REGULAR GRAMMARS

After going through this chapter, you should be able to understand :

- Regular Grammar
- Equivalence between Regular Grammar and FA
- Interconversion

4.1 REGULAR GRAMMAR

Definition : The grammar G = (V, T, P, S) is said to be regular grammar iff the grammar is right linear or left linear.

A grammar G is said to be right linear if all the productions are of the form

$A \rightarrow wB$ and / or $A \rightarrow w$ where $A, B \in V$ and $w \in T^*$.

A grammar G is said to be left linear if all the productions are of the form

$A \rightarrow Bw$ and / or $A \rightarrow w$ where $A, B \in V$ and $w \in T^*$.

Example 1 : The grammar

$S \rightarrow aaB \mid bbA \mid \in$

$A \rightarrow aA \mid b$

$B \rightarrow bB \mid a \mid \in$

is a right linear grammar. Note that $\in$ and string of terminals can appear on RHS of any production and if non - terminal is present on R. H. S of any production, only one non - terminal should be present and it has to be the right most symbol on R. H. S.

Example 2 :

The grammar

$S \rightarrow Baa \mid Abb \mid \in$

$A \rightarrow Aa \mid b$

$B \rightarrow Bb \mid a \mid \in$

is a left linear grammar. Note that $\in$ and string of terminals can appear on RHS of any production and if non - terminal is present on L. H. S of any production, only one non - terminal should be present and it has to be the left most symbol on L. H. S.

Example 3 :

Consider the grammar

S → aA
A → aB | b
B → Ab | a

In this grammar, each production is either left linear or right linear. But, the grammar is not either left linear or right linear. Such type of grammar is called linear grammar. So, a grammar which has at most one non terminal on the right side of any production without restriction on the position of this non - terminal (note the non - terminal can be leftmost or right most) is called linear grammar.

Note that the language generated from the regular grammar is called regular language. So, there should be some relation between the regular grammar and the FA, since, the language accepted by FA is also regular language. So, we can construct a finite automaton given a regular grammar.

4.2 FA FROM REGULAR GRAMMAR

Theorem : Let G = (V, T, P, S) be a right linear grammar. Then there exists a language L(G) which is accepted by a FA. i. e., the language generated from the regular grammar is regular language.

Proof : Let $V = (q_0, q_1, ...)$ be the variables and the start state $S = q_0$ Let the productions in the grammar be

$$q_0 \rightarrow x_1 q_1$$
$$q_1 \rightarrow x_2 q_2$$
$$q_3 \rightarrow x_3 q_3$$
$$\vdots$$
$$q_n \rightarrow x_n q_n$$

Assume that the language L(G) generated from these productions is w. Corresponding to each production in the grammar we can have a equivalent transitions in the FA to accept the string w. After accepting the string w, the FA will be in the final state. The procedure to obtain FA from these productions is given below :

Step 1 : q_0 which is the start symbol in the grammar is the start state of FA.

Step 2 : For each production of the form

$$q_i \rightarrow wq_j$$

the corresponding transition defined will be

$$\delta^*(q_i, w) = q_j;$$

Step 3 : For each production of the form $q_i \rightarrow w$

the corresponding transition defined will be $\delta^*(q_i, w) = q_f$, where q_f is the final state,

As the string $w \in L(G)$ is also accepted by FA, by applying the transitions obtained from step1 through step3, the language is regular. So, the theorem is proved.

Example 1 : Construct a DFA to accept the language generated by the following grammar

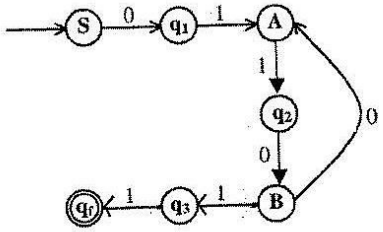
$$S \rightarrow 01A$$
$$A \rightarrow 10B$$
$$B \rightarrow 0A \mid 11$$

Solution :

Note that for each production of the form A $\rightarrow$ wB, the corresponding transition will be $\delta(A, w) = B$. Also, for each production $A \rightarrow w$, we can introduce the transition $\delta(A, w) = q_f$ where q_f is the final state. The transitions obtained from grammar G is shown using the following table :

Productions			Transitions
S	$\rightarrow$	01A	$\delta(S, 01) = A$
A	$\rightarrow$	10B	$\delta(A, 10) = B$
B	$\rightarrow$	0A	$\delta(B, 0) = A$
B	$\rightarrow$	11	$\delta(B, 11) = q_f$

The FA corresponding to the transitions obtained is shown below :



So, the DFA $M = (Q, \Sigma, \delta, q_0, A)$ where

$Q = \{ S, A, B, q_f, q_1, q_2, q_3 \}$, $\Sigma = \{0,1\}$

$q_0 = S$, $A = \{q_f\}$

δ is as obtained from the above table.

The additional vertices introduced are q_1, q_2, q_3.

Example 2 : Construct a DFA to accept the language generated by the following grammar .

