1 by 0 and replacing the last 0 by blank symbol.

The transition table is given as follows:

δ:

	0	1	B
→ q_0	$(q_1, 0, R)$	$(q_1, 0, R)$	-
q_1	$(q_1, 0, R)$	-	(q_2, B, L)
q_2	(q_3, B, R)	-	-
* q_3	-	-	-

Eg:

x = 3, y = 2, f(x + y)

(i) $q_0\,0^3\;10^2 \Rightarrow q_0\,000100 \vdash 0\,q_0 00100 \vdash$

$00q_0\,0100 \vdash 000\,q_0 100 \vdash$

$0000q_1\,00 \vdash 00000\,q_1 0 \vdash$

$000000q_1\,B \vdash 00000\,q_2 0B \vdash$

$00000B\,q_3$

Transition diagram

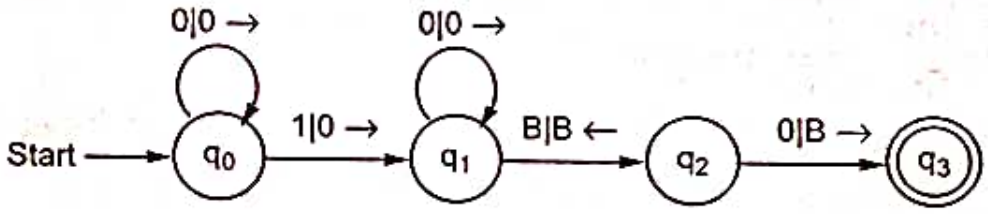


Fig.4.19.

$\therefore$ M = $(Q, \Sigma, \Gamma, \delta, q_0, B\ \{q_3\})$

where

Q = $\{q_0, q_1, q_2, q_3\}$

Σ = {0, 1}

Γ = {0, 1, B}

q_0 = initial state

q_3 = final state

Example:4.24 ***Construct a Turing machine to compute the function $f: N \to N$ such that $f(x) = x + 1$***

☺ ***Solution:***

x is represented as 0^x

$f(x) = x + 1 \Rightarrow 0^{x+1}$

The output contains one more 0 than the input. Initially the Turing machine is at q_0. At q_0, if it reads a blank symbol by skipping 0's, replace it with 0 and enters the final state.

δ:

	0	1
→ q_0	$(q_0, 0, R)$	$(q_1, 0, R)$
• q_1	-	-

The Turing machine M = (Q, Σ, Γ, δ, q_0, B, F)

where

Q = $\{q_0, q_1\}$

Σ = {0}

Γ = {B, 0}

q_0 = initial state

$\{q_1\}$ = final state

Transition diagram

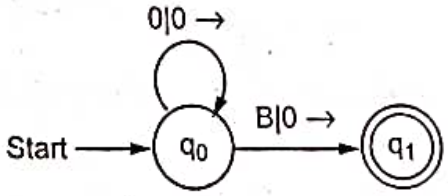


Fig.4.20.

Example:4.25 ***Design a Turing machine to compute f(x) = x + 2.***

☺ ***Solution:***

X can be represented as 0^x .

To find: 0^{x+2}

Initially the TM is at state q_0. At q_0, if it finds a blank symbol, replace it with 0 and enters state q_1. At q_1 if it finds a blank symbol. Replace it with 0 and enters the final state q_2.

δ:

	0	1
→ q_0	$(q_0, 0, R)$	$(q_1, 0, R)$
q_1	-	$(q2, 0, R)$
* q_2	-	-

The Turing machine $M = (Q, \Sigma, \Gamma, \delta, q_0, B, \{q_2\})$

where

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

$\Sigma = \{0\}$

$\Gamma = \{0, B\}$

q_0 = initial state

$\{q_2\}$ = final state

Transition diagram

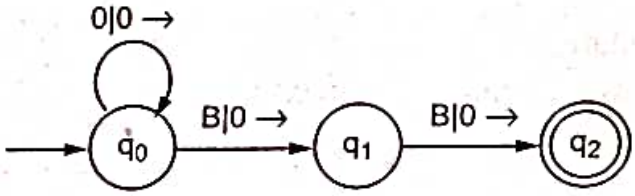


Fig. 4.21.

Example:4.26 ***Design a Turing machine to compute proper subtraction:***

i.e., $m - n$ ***for*** $m \geq n$

0 ***for*** $m < n$

☺ ***Solution:***

The Turing machine started its operation with $0^m\ 10^n$ on its tape. Initially the TM is at state q_0. At q_0, it replaces the leading 0 by blank and search right looking for the first 1. After finding it, the TM searches right for 0 and change it to 1. Then move the tape head to the left till reaches the blank symbol. And then enter state q_0 to repeat the cycle.

The repetition ends if

(i) Searching right for a 0, m encounters a blank. Then n 0's in $0^m\ 10^n$ have all been changed to 1's and n + 1 of the m 0's have been changed to B ($B^{n+1}\ 0^{m-(n+1)}\ 1\ 1^n$). Replace the $(n + 1)^{th}$ 1 by 0 and n B's, leaving $m - n$ 0's on its tape.

(ii) m cannot find a 0 to change it to blank during the beginning of the cycle, change all zeros and 1's to blank and the result is zero.

The role played by each of the 7 states are

q_0: This is the initial state of this design. At this if it reads 0, change it to B's and if it reads 1, change it to B's enters state q_1 and q_5 respectively.

q_1: In this state, if it reads the leftmost 1, m goes to state q_2 by skipping the zeroes.

q_2: m moves right, if it finds 0's by skipping 1's change it to 1's and enter state q_3. In some situation, m encounters a blank at state q_2 and enters state q_4.

q_3: At q_3, m moves left and searches for a blank by skipping 0's and 1's. If it finds B, it moves right and returns to state q_0 and start the cycle again.

q_4: The subtraction is complete, but the n + 1th 0 incorrectly changed to B. m changes all 1's to B until it encounters a B on the tape. It changes the B back to 0 and enters state q_6, where m halts.

q_5: State q_5 is entered from q_0 when m 0's have been changed to B and some of the n 0's are remaining. *i.e.*, m < n, m changes all the remaining 0's and 1's to B and enter the final state q_6.

q_6: This represents the final state of the design.

δ:

	0	1	B
→ q_0	(q_1, B, R)	(q_5, B, R)	-
q_1	$(q_1, 0, R)$	$(q_2, 1, R)$	-
q_2	$(q_3, 1, L)$	$(q_2, 1, R)$	(q_4, B, L)
q_3	$(q_3, 0, L)$	$(q_3, 1, L)$	(q_0, B, R)
q_4	$(q_4, 0, L)$	(q_4, B, L)	$(q_6, 0, R)$
q_5	(q_5, B, R)	(q_5, B, R)	(q_6, B, R)
* q_6	-	-	-

Transition diagram

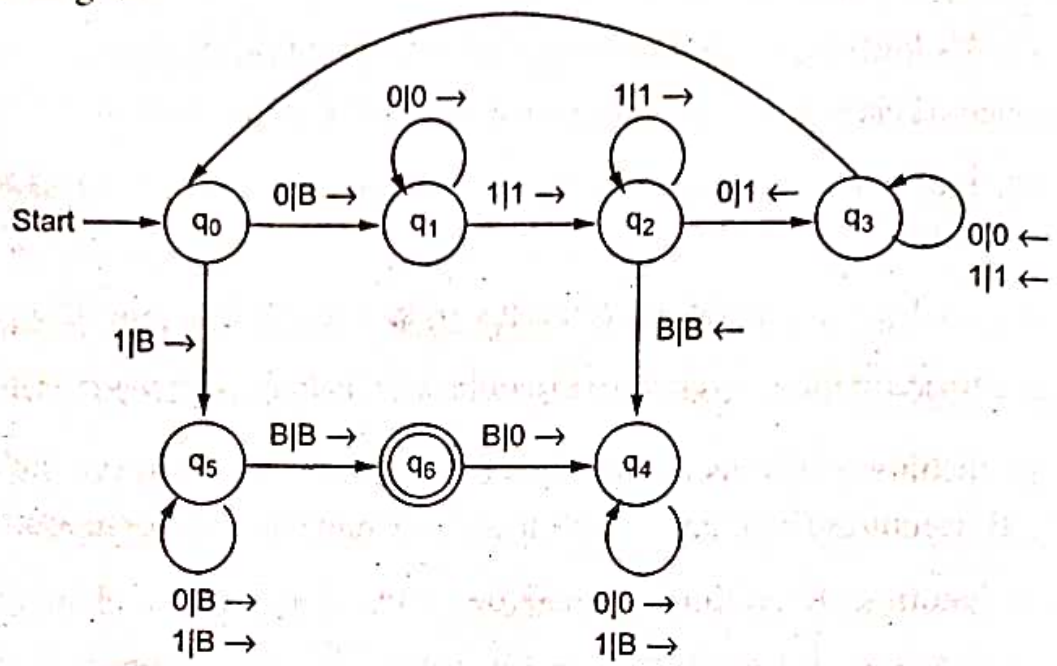


Fig. 4.22. (m – n) Proper subtraction

∴ Turing machine M = (Q, Σ, Γ, δ, q_0, B, {q_6})

where

Q = $\{q_0, q_1, q_2, q_3, q_4, q_5, q_6\}$

Σ = {0, 1}

Γ = {0, 1, B}

q_0 = initial state

q_6 = final state

Eg:

(i) 0010

q_0 0010 ⊢ B q_1 010 ⊢ B 0 $q_1$10 ⊢ B01 $q_2$0 ⊢ B0 $q_3$11 ⊢ B q_3 011 ⊢ q_3B011 ⊢ B q_0 011 ⊢ BB $q_1$11 ⊢ BB1 q_2 1 ⊢ BB11 q_2 ⊢ BB1 q_4 ⊢ BB $q_4$1 ⊢ B q_4 ⊢ B0 q_6

(ii) 0100

q_0 0100 ⊢ B q_1 100 ⊢ B 1 $q_2$00 ⊢ B $q_3$110 ⊢ q_3 B110 ⊢ B q_0 110 ⊢ BB $q_5$10 ⊢ BBB $q_5$0 ⊢ BBBBq_5 ⊢ BBBBB q_6

Exercises

1. Construct a Turing machine to compute the function f(x) = 2x.
2. Construct a Turing machine to compute the function f(x) = x + 2.
3. Construct a Turing machine that accepts strings over {1} containing even number of 1's.
4. Construct a Turing machine that accepts all strings contains substring aba.

4.7 PROGRAMMING TECHNIQUES FOR TURING MACHINE CONSTRUCTION

A Turing machine is also as powerful as a conventional computer. The following are the different techniques of constructing a TM to meet high-level needs.

1. Storage in the finite control (or) state
2. Multiple tracks
3. Subroutines
4. Checking off symbols

4.7.1 STORAGE IN THE STATE (OR) STORAGE IN THE FINITE CONTROL

The finite control can also be used to hold a finite amount of information along with the task of representing a position in the program.

The state is written as a pair of elements, one for control and the other storing a symbol.

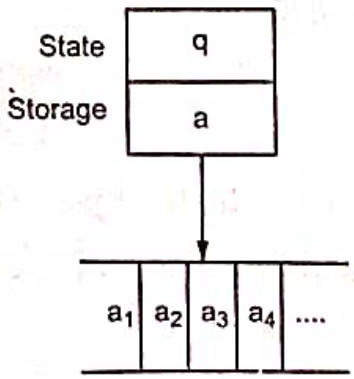


Fig.4.23.Storage in finite control

Example:4.27 ***Consider a Turing machine M = (Q, {0, 1}, {0, 1, B}, δ, [q0, B], [q1, B]} that looks at the first input symbol, records in the finite control and checks that the symbol does not appear elsewhere on its input.***

☺ ***Solution:***

(i) Find the states of Q as $Q \times \{0, 1, B\} = \{q_0, q_1\} \times \{0, 1, B\}$

Q consists of $[q_0, 0]$, $[q_0, 1]$, $[q_0, B]$, $[q_1, B]$, $[q_1, 0]$, $[q_1, 1]$, $[q_1, B]$

(ii) The finite control holds both the state and the symbol.

(iii) At q_0, the TM reads the first symbol a and goes to state q_1

where a = 0 or 1.

$\delta([q_0, B], a) = ([q_1, a], a, R)$

The second symbol is copied into the second component of the state, moves right and enters q_1.

(iv) At q_1, if the TM reads the other symbols, M skips over and moves right.

$\delta([q_1, a], \bar{a}) = ([q_1, a], \bar{a}, R)$

where $\bar{a}$ is the complement of a

If $a = 0$, $\bar{a} = 1$

If $a = 1$, $\bar{a} = 0$

(v) If M reaches the same symbol, it halts without accepting.
(*i.e.*, for $\delta([q_1, a], a)$)

(vi) If M reaches the first blank, it enters the accepting state.
$\delta([q_1, a], B) = ([q_1, B], B, R)$

Input

01 + 1

$\Rightarrow [q_0, B]01 + 1 \vdash 0[q_1, 0]1 + 1 \vdash 01[q_1, 0] + 1$

$\vdash 01 + [q_1, 0]1 \vdash 01 + 1[q_1, 0]B \vdash 01 + 1B[q_1, B]$

Thus the input is accepted.

4.7.2 MULTIPLE TRACKS

It is also possible that a Turing machine's input tape can be divided into several tracks. Each track can hold one symbol, and the tape alphabet of the TM consists of tuples with one component for each track.