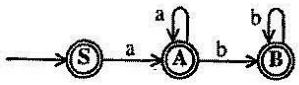
S → aA | ∈
A → aA| bB | ∈
B → bB | ∈

Solution :

Note that for each production of the form $A \to wB$, the corresponding transition will be $\delta(A,w) = B$. Also, for each production $A \to w$, we can introduce the transition $\delta(A,w) = q_f$ where q_f is the final state. The transitions obtained from grammar G is shown using the following table :

Productions			Transitions
S	→	aA	$\delta(S,a) = A$
S	→	∈	S is the final state
A	→	aA	$\delta(A,a) = A$
A	→	bB	$\delta(A,b) = B$
A	→	∈	A is the final state
B	→	bB	$\delta(B,b) = B$
B	→	∈	B is the final state.

Note : For each transition of the form $A \to \in$, make A as the final state.
The FA corresponding to the transitions obtained is shown below :



So, the DFA $M = (Q, \Sigma, \delta, q_0, A)$ where

$Q = \{S, A, B\}$, $\Sigma = \{a, b\}$

$q_0 = S$, $A = \{S, A, B\}$

δ is as obtained from the above table .

4.3 REGULAR GRAMMAR FROM FA

Theorem : Let $M = (Q, \Sigma, \delta, q_0, A)$ be a finite automaton. If L is the regular language accepted by FA, then there exists a right linear grammar G = (V, T, P, S) so that L = L(G).

Proof : Let $M = (Q, \Sigma, \delta, q_0, A)$ be a finite automata accepting L where

$Q = \{q_0, q_1, \ldots q_n\}$

$\Sigma = \{a_1, a_2, \ldots a_m\}$

A regular grammar G = (V, T, P, S) can be constructed where

$V = \{q_0, q_1, \ldots q_n\}$

$T = \Sigma$

$S = q_0$

The productions P from the transitions can be obtained as shown below :

Step 1 : For each transition of the form $\delta(q_i, a) = q_j$

the corresponding production defined will be $q_i \to aq_j$

Step 2 : If $q \in A$ i. e., if q is the final state in FA, then introduce the production

$q \to \in$

As these productions are obtained from the transitions defined for FA, the language accepted by FA is also accepted by the grammar.

UNIT-3

REGULAR GRAMMARS

After going through this chapter, you should be able to understand :

- Regular Grammar
- Equivalence between Regular Grammar and FA
- Interconversion

4.1 REGULAR GRAMMAR

Definition : The grammar G = (V, T, P, S) is said to be regular grammar iff the grammar is right linear or left linear.

A grammar G is said to be right linear if all the productions are of the form

$A \rightarrow wB$ and / or $A \rightarrow w$ where $A, B \in V$ and $w \in T^*$.

A grammar G is said to be left linear if all the productions are of the form

$A \rightarrow Bw$ and / or $A \rightarrow w$ where $A, B \in V$ and $w \in T^*$.

Example 1 : The grammar

$S \rightarrow aaB \mid bbA \mid \in$

$A \rightarrow aA \mid b$

$B \rightarrow bB \mid a \mid \in$

is a right linear grammar. Note that $\in$ and string of terminals can appear on RHS of any production and if non - terminal is present on R. H. S of any production, only one non - terminal should be present and it has to be the right most symbol on R. H. S.

Example 2 :

The grammar

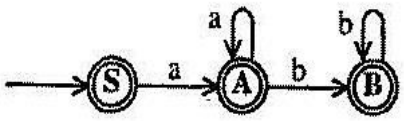
$S \rightarrow Baa \mid Abb \mid \in$

$A \rightarrow Aa \mid b$

$B \rightarrow Bb \mid a \mid \in$

is a left linear grammar. Note that $\in$ and string of terminals can appear on RHS of any production and if non - terminal is present on L. H. S of any production, only one non - terminal should be present and it has to be the left most symbol on L. H. S.

Note : For each transition of the form $A \to \in$, make A as the final state.
The FA corresponding to the transitions obtained is shown below :



So, the DFA $M = (Q, \Sigma, \delta, q_0, A)$ where

$Q = \{S, A, B\}$, $\Sigma = \{a, b\}$

$q_0 = S$, $A = \{S, A, B\}$

δ is as obtained from the above table .

4.3 REGULAR GRAMMAR FROM FA

Theorem : Let $M = (Q, \Sigma, \delta, q_0, A)$ be a finite automaton. If L is the regular language accepted by FA, then there exists a right linear grammar G = (V, T, P, S) so that L = L(G).

Proof : Let $M = (Q, \Sigma, \delta, q_0, A)$ be a finite automata accepting L where

$Q = \{q_0, q_1, \ldots q_n\}$

$\Sigma = \{a_1, a_2, \ldots a_m\}$

A regular grammar G = (V, T, P, S) can be constructed where

$V = \{q_0, q_1, \ldots q_n\}$

$T = \Sigma$

$S = q_0$

The productions P from the transitions can be obtained as shown below :

Step 1 : For each transition of the form $\delta(q_i, a) = q_j$

the corresponding production defined will be $q_i \to aq_j$

Step 2 : If $q \in A$ i. e., if q is the final state in FA, then introduce the production

$q \to \in$

As these productions are obtained from the transitions defined for FA, the language accepted by FA is also accepted by the grammar.

CONTEXT FREE GRAMMARS

After going through this chapter, you should be able to understand :

- Context free grammars
- Left most and Rightmost derivation of strings
- Derivation Trees
- Ambiguity in CFGs
- Minimization of CFGs
- Normal Forms (CNF & GNF)
- Pumping Lemma for CFLs
- Enumeration properties of CFLs

5.1 CONTEXT FREE GRAMMARS

A grammar $G = (V, T, P, S)$ is said to be a CFG if the productions of G are of the form :

$$A \to \alpha, \text{ where } \alpha \in (V \cup T)^*$$

The right hand side of a CFG is not restricted and it may be null or a combination of variables and terminals. The possible length of right hand sentential form ranges from 0 to ∞ i.e., $0 \leq |\alpha| \leq \infty$.

As we know that a CFG has no context neither left nor right. This is why, it is known as CONTEXT - FREE. *Many programming languages have recursive structure that can be defined by CFG's.*

Example 1 : Consider the grammar $G = (V, T, P, S)$ having productions :
$S \to aSa \mid bSb \mid \in$. Check the productions and find the language generated.

Solution :

Let $P_1 : S \to aSa$ (RHS is terminal variable terminal)

$P_2 : S \to bSb$ (RHS is terminal variable terminal)

$P_3 : S \to \in$ (RHS is null string)

Since, all productions are of the form $A \to \alpha$, where $\alpha \in (V \cup T)^*$, hence G is a CFG.

So, the final grammar to generate the language $L = \{ w \mid n_a(w) = n_b(w) \}$ is G = (V, T, P, S) where

$$V = \{ S \} \quad , \quad T = \{ a, b \}$$

$$P = \{ \; S \to \in \quad S \to aSb \quad S \to bSa \quad S \to SS \; \} \quad S \text{ is the start symbol}$$

5 . 2 LEFTMOST AND RIGHTMOST DERIVATIONS

Leftmost derivation :

If $G = (V, T, P, S)$ is a CFG and $w \in L(G)$ then a derivation $S \overset{*}{\underset{L}{\Rightarrow}} w$ is called leftmost derivation if and only if all steps involved in derivation have leftmost variable replacement only.

Rightmost derivation :

If $G = (V, T, P, S)$ is a CFG and $w \in L(G)$, then a derivation $S \overset{*}{\underset{R}{\Rightarrow}} w$ is called rightmost derivation if and only if all steps involved in derivation have rightmost variable replacement only.

Example 1 : Consider the grammar $S \to S + S \mid S * S \mid a \mid b$. Find leftmost and rightmost derivations for string $w = a * a + b$.

Solution :

Leftmost derivation for $w = a * a + b$

$S \underset{L}{\Rightarrow} S * S$ (Using $S \to S * S$)

$\underset{L}{\Rightarrow} a * S$ (The first left hand symbol is a, so using $S \to a$)

$\underset{L}{\Rightarrow} a * S + S$ (Using $S \to S + S$, in order to get $a + b$)

$\underset{L}{\Rightarrow} a * a + S$ (Second symbol from the left is a, so using $S \to a$)

$\underset{L}{\Rightarrow} a * a + b$ (The last symbol from the left is b, so using $S \to b$)

Rightmost derivation for $w = a * a + b$

$S \underset{R}{\Rightarrow} S * S$ (Using $S \rightarrow S * S$)

$\underset{R}{\Rightarrow} S * S + S$ (Since, in the above sentential form second symbol from the right is * so, we can not use $S \rightarrow a|b$. Therefore, we use $S \rightarrow S + S$)

$\underset{R}{\Rightarrow} S * S + b$ (Using $S \rightarrow b$)

$\underset{R}{\Rightarrow} S * a + b$ (Using $S \rightarrow a$)

$\underset{R}{\Rightarrow} a * a + b$ (Using $S \rightarrow a$)

Example 2 : Consider a CFG $S \rightarrow bA|aB,\ A \rightarrow aS|aAA|a,\ B \rightarrow bS|aBB|b$. Find leftmost and rightmost derivations for $w = aaabbabbba$.

Solution :

Leftmost derivation for $w = aaabbabbba$:

$S \Rightarrow aB$ (Using $S \rightarrow aB$ to generate first symbol of w)

$\Rightarrow aaBB$ (Since, second symbol is a, so we use $B \rightarrow aBB$)

$\Rightarrow aaaBBB$ (Since, third symbol is a, so we use $B \rightarrow aBB$)

$\Rightarrow aaabBB$ (Since fourth symbol is b, so we use $B \rightarrow b$)

$\Rightarrow aaabbB$ (Since, fifth symbol is b, so we use $B \rightarrow b$)

$\Rightarrow aaabbaBB$ (Since, sixth symbol is a, so we use $B \rightarrow aBB$)

$\Rightarrow aaabbabB$ (Since, seventh symbol is b, so we use $B \rightarrow b$)

$\Rightarrow aaabbabbS$ (Since, eighth symbol is b, so we use $B \rightarrow bS$)

$\Rightarrow aaabbabbbA$ (Since, ninth symbol is b, so we use $S \rightarrow bA$)

$\Rightarrow aaabbabbba$ (Since, the tenth symbol is a, so using $A \rightarrow a$)

Rightmost derivation for $w = aaabbabbba$

$S \Rightarrow aB$ (Using $S \rightarrow aB$ to generate first symbol of w)