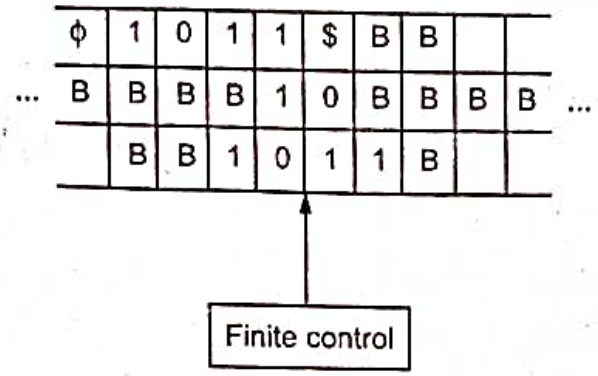


Fig.4.24.A three track Turing machine

Example:4.28 ***Design a Turing machine to check whether the given input is prime or not using multiple tracks.***

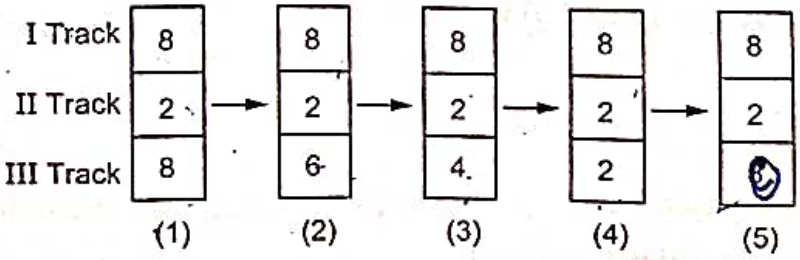
☺ ***Solution:***

The binary input greater than two is placed on the first track. And also the same input is placed on the third track. Then TM writes the number two in binary form on the second track. Then divide the third track by the second as follows.

The number on the second track is subtracted from the third track as many times as possible, till getting the remainder. If the remainder is zero, then the number on the first track is not a prime.

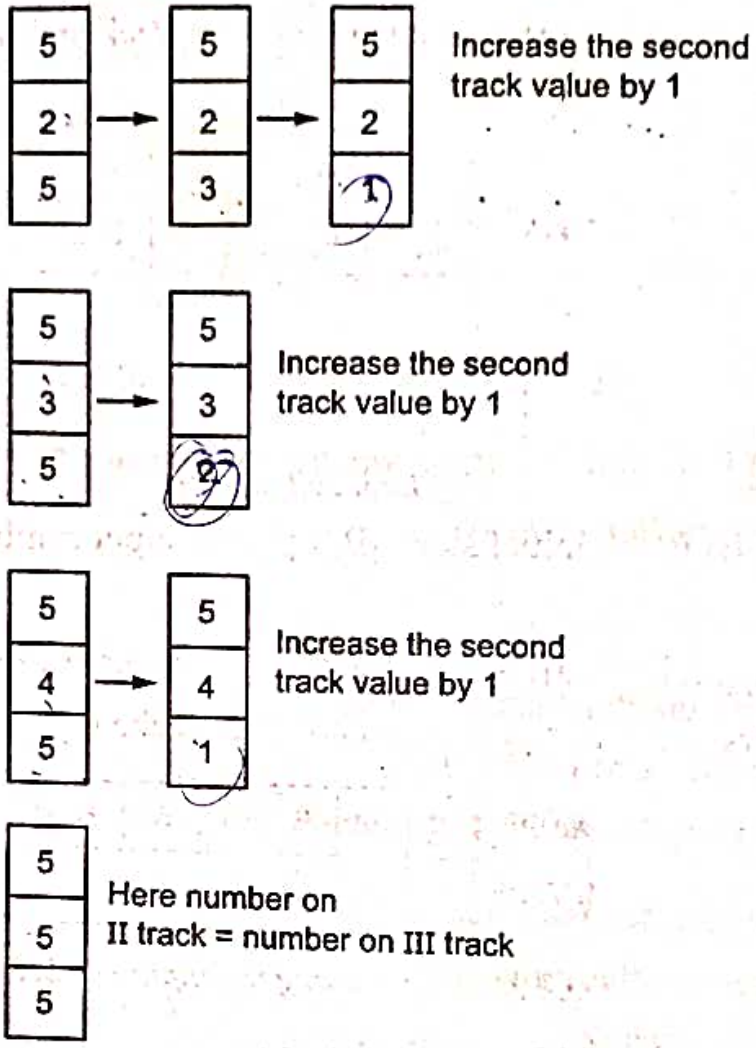
If the remainder is non-zero, then increase the number on the second track by one. If the second track equals the first, the number given is a prime because it should be divided by one and itself.

Eg: (i) 8



The given number is not a prime number.

Eg: (ii) 5



∴ The given number is a prime number.

4.7.3 CHECKING-OFF SYMBOLS

Here one track of the tape can be used to mark the symbols already read.

Example:4.29 ***Design a Turing machine to recognize the language L = {wcw | where w is in (0 + 1)*}.***

☺ ***Solution:***

(i) Store the input symbols in the finite control and the extra track is used to check those input symbols.

(ii) The set of states Q consists of the ordered pairs [q, a] where q is the current state and a is the input symbol stored in that state.

(iii) Every input is also an ordered pair consisting either a blank symbol B or the symbol * to check off symbols in the first track and the input symbol in the second track.

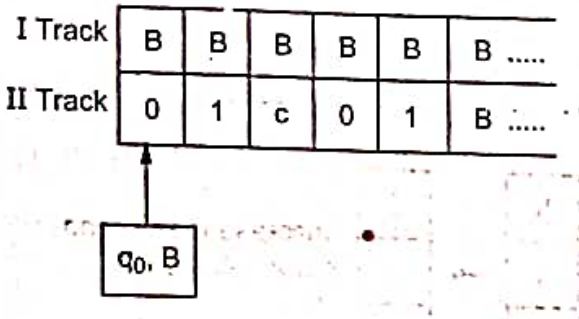


Fig.4.25. Initial state

(iv) Let $[q_0, B]$ is the initial state, [B, a] is the input symbol, where a is eitner 0 or 1.

i.e., $\delta([q_0, B], [B, a]) = ([q_1, a], [*, a], R)$ Make the entry of the 'a' position at the first track to *, as the symbol a is checked.

I Track	*	B	B	B	B	B .....
II Track	0	1	c	0	1	B .....

Fig.4.26.After reading the input symbol a

(v) M moves right looking for the symbol c.

$$\delta([q_1, a], [B, b]) = ([q_1, a], [B, b], R)$$

where a and b can be either 0 or 1 but not c.