$\Rightarrow aaBB$ (We need a as the rightmost symbol and second symbol from the left side, so we use $B \rightarrow aBB$)

$\Rightarrow aaBbS$ (We need a as rightmost symbol and this is obtained from A only, we use $B \rightarrow bS$)

$\Rightarrow aaBbbA$ (Using $S \rightarrow bA$)

$\Rightarrow aaBbba$ (Using $A \rightarrow a$)

$\Rightarrow aaaBBbba$ (We need b as the fourth symbol from the right)

$\Rightarrow aaaBbbba$ (Using $B \rightarrow b$)

$\Rightarrow aaabSbbba$ (Using $B \rightarrow bS$)

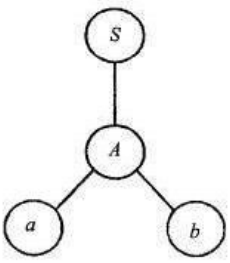


Figure (c) Parse tree for $w = ab$

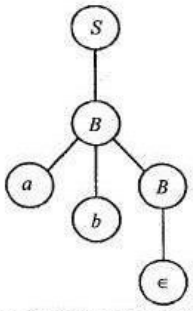


Figure (d) Parse tree for $w = ab$

So, the given grammar is ambiguous.

5.4.1 Removal of Ambiguity

5.4.1.1 Left Recursion

A grammar can be changed from one form to another accepting the same language. If a grammar has left recursive property, it is undesirable and left recursion should be eliminated. The left recursion is defined as follows.

Definition : A grammar G is said to be left recursive if there is some non terminal A such that $A \Rightarrow^{+} A\alpha$. In other words, in the derivation process starting from any non - terminal A, if a sentential form starts with the same non - terminal A, then we say that the grammar is having left recursion.

Elimination of Left Recursion

The left recursion in a grammar G can be eliminated as shown below. Consider the A - production of the form $A \rightarrow A\alpha_1 | A\alpha_2 | A\alpha_3 ..... A\alpha_n | \beta_1 | \beta_2 | \beta_3 ..... \beta_m$

where β_i's do not start with A. Then the A productions can be replaced by

$$A \rightarrow \beta_1 A^1 | \beta_2 A^1 | \beta_3 A^1 | ..... \beta_m A^1$$

$$A^1 \rightarrow \alpha_1 A^1 | \alpha_2 A^1 | \alpha_3 A^1 | ..... | \alpha_n A^1 | \in$$

Note that α_i's do not start with A^1.

Example 1 : Eliminate left recursion from the following grammar

$$E \rightarrow E + T | T$$

$$T \rightarrow T * F | F$$

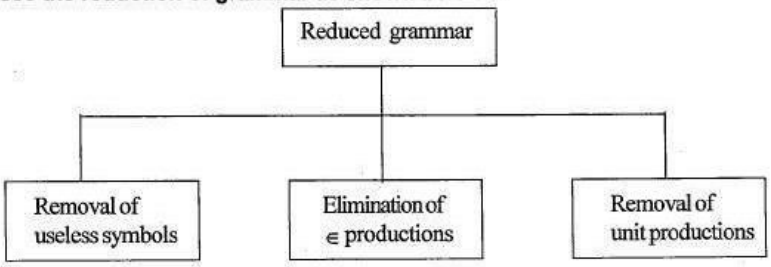
$$F \rightarrow (E) | id$$

5.5 MINIMIZATION OF CFGs

As we have seen various languages can effectively be represented by context free grammar. All the grammars are not always optimized. That means grammar may consists of some extra symbols (non - terminals). Having extra symbols unnecessary increases the length of grammar. Simplification of grammar means reduction of grammar by removing useless symbols. The properties of reduced grammar are given below :

1. Each variable (i. e. non - terminal) and each terminal of G appears in the derivation of some word in L.
2. There should not be any production as $X \to Y$ where X and Y are non - terminals.
3. If $\in$ is not in the language L then there need not be the production $X \to \in$.

We see the reduction of grammar as shown below :



5.5.1 Removal of useless symbols

Definition : A symbol X is useful if there is a derivation of the form

$$S \Rightarrow^* \alpha X \beta \Rightarrow^* w$$

Otherwise, the symbol X is useless. Note that in a derivation, finally we should get string of terminals and all these symbols must be reachable from the start symbol S. Those symbols and productions which are not at all used in the derivation are useless.

Theorem 5.5.1 : Let G = (V, T, P, S) be a CFG. We can find an equivalent grammar $G_1 = (V_1, T_1, P_1, S)$ such that for each A in $(V_1 \cup T_1)$ there exists α and β in $(V_1 \cup T_1)^*$ and x in T^+ for which $S \Rightarrow^* \alpha A \beta \Rightarrow^* x$.

P_1	T_1	V_1
-	-	S
S → a | Bb | Aa	a, b	S, A, B
A → a B	a, b	S, A, B
B → a | A a	a, b	S, A, B

The resulting grammar $G_1 = (V_1, T_1, P_1, S)$ where

V_1 = { S, A, B }

T_1 = { a, b }

P_1 = {

S → a | Bb | aA

A → aB

B → a | Aa

} S is the start symbol

such that each symbol X in $(V_1 \cup T_1)$ has a derivation of the form $S \Rightarrow^* \alpha X \beta \Rightarrow^* w$.