(vi) When M finds c, it changes to state q_2.

$$\delta([q_1, a], [B, c]) = ([q_2, a], [B, c], R)$$

(vii) In state q_2, M searches for the matched symbol or else skip to the next symbol.

$$\delta([q_2, a], [*, b]) = ([q_2, a], [*, b], R)$$

(viii) If the unchecked symbol matches with the symbol at the finite control, it performs the check-off and marked with * in the allotted track.

$$\delta([q_2, a], [B, a]) = ([q_3, B], [*, a], L)$$

$$\delta([q_3, B], [B, c]) = ([q_4, B], [B, c], L)$$

$$\delta([q_4, B], [B, a]) = ([q_5, B], [B, a], L)$$

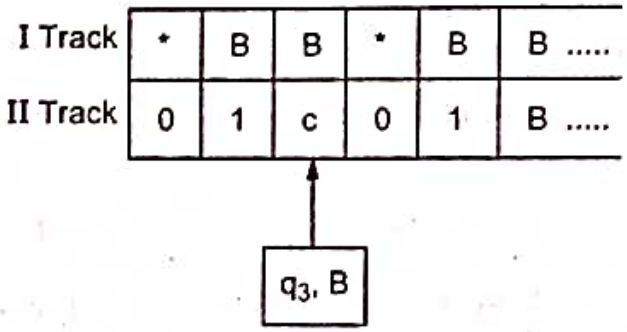


Fig 4.27.After reading the input symbol a at state q_2

(ix) Repeat the process cyclically till it meets the input symbol [B, B]

$\delta([q_5, B], [B, a]) = ([q_5, B], [B, a], L)$

$\delta([q_5, B], [*, a]) = ([q_0, B], [*, a], R)$

M remains in state q_6 and proceeds left.

(x) $\delta([q_4, B], [*, a]) = ([q_6, B], [*, a], R)$

Pick the branch from state q_5 and moved left from the c.

(xi) $\delta([q_6, B], [B, c]) = ([q_7, B], [B, c], R)$

(xii) $\delta([q_7, B], [*, a]) = ([q_7, B], [*, a], R)$

(xiii) $\delta([q_7, B], [B, B]) = ([q_8, B], [B, B], R)$

4.7.4 SUBROUTINES

A problem with same tasks to be repeated for many number of times, can be programmed using subroutines. A Turing machine with subroutine is a set of states that perform some useful processes.

The idea here is to write part of a TM programs to serve as a subroutine which has its own initial state and a return state for returning to the calling routine. It improves the modular or top-down programming design.

Example:4.30 ***Construct a Turing machine m to implement the total ecursive function for multiplication 0^{mn} where m and n are two positive numbers.***

☺ ***Solution:***

m starts with 0^m 1 0^n on its tape and ends with 0^{mn}. The idea is to place a 1 after 0^m 1 0^n and then copy the block of n 0's on to the right end m times, each time erasing one of the m 0's. Then the remaining n 0's and 1's are to be erased and finally getting the result 0^{mn}.

Here the copying operation is repeated for m times. Let us consider this operation as a subroutine copy.

This algorithm enters its initial state when m has an ID $0^m\ 1\ q_1\ 0^n\ 10^i$.

In state q_1, when it finds 0, M changes it to 2 and enters state q_2. In state q_2, m searches for the blank symbol B by skipping 0's and 1's and change it into 0 and move to state q_3. In state q_3, m moves left to 2. If a 2 is encountered, it enters the state q_1 and the process is repeated until it finds 1. If it encounters 1, it means that the copying process is complete. State q_4 is used to convert the 2's back to 0's and the subroutine halts in q_5.

Transition function δ

	0	1	2	B
q_1	$(q_2, 2, R)$	$(q_4, 1, L)$	-	-
q_2	$(q_2, 0, R)$	$(q_2, 1, R)$	-	$(q_3, 0, L)$
q_3	$(q_3, 0, L)$	$(q_3, 1, L)$	$(q_1, 2, R)$	-
q_4	-	$(q_5, 1, R)$	$(q_4, 0, L)$	-

Fig.4.28. Subroutine copy

Initially the TM is in the position $q_0\ 0^m\ 10^n$. The TM enters into the subroutine only at B $0^{m-1}\ 1q\ 0^n\ 1$. The transition function δ for the main program is given by

	0	1	2	B
→ q_0	(q_6, B, R)	-	-	-
q_5	$(q_7, 0, R)$	-	-	-
q_6	$(q_6, 0, R)$	$(q_1, 1, R)$	-	-
q_7	-	$(q_8, 1, L)$	-	-
q_8	$(q_9, 0, L)$	-	-	(q_{10}, B, R)
q_9	$(q_9, 0, L)$	-	-	(q_0, B, R)
q_{10}	-	(q_{11}, B, R)	-	-
q_{11}	(q_{11}, B, R)	(q_{12}, B, R)	-	-
q_{12}	-	-		

Input: m = 2, n = 2

Initial state: $q_0 0^m 10^n$

q_0 001001 ⊢ B q_6 01001 ⊢ B0 q_6 1001 ⊢ B01 q_1 001

⊢ B012 q_2 01 ⊢B0120 $q_2$1B ⊢ B01201 q_2B ⊢B0120 $q_3$10

⊢B012 $q_3$010 ⊢B01 $q_3$2010 ⊢B012 $q_1$010 ⊢B0122 q_2 10

⊢B01221 $q_2$0 ⊢ B012210 q_2B ⊢ B01221 $q_3$00

⊢B0122 $q_3$100 ⊢ B0121 $q_3$2100 ⊢ B0122 $q_1$100

⊢B012 $q_4$2100 ⊢ B01 $q_4$20100 ⊢ B0 $q_4$100100

⊢B01$q_5$00100 ⊢ B0$q_7$100100 ⊢ B$q_8$0100100

⊢ q_9B0100100 ⊢ B $q_0$0100100 ⊢ ...

⊢ ... (Enter into subroutine copy)

⊢ BB1 q_5 0010000 ⊢ BB q_7 10010000

⊢ B q_8 B10010000 ⊢ B q_{10} 10010000

⊢ B q_{11} 0010000 ⊢ B q_{11} 010000

⊢ B q_{11} 10000 ⊢ q_{12} 0000

4.8 MULTI-TAPE TURING MACHINE

A multi-tape Turing machine has a finite control with some finite number of tapes. Each tape is infinite in both directions. It has its own initial state and some accepting states.