5.5.2 Eliminating ϵ - productions

A production of the form $A \rightarrow \epsilon$ is undesirable in a CFG, unless an empty string is derived from the start symbol. Suppose, the language generated from a grammar G does not derive any empty string and the grammar consists of ϵ- productions. Such ϵ - productions can be removed. An ϵ - production is defined as follows :

Definition 1 : Let G = (V, T, P, S) be a CFG. A production in P of the form

$$A \rightarrow \epsilon$$

is called an ϵ - production or NULL production. After applying the production the variable A is erased. For each A in V, if there is a derivation of the form

$$A \Rightarrow^* \epsilon$$

then A is a nullable variable.

Example : Consider the grammar

S → ABCa | bD

A → BC | b

B → b | ϵ

Step 2 : Construction of productions P_1 . Add a non $\in$- production in P to P_1 . Take all the combinations of nullable variables in a production, delete subset of nullable variables one by one and add the resulting productions to P_1 .

Productions	Resulting productions (P_1)
S → BAAB	S → BAAB | AAB | BAB | BAA | AB | BB | BA | AA | A | B
A → 0A 2	A → 0A2 | 02
A → 2A0	A → 2A0 | 20
B → AB	B → AB | B | A
B → 1 B	B → 1 B | 1

We can delete the productions of the form A → A. In P_1 , the production $B \to B$ can be deleted and the final grammar obtained after eliminating $\in$ -productions is shown below.

The grammar $G_1 = (V_1, T_1, P_1, S)$ where

V_1 = { S, A, B, C, D }

T_1 = { a, b, c, d }

P_1 = { S → BAAB \| AAB \| BAB \| BAA \| AB \| BB \| BA \| AA \| A \| B
A → 0A2 \| 02 \| 2A0 \| 20
B → AB \| A \| 1B \| 1
} S is the start symbol

5.5.3 Eliminating unit productions

Consider the production $A \to B$. The left hand side of the production and right hand side of the production contains only one variable. Such productions are called unit productions. Formally, a unit production is defined as follows.

Definition : Let G = (V, T, P, S) be a CFG. Any production in G of the form

$$A \to B$$

where A, $B \in V$ is a unit production.
In any grammar, the unit productions are undesirable. This is because one variable is simply replaced by another variable.

In a CFG, there is no restriction on the right hand side of a production. The restrictions are imposed on the right hand side of productions in a CFG resulting in normal forms. The different normal forms are :

1. Chomsky Normal Form (CNF)
2. Greiback Normal Form (GNF)

5.6.1 Chomsky Normal Form (CNF)

Chomsky normal form can be defined as follows.

Non - terminal → Non - terminal.Non - terminal
Non - terminal → terminal

The given CFG should be converted in the above format then we can say that the grammar is in CNF. Before converting the grammar into CNF it should be in reduced form. That means remove all the useless symbols, ϵ productions and unit productions from it. Thus this reduced grammar can be then converted to CNF.

Definition :
Let G = (V, T, P, S) be a CFG. The grammar G is said to be in CNF if all productions are of the form

$$A \rightarrow BC$$
or
$$A \rightarrow a$$

where A, B and $C \in V$ and $a \in T$.

Note that if a grammar is in CNF, the right hand side of the production should contain two symbols or one symbol. If there are two symbols on the right hand side those two symbols must be non - terminals and if there is only one symbol, that symbol must be a terminal.

Theorem 5.6.1 : Let G = (V, T, P, S) be a CFG which generates context free language without ϵ. We can find an equivalent context free grammar $G_1 = (V_1, T, P_1, S)$ in CNF such that $L(G) = L(G_1)$ i. e., all productions in G_1 are of the form

$$A \rightarrow BC$$
or
$$A \rightarrow a$$

Thus, from (7), (8) and (9), the resultant grammar becomes :

$$S \to V_1\ S \mid V_2 V_5 V_6 \mid a \mid b$$
$$V_1 \to -$$
$$V_2 \to [$$
$$V_5 \to SV_3 \qquad \text{.....(C)}$$
$$V_6 \to SV_4$$
$$V_3 \to \uparrow$$
$$V_4 \to]$$

Now, in the resultant grammar (C), following is the production which is not in the form of CNF:

$$S \to V_2 V_5 V_6$$

We can write this production as :

$$S \to V_2 V_7 \qquad \text{.....(10)}$$
$$V_7 \to V_5 V_6 \qquad \text{.....(11)}$$

Thus, from (10) and (11), the resultant grammar becomes :

$$S \to V_1 S \mid V_2 V_7 \mid a \mid b$$
$$V_1 \to -$$
$$V_2 \to [$$
$$V_7 \to V_5 V_6 \qquad \text{.....(D)}$$
$$V_5 \to SV_3$$
$$V_6 \to SV_4$$
$$V_3 \to \uparrow$$
$$V_4 \to]$$

Thus, the resultant grammar (D) is in the form of CNF, which is the required solution.

5.6.2 Greibach Normal form (GNF)

Greibach normal form can be defined as follows :

Non - terminal $\to$ one terminal. Any number of non - terminals

Example :

$S \to aA$ is in GNF
$S \to a$ is in GNF

From the subtree shown in figure (b), we get $S \overset{*}{\Rightarrow} aaS \in$ or $S \overset{*}{\Rightarrow} z_3 S z_4$ and considering the subtree shown in figure(c), we get $S \overset{*}{\Rightarrow} a$ or $S \overset{*}{\Rightarrow} z_2$.

The subtree shown in figure (b) can be added as many times as we like in the parse tree shown in figure (a). So, $S \overset{*}{\Rightarrow} z_3^i S z_4^i \overset{*}{\Rightarrow} z_3^i z_2 z_4^i$

Therefore, string z can be written as $uz_3z_2z_4y$ for some u and y substrings of z. The substrings z_3 and z_4 can be pumped as many times as we like. Replacing z_3, z_2 and z_4 by v, w and x respectively, we get z = uvwxy and $S \overset{*}{\Rightarrow} uv^iwx^iy$ for some i = 0, 1, 2, ...

Hence, the statement of theorem is proved.

Application of Pumping Lemma for CFLs

We use the pumping lemma to prove certain languages are not CFL. We proceed as we have seen in application of pumping lemma for regular sets and get contradiction. The result of this lemma is always negative.

Procedure for Proving Language is not Context - free

The following steps are considered to show a given language is not context - free.

Step 1 :

Suppose that L is context - free. Let 1 be the natural number obtained by using pumping lemma.

Step 2 :

Choose a string $x \in L$ such that $|x| \geq 1$ using pumping lemma principle write z = uvwxy.

Step 3 :

Find suitable i so that $uv^iwx^iy \notin L$. This is a contradiction. So L is not context - free.

Case 2 :

$v \in a^+$ and $x \in c^*$. Let $v = a^p$ and pq=n!. Pumping v and x, $(q+1)$ times, we get : $z' = uv^{q+1}wx^{q+1}y$.

In z', no. of a's will be $n - p + n! + p = n! + n$.

No.of b's in z' will remain n! + n. Hence, no. of a's = no. of b's in z'.

Similarly, in other cases, we can arrive at strings not as per specification of L.

Hence, L is not context free.

5.8 CLOSURE PROPERTIES OF CFLs

The closure properties that hold for regular languages do not always hold for context free languages. Consider those operations which preserve CFL.

The purpose of these operations are to prove certain languages are CFL and certain languages are not CFL.

Context-free languages are closed under following properties.

1. Union
2. Concatenation and
3. Kleene Closure (Context-free languages **may** or **may not** close under following properties)
4. Intersection
5. Complementation

Theorem 5.8.1 : If L_1 and L_2 are two CFLs, then union of L_1 and L_2 denoted by $L_1 + L_2$ or $L_1 \cup L_2$ is also a CFL.

Proof :

Let CFG $G_1 = (V_1, T_1, P, S)$ generates L_1 and CFG $G_2 = (V_2, T_2, P, S)$ generates L_2 and $G = (V, T, P, S)$ generates $L = L_1 + L_2$.

We construct G as follows :

Step 1 : Rename the variables of CFG G_1

If $V_1 = \{S, A, B, ..., X\}$, then the renamed variables are $\{S_1, A_1, B_1, ...X_1\}$. This modification should be reflected in productions also.

Step 2 : Rename the variables of CFG G_2

If $V_2 = \{S, A, B, \ldots X\}$, then the renamed variables are $\{S_2, A_2, B_2 \ldots . X_2\}$. This modification should be reflected in production also.

Step 3 : We get of the productions of G_1 and G_2 to get productions of G as follows :

$S \rightarrow S_1 \mid S_2$, where S_1 and S_2 are starting symbols of grammars G_1 and G_2 respectively and S_1 - productions and S_2 - productions remain unchanged.

$T = T_1 \cup T_2$,

$V = \{S_1, A_1, B_1, \ldots X_1\} \cup \{S_2, A_2, B_2, \ldots X_2\}$

Since, all productions of G_1 and G_2 including $S \rightarrow S_1 \mid S_2$ are in context-free form, so G is a CFG.

Language generated by *G* :

$L(G)$ = Language generated from (S_1 *or* S_2)

= Language generated from S_1 or language generated from S_2

= $L(G_1)$ or $L(G_2)$ (Since, S_1 and S_2 are starting symbols of G_1 and G_2 respectively.)

= L_1 *or* L_2 (Since, G_1 produces L_1 and G_2 produces L_2.)

= $L_1 + L_2$

Hence, statement of the theorem is proved.

Example : Consider the CFGs $S \rightarrow aSb \mid ab$ and $S \rightarrow cSdd \mid cdd$, which generate languages L_1 and L_2 respectively. Construct grammar for $L = L_1 + L_2$.

Solution :

Let G_1 generates L_1 and G_2 generates L_2 and $G = (V, T, P, S)$ generates $L = L_1 + L_2$.