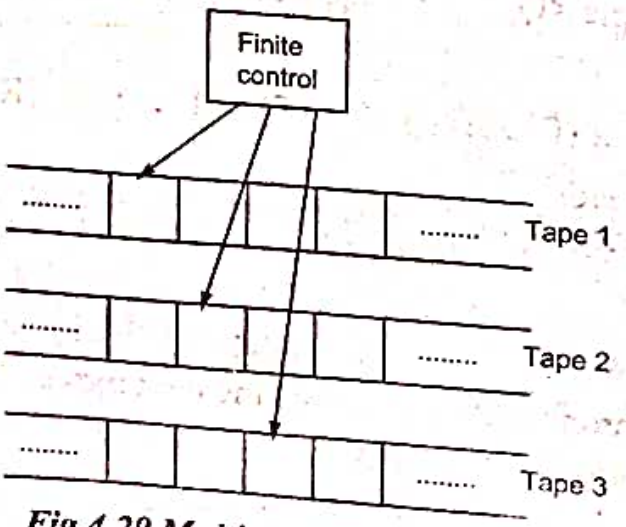


Fig.4.29.Multi-tape Turing machine

Initially

- ✓ The finite set of input symbols is placed on the first tape.
- ✓ All the other cells of all the tapes hold the blank.
- ✓ The control head of the first tape is at the left end of the input.

In one move, the multi-tape TM can

- ✓ Change state
- ✓ Print a new symbol on each of the cell scanned by its tape heads.
- ✓ Move each of its tape heads, independently, one cell to the left or right or keep it stationary.

4.8.1 MULTI-HEAD TURING MACHINE

A single tape Turing machine with multiple read/write heads. In every state of the Turing machine, only one head may be used. For K-heads, there is a set of states $Q_1, Q_2, \ldots Q_k$ where each Q_i contains the set of states which is used by the i^{th} head.

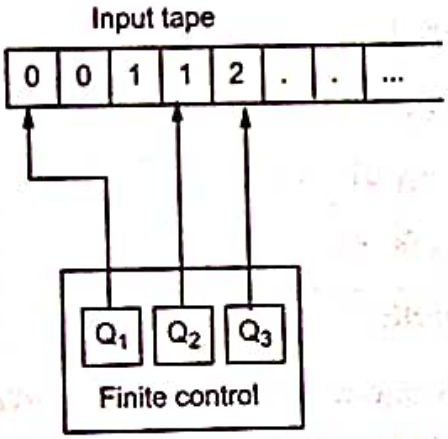


Fig. 4.30. Multi-head Turing machine

It is denoted by a 5-tuple $<Q, \Sigma, \Gamma, \delta, q_0, B, F>$

where δ is given by

$\delta: Q \times \{H_1, H_2, \ldots H_n\} \times (\Gamma \cup B) \rightarrow (Q \cup F) \times (\Gamma \cup B) \times \{R, L\}$

$H_1, H_2, \ldots H_n$ – tape heads

4.9 NON-DETERMINISTIC TURING MACHINE

A non-deterministic Turing machine is a device with a finite control and a single one-way infinite tape. For a given state and a tape symbol scanned by the tape head, the machine has a finite number of choices for the next move.

For δ (q, X) there is a set of triples like

$$\{(q_1, Y_1, D_1), (q_2, Y_2, D_2) \ldots (q_k, Y_k, D_k)\}$$

where k is any integer.

M accepts an input 'w' if there is any sequence of choices of move that leads from the initial ID with w as input, to an ID with an accepting state.

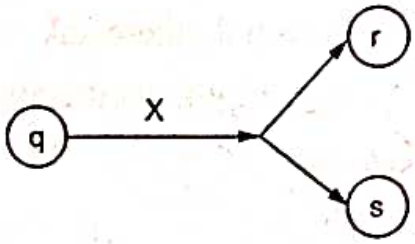


Fig. 4.31. Non-deterministic Turing machine

SOLVED PROBLEMS

Example:4.31 ***Construct a TM that accept strings over {1} containing even number of 1's.***

☺ ***Solution:***

The input string contains only strings of 1's. It should be checked that the number of 1's in that string should be even.

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

$F = \{q_1\}$

$\Sigma = \{1\}$

$\Gamma = \{1, x, B\}$

(i) Read the input symbols one by one where on reading one 1, replace it by x and go to next state. At this state if it encounters 1, replace it by x and go to the previous state.

(ii) Repeat the steps, till q_1 encounters the blank.

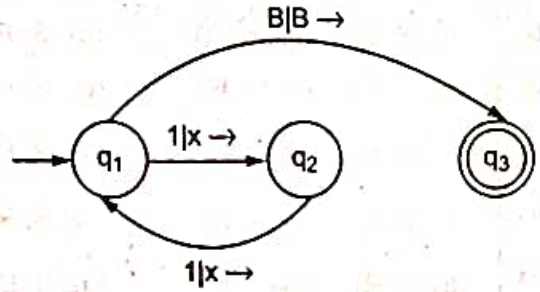
	1	B
→ q_0	(q_2, x, R)	(q_3, B, R)
q_1	(q_1, x, R)	-
* q_2	-	-

Eg:

(*i*) $q_1 11B \vdash x\, q_2 1 \vdash xx\, q_1 B \vdash xxB\, q_3$ accepted

(*ii*) $q_1 111B \vdash x\, q_2 11 \vdash xx\, q_1 1 \vdash xxx\, q_2 B$ rejected

Transition diagram



∴ The Turing machine M is given by

$$M = (Q, \Sigma, , \delta, q_0, B, \{q_3\})$$

where

$Q = \{q_1, q_2, q_3\}$

$\Sigma = \{1, x\}$

$\Gamma = \{1, x, B\}$

q_1 = initial state

q_3 = final state

δ is provided in the table

Example:4.32 ***Construct a Turing machine that accepts all palindromes over {0, 1}.***

☺ ***Solution:***

Let L = {w ∈ {0, 1} * | w is a palindrome}

The Turing machine is constructed in such a way that the leftmost symbol is replaced by B and keeps on moving right without changing any symbols until B occurs. For B, it moves left by one cell and write B if it is same as the first symbol, otherwise it halts without accepting.

Then it moves left till the end to repeat the process. Suppose if the given string is of length n which is odd, it enters the final state on seeing the blank symbol B in the midway.

	0	1	B
→ q_0	(q_1, B, R)	(q_4, B, R)	(q_7, B, R)
q_1	$(q_1, 0, R)$	$(q_1, 1, R)$	(q_2, B, L)
q_2	(q_3, B, L)	-	(q_7, B, R)
q_3	$(q_3, 0, L)$	$(q_3, 1, L)$	(q_0, B, R)
q_4	$(q_4, 0, R)$	$(q_4, 1, R)$	(q_5, B, L)
q_5	-	(q_6, B, L)	(q_7, B, R)
q_6	$(q_6, 0, L)$	$(q_6, 1, L)$	(q_0, B, R)
* q_7	-	-	-

Eg:

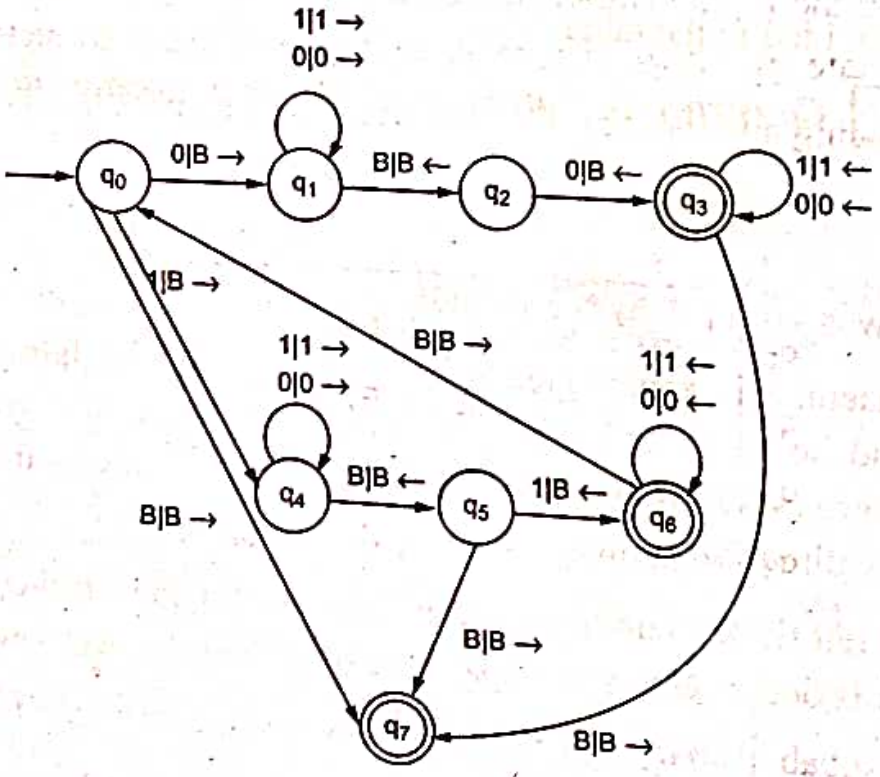
q_0110 $\vdash B\, q_1\, 110$ $\vdash B1\, q_1 10$ $\vdash B11\, q_1 0$

$\vdash B110\, q_1\, B$ $\vdash B11\, q_2 0B$ $\vdash B1\, q_3\, 1BB$ $\vdash B\, q_3 11B$

$\vdash q_3 B11B$ $\vdash q_0\, 11B$ $\vdash B\, q_4\, 1B$ $\vdash B1\, q_4 B$

$\vdash B\, q_5 1B$ $\vdash q_6 BB$ $\vdash q_0 B$ $\vdash q_7\, B$ accepted

Transition diagram



∴ The Turing machine M is given by

$M = (Q, \Sigma, \Gamma, \delta, q_0, B, q_7)$

where

$Q = \{q_0, q_1, q_2, q_3, q_4, q_5, q_6, q_7\}$

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

$\Gamma = \{0, 1, B\}$

q_0 = initial state

q_7 = final state

δ is defined in the table

Example:4.33 ***Construct a TM M that accepts all strings contain a substring aba.***

☺ ***Solution:***

The Turing machine is designed to accept all strings contain a substring *aba*. Initially it reads all symbols and moves right except a.

At q_0, it if reads the input symbol a, it goes to next state q_1. At q_1, if it reads b, it goes to the next state q_2. At q_2, if it reads a, it goes to the next state q_3. At q_3, it moves right on reading any input symbol till reads blank symbol B.

δ:

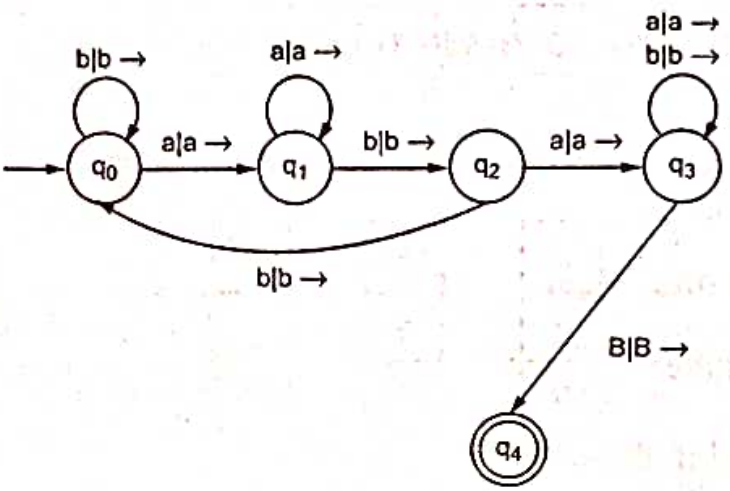
	a	b	B
q_0	(q_1, a, R)	(q_0, b, R)	-
q_1	(q_1, a, R)	(q_2, b, R)	-
q_2	(q_3, a, R)	(q_0, b, R)	-
q_3	(q_3, a, R)	(q_3, b, R)	(q_4, B, R)

Eg:

(*i*) q_0 abab $\vdash$ a $q_1$bab $\vdash$ ab q_2 ab $\vdash$ aba q_3b $\vdash$ abab q_3B $\vdash$ abab q_4 accepted

(*ii*) q_0, abba $\vdash$ a $q_1$bba $\vdash$ ab q_2 ba $\vdash$ abb q_0a $\vdash$ abba q_1B rejected

Transition diagram



∴ The Turing machine M is given by

$$M = (Q, \Sigma, \vdash, \delta, q_0, B, \{q_4\})$$

where

$$Q = \{q_0, q_1, q_2, q_3, q_4\}$$
$$\Sigma = \{a, b\}$$
$$\Gamma = \{a, b, B\}$$
$$S = q_0$$
$$F = \{q_4\}$$

Example: 4.34 ***Construct a TM that takes an input greater than 2 and checks whether it is even or not.***

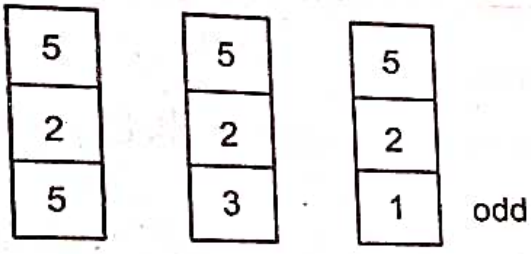
☺ ***Solution:***

The input is placed on the first track and the integer 2 is placed on the second track. The input on the first track is copied into third track.

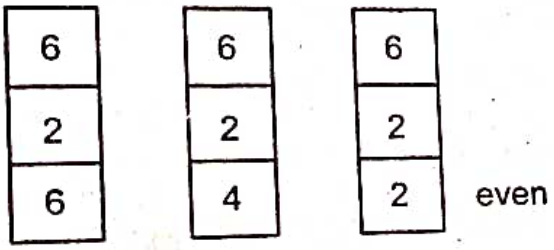
Initial state

The numbers on the second track is subtracted from the third track. If the remainder is same as the number in the second track, then the given number is even. If it is greater than 2, then continue. This process repeats until the remainder in the third track is less than or equal to 2. If it is equal to 2, then the number is even, else it is odd.