Renaming the variables of G_1 and G_2, we get

$V_1 = \{S_1\}$ and $V_2 = \{S_2\}$, where S_1 - productions are $S_1 \rightarrow aS_1b \mid ab$, and S_2 - productions are $S_2 \rightarrow cS_2dd \mid cdd$

UNIT-4

PUSH DOWN AUTOMATA

After going through this chapter, you should be able to understand :

- Push down automata
- Acceptance by final state and by empty stack
- Equivalence of CFL and PDA
- Interconversion
- Introduction to DCFL and DPDA

6.1 INTRODUCTION

A PDA is an enhancement of finite automata (FA). Finite automata with a stack memory can be viewed as pushdown automata. Addition of stack memory enhances the capability of Pushdown automata as compared to finite automata. The stack memory is potentially infinite and it is a data structure. Its operation is based on last - in - first - out (LIFO). It means, the last object pushed on the stack is popped first for operation. We assume a stack is long enough and linearly arranged. We add or remove objects at the left end.

6.1.1 Model of Pushdown Automata (PDA)

A model of pushdown automata is shown in below figure. It consists of a finite tape, a reading head, which reads from the tape, a stack memory operating in LIFO fashion.

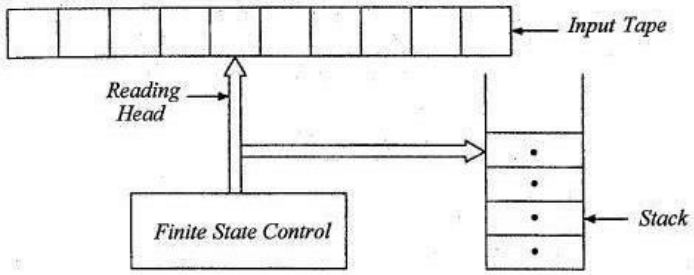


FIGURE : Model of Pushdown Automata

There are two alphabets ; one for input tape and another for stack. The stack alphabet is denoted by Γ and input alphabet is denoted by Σ. PDA reads from both the alphabets ; one symbol from the input and one symbol from the stack.

6.1.2 Mathematical Description of PDA

A pushdown automata is described by 7 - tuple $(Q,\Sigma,\Gamma,\delta,q_0,Z_0,F)$, where

1. Q is finite and nonempty set of states,
2. Σ is input alphabet,
3. Γ is finite and nonempty set of pushdown symbols,
4. δ is the transition function which maps
   From $Q \times (\Sigma \cup \{\in\}) \times \Gamma$ to (**finite subset** of) $Q \times \Gamma^*$,
5. $q_0 \in Q$, is the starting state,
6. $Z_0 \in \Gamma$, is the starting (top most or initial) stack symbol, and
7. $F \subseteq Q$, is the set of final states.

6.1.3 Moves of PDA

The move of PDA means that what are the options to proceed further after reading inputs in some state and writing some string on the stack. As we have discussed earlier that PDA is nondeterministic device having some finite number of choices of moves in each situation.

The **move** will be of two types :

1. In the first type of move, an input symbol is read from the tape, it means, the head is advanced and depending upon the topmost symbol on the stack and present state, PDA has number of choices to proceed further.
2. In the second type of move, the input symbol is not read from the tape, it means, head is not advanced and the topmost symbol of stack is used. The topmost of stack is modified without reading the input symbol. It is also known as an $\in$ - move.

Mathematically first type of move is defined as follows.

$\delta(q,a,Z) = \{(p_1,\alpha_1),(p_2,\alpha_2),\ldots(p_n,\alpha_n)\}$, where for $1 \le i \le n, q, p_i$ are states in Q, $a \in \Sigma$, $Z \in \Gamma$, *and* $\alpha_i \in \Gamma^*$.

PDA reads an input symbol a and one stack symbol Z in present state q and for any value(s) of i, enters state p_i, replaces stack symbol Z by string $\alpha_i \in \Gamma^*$, and head is advanced one cell on the tape. Now, the leftmost symbol of string α_i is assumed as the topmost symbol on the stack.

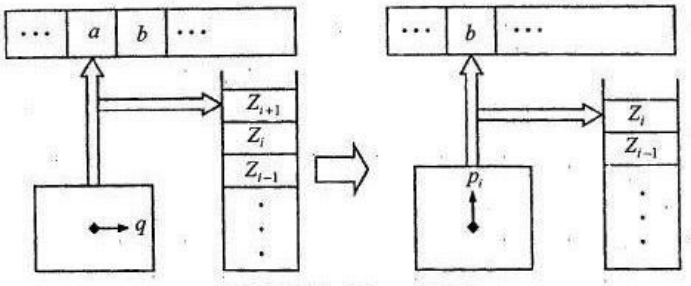
Mathematically second type of move is defined as follows.

$\delta(q,\in,Z) = \{(p_1,\alpha_1),(p_2,\alpha_2),\ldots(p_n,\alpha_n)\}$, where for $1 \le i \le n, q, p_i$ are states in $Q, a \in \Sigma$, $Z \in \Gamma$, *and* $\alpha_i \in \Gamma^*$.

PDA does not read input symbol but it reads stack symbol Z in present state q and for any value(s) of i, enters state p_i, replaces stack symbol Z by string $\alpha_i \in \Gamma^*$, and head is not advanced on the tape. Now, the leftmost symbol of string α_i is assumed as the topmost symbol on the stack.