Eg: n = 5



N = 6



TWO MARKS QUESTIONS AND ANSWERS

1. *What is a Turing machine?*

A finite state machine with storage is called a Turing machine, where the tape head moves in both the directions.

2. *Define a Turing machine.*

The Turing machine is denoted by

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

where
- Q - finite set of states
- Γ - finite set of allowable tape symbols
- B - a symbol of Γ, a blank
- Σ - set of input symbols not including blank
- $q_0 \in Q$ - start state
- $F \subseteq Q$ - set of final states
- δ - transition function mapping

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

where L, R – directions(Left and Right)

3. *What are the required fields of an instantaneous description or configuration of a TM?*

It requires

- ✓ The state of the TM.
- ✓ The contents of the tape.
- ✓ The position of the tape head on the tape.

4. *What is multiple tracks Turing machine?*

A Turing machine in which the input tape is divided into multiple tracks where each track is having different inputs is called multiple track Turing machine.

5. *What is multidimensional Turing machine?*

The Turing machine which has the usual finite control, but the tape consists of a k-dimensional array of cells infinite in all 2K directions for some fixed K. Depending on the state and symbol scanned, the device changes state, prints a new symbol and moves its tape head in one of 2K directions along one of the K axes.

6. ***When is a function f said to be Turing computable?***

A Turing machine defines a function $y = f(x)$ for strings $x, y \in \Sigma^*$, if $q_0 x \overset{*}{\vdash} q_f y$.

where

q_0-initial state; q_f– final state

A function f is 'Turing computable' if there exists a Turing machine that performs a specific function.

7. ***What is off-line Turing machine?***

An off-line Turing machine is a Multi-tape TM whose input tape is read-only. The Turing machine is not allowed to move the input tape head off the region between left and right end markers.

8. ***What are the applications of TM?***

TM can be used as:

- ✓ Recognizers of languages.
- ✓ Computing functions
- ✓ Computers of functions on non negative integers.
- ✓ Generating devices.

9. ***What is the basic difference between 2-way FA and TM?***

Turing machine can change symbols on its tape, whereas the FA cannot change symbols on tape. Also TM has a tape head that moves both left and right side, whereas the FA doesn't have such a tape head.

10. ***What is (a) total recursive function and (b) partial recursive function?***

If $f(i_1,i_2,\ldots\ldots i_k)$ is defined for all $i_1,\ldots..i_k$ then we say f is a total recursive function. They are similar to recursive languages as they are computed by TM that always halts.

A function $f(i_1,\ldots i_k)$ computed by a Turing machine is called a partial recursive function. They are similar to RE languages as they are computed by TM that may or may not halt on a given input.

11. ***Give examples of total recursive functions.***

All common arithmetic functions on integers such as multiplication, n!, [log2n] are total recursive functions.

12. *What are (a) recursively enumerable languages (b) recursive sets?*

The languages that is accepted by TM is said to be recursively enumerable (r. e) languages. Enumerable means that the strings in the language can be enumerated by the TM. The class of r. e languages includes CFL's.

The recursive sets include languages accepted by at least one TM that halts on all inputs.

13. *What are the possibilities of a TM when processing an input string?*

- ✓ TM can accept the string by entering accepting state.
- ✓ It can reject the string by entering non-accepting state.
- ✓ It can enter an infinite loop so that it never halts.

14. *What are the techniques for Turing machine construction?*

- ✓ Storage in finite control.
- ✓ Multiple tracks.
- ✓ Checking off symbols.
- ✓ Shifting over
- ✓ Subroutines.

15. *What is the storage in FC?*

The finite control (FC) stores a limited amount of information. The state of the finite control represents the state and the second element represents a symbol scanned.

16. *When is checking off symbols used in TM?*

Checking off symbols is useful method when a TM recognizes a language with repeated strings and also to compare the length of substrings. This is implemented by using an extra track on the tape with symbols Blank or _.

17. *When is shifting over used?*

A Turing machine can make space on its tape by shifting all nonblank symbols a finite number of cells to the right. The tape head moves to the right, repeatedly storing the symbols in the FC and replacing the symbols read from the cells to the left. The TM can then return to the vacated cells and prints symbols.

18. What is a multi-head TM?

A k-head TM has some k heads. The heads are numbered 1 through k, and move of the TM depends on the state and on the symbol scanned by each head. In one move, the heads may each move independently left or right or remain stationary.

19. What is a 2-way infinite tape TM?

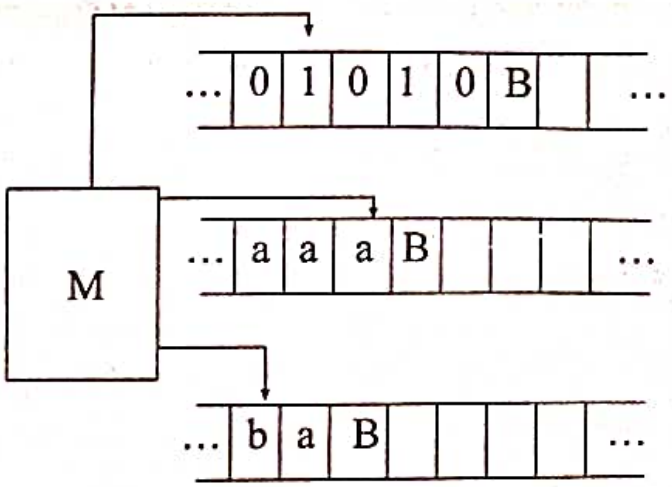
In 2-way infinite tape TM, the tape is infinite in both directions. The leftmost square is not distinguished. Any computation that can be done by 2-way infinite tape can also be done by standard TM.

20. Differentiate PDA and TM.

PDA	TM
1. PDA uses a stack for storage.	1. TM uses a tape that is infinite.
2. The language accepted by PDA is CFL.	2. TM recognizes recursively enumerable languages.

21. What is a multi-tape Turing machine?

A multi-tape Turing machine consists of a finite control with k-tape heads and k-tapes; each tape is infinite in both directions. On a single move depending on the state of finite control and symbol scanned by each of tape heads, the machine can change state print a new symbol on each cells scanned by tape head, move each of its tape head independently one cell to the left or right or remain stationary.



22. *When a recursively enumerable language is said to be recursive? Is it true that the language accepted by a non-deterministic Turing machine is different from recursively enumerable language?*

A language L is recursively enumerable if there is a TM that accepts L and recursive if there is a TM that recognizes L. Thus RE language is Turing acceptable and recursive language is Turing decidable languages. No, the language accepted by non-deterministic Turing machine is same as recursively enumerable language.

23. *What is Church's Hypothesis?*

The notion of computable function can be identified with the class of partial recursive functions is known as Church-hypothesis or Church-Turing thesis. The Turing machine is equivalent in computing power to the digital computer.

24. *What is multi-head Turing Machine?*

This is a kind of Turing machines that have one finite control and one tape but more than one read-write heads. In each state only one of the heads is allowed to read and write. It is denoted by a 5-tuple $< Q, \Sigma, \Gamma, q0, \delta>$.

The transition function is a partial functionδ:

$Q \times \{H_1, H_2 ..., H_n\} \times (\Gamma \cup \{\Delta\}) \rightarrow (Q \cup \{h\}) \times (\Gamma \cup \{\Delta\} \times \{R, L\}$

where $H_1, H_2 ..., H_n$ denote the tape heads.

It can be easily seen that this type of Turing machines are as powerful as one tape Turing machines.

25. *What are the two major normal forms for context-free grammars?*

The two normal forms are

- Chomsky normal form
- Greibach normal form

26. *What is a useless symbol?*

A symbol x is useful if there is a derivation.

$S \overset{*}{\Rightarrow} \alpha X \beta \overset{*}{\Rightarrow} w$ for some $\alpha, \beta, w \in T^*$

or else, it is useless.

27. How do you simplify the context free grammar?

- ✓ First eliminate useless symbols, where the variables or terminals that do not appear in any derivation of a terminal string from the start symbol.
- ✓ Next eliminate ε-production which is of the form $A \rightarrow \varepsilon$ for some variable A.
- ✓ Eliminate unit productions, which are of the form $A \rightarrow B$ for variables A, B.
- ✓ Finally use any of the normal forms to get the simplified CFG.

28. Define nullable variable.

Nullable variable in a CFG G = (V, T, S, P) can be defined as follows:

- ✓ Any variable A for which P contains the production $A \rightarrow \xi$ is nullable.
- ✓ If P contains the production $A \rightarrow B, B_2, \ldots, B_n$ and $B_1, B_2, \ldots B_n$ are nullable variables, then A is nullable.
- ✓ No other variables in V are nullable,

29. Define generating symbol.

Let G = (V, T, P, S) x is generating, if $x \overset{*}{\Rightarrow} w$ for some terminal string w.

Eg: $A \rightarrow aAB \mid \varepsilon$

$B \rightarrow b$

Then A is a generating symbol since

$A \overset{*}{\Rightarrow} ab$

25. Let G = (V, T, P, S) with the productions given by

$S \rightarrow aSbS \mid B \mid \varepsilon$

$B \rightarrow abB$

Eliminate the useless production.

☺ **Solution:**

V = {S, B}

T = {a, b, c}

- ✓ Check whether the variables {S, B} produce the terminal string.
- ✓ S produces the terminal string ε, but B does not B has only productions like $B \rightarrow abB$, where it does not produce the terminal string. So,
- ✓ Remove the variable 'B' and its productions $B \rightarrow abB$, finally we get

$S \rightarrow aSbS \mid \varepsilon$

26. *What is substitution rule?*

A production $A \to x_1 B x_2$ can be eliminated from a grammar if B is replaced by all strings derived by B in one step, provided A and B are variables.

27. *What is a useful production?*

Let G = (V, T, P, S) be a CFG. A variable A ε V is said to be useful if and only if there is atleast one w ε L(G) such that

$S \overset{*}{\Rightarrow} x\,Ay \overset{*}{\Rightarrow} w$

with x, y in $(V \cup T)^*$.

28. *Determine whether the grammar G has a useless production?*

$S \to A$

$A \to aA \mid \varepsilon$

$B \to bA$

The variable B is useless, since it is used by the start variable or by the variable in the start production.

$\therefore$ $B \to bA$ is a useless production.

29. *Write a procedure to eliminate ε production.*

(i) For all productions $A \to \varepsilon$, put A into V^1

(ii) Repeat the following steps until no new variables are added.

(*a*) For all productions

$B \to A_1 A_2 A_3 \ldots A_n$

where $A_1 A_2 A_3 \ldots A_n$ are in V^1

(*b*) Put B into V^1

30. *Write the procedure to eliminate the unit production.*

✓ Find all variables B, for each A such that

$A \overset{*}{\Rightarrow} B$

✓ The new grammar G^1 is obtained by letting into P^1 all non-unit productions of P.

✓ For all A and B satisfying $A \overset{*}{\Rightarrow} B$, add to P^1

$A \to y_1 \mid y_2 \mid \ldots \mid y_n \mid$

where $B \to y_1 \mid y_2 \mid \ldots \mid y_n \mid$ is the set of productions in P^1.

31. Define CNF.

A CFG without ε-productions is generated by a grammar in which the productions are of the form:

$A \rightarrow BC$

or $A \rightarrow a$

where A, B ε V and a ε T

32. What is GNF?

Every context-free language L without ε can be generated by a grammar for which every production is of the form $A \rightarrow a\alpha$, where A ε V, a ε T, α is a string of variables.

38. Define Pumping Lemma for CFL.

Let L be any CFL. Then there exists a constant n, depending only on L such that if z is in L and $|z| \geq n$, then we can write z = uvwxy such that

(*i*) $|vx| \geq 1$

(*ii*) $|vwx| \leq n$

(*iii*) For all $i \geq 0$, $uv^i wx^i y \Sigma L$

REVIEW QUESTIONS

1. Design a TM to accept the language.
   $L = \{x \in \{a, b\}^* : x \text{ ends with } aba\}$
2. Construct a TM that computes the function $f(n) = n \pmod z$.
3. Construct a TM to compute the function $f(n_1, n_2) = \min(n_1, n_2)$ for all non-negative integers n_1 and n_2.
4. Construct a TM that recognizes the set $\{0^{2n}\ 1^n : n \geq 0\}$
5. Is it possible that a Turing machine could be considered as a computer of functions from integers to integer? If yes, justify your answer.
6. Design a Turing machine to compute proper subtraction $m - n$.
7. Design a Turing machine M to implement the function "Multiplication" using the subroutine 'Copy'.
8. Explain how a Turing machine with the multiple tracks of the tape can be used to determine the given number is prime or not?
9. Design a Turing machine to accept the language $L = 0^n\ 1^n \mid n \geq 1$ and simulate its action on the input 00111.
10. Explain how the finite control of a Turing machine can be used to hold a finite amount of information with an example.
11. Design a Turing machine to compute $\forall (m, n) = m * n, m, n \in N$.
12. Explain how a multiple track in the Turing machine can be used for testing given positive integer is a prime or not.
13. Design a Turing machine to compute $\forall (m + n) = m + n, m, n > 0$ and simulate their action on the input 0100.