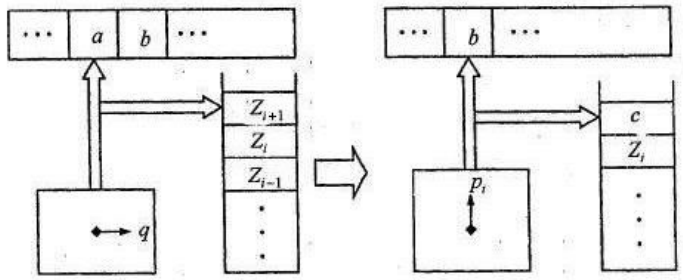
The **string** α_i be any one of the following :

1. $\alpha_i = \in$ in this case the topmost stack symbol Z_{i+1} is erased and second topmost symbol becomes the topmost symbol in the next move. It is shown in figure (a).



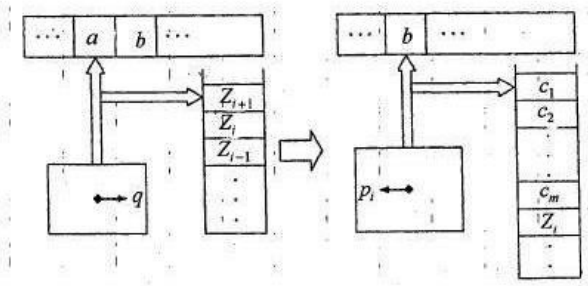
FIGURE(a): Move of PDA

2. $\alpha_i = c, c \in \Gamma$, in this case the topmost stack symbol Z_{i+1} is replaced by symbol c. It is shown in figure(b)



FIGURE(b): Move of PDA

3. $\alpha_i = c_1 c_2 \ldots c_m$, in this case the topmost stack symbol Z_{i+1} is replaced by string $c_1 c_2 \ldots c_m$. It is shown in figure(c).



FIGURE(c): Move of PDA

6.1.4 Instantaneous Description (ID) of PDA

Let PDA $M = (Q,\Sigma,\Gamma,\delta,q_0,Z_0,F)$, then its configuration at a given instant can be defined by instantaneous description (ID). An ID includes state, remaining input string, and remaining stack string (symbols). So, an ID is (q,x,α), where $q \in Q, x \in \Sigma^*, \alpha \in \Gamma^*$.

The relation between two consecutive IDs is represented by the sign $\vdash$.

We say $(q,ax,Z\beta) \vdash_M (p,x,\alpha\beta)$ if $\delta(q,a,Z)$ contains (p,α), where $Z,\beta,\alpha \in \Gamma^*$, a may be null or $a \in \Sigma$, $p, q \in Q$ for M

The reflexive and transitive closure of the relation $\vdash_M$ is denoted by $\vdash_M^*$

Properties :

1. If $(q,x,\alpha) \vdash_M^* (p,\in,\alpha)$, where $\alpha \in \Gamma^*, x \in \Sigma^*$, and $p, q \in Q$, then for all $y \in \Sigma^*$. $(q,xy,\alpha) \vdash_M^* (p,y,\alpha)$,
2. If $(q,xy,\alpha) \vdash_M^* (p,y,\alpha)$, where $\alpha \in \Gamma^*, x,y \in \Sigma^*$, and $p,q \in Q$, then $(q,x,\alpha) \vdash_M^* (p,\in,\alpha)$, and
3. If $(q,x,\alpha) \vdash_M^* (p,\in,\beta)$, where $\alpha,\beta \in \Gamma^*, x \in \Sigma^*$, and $p,q \in Q$, then $(q,x,\alpha\gamma) \vdash_M^* (p,\in,\beta\gamma)$, where $\gamma \in \Gamma^*$

6.1.5 Acceptance by PDA

Let M be a PDA, the accepted language is represented by N(M). We defined the acceptance by PDA in two ways.

1. Let $M = (Q,\Sigma,\Gamma,\delta, q_0, Z_0, F)$, then N(M) is accepted by final state such that

   $N(M)=\{w:(q_0,w,Z_0)\vdash_M^*(q_f,\in,\beta)$, where $q \in Q$, $w \in \Sigma^*, Z_0, \beta \in \Gamma^*$, and $q_f \in F\}$

   It is similar to the acceptance by FA discussed earlier. We define some final states and the accepted language $N(M)$ is the set of all input strings for which some choice of moves leads to some final state.

2. Let $M = (Q,\Sigma,\Gamma,\delta,q_0,Z_0,\phi)$, then $N(M)$ is accepted by empty stack or null stack such that $N(M) = \{w:(q_0,w,Z_0)\vdash_M^*(p,\in,\in)$, where $p \in Q, w \in \Sigma^*\}$

   The language $N(M)$ is the set of all input strings for which some sequence of moves causes the PDA to empty its stack.

Note : If acceptance is defined by empty stack then there is no meaning of final state and it is represented by ϕ.

Example : consider a PDA $M = (\{q_0,q_1,q_2\},\{a,c\},\{a,Z_0\},\delta,q_0,Z_0,\{q_2\})$ shown in below figure. Check the acceptability of string aacaa.

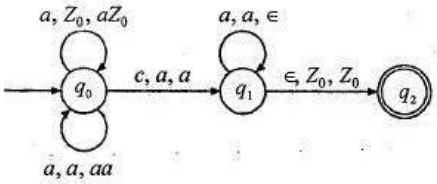


FIGURE : PDA accepting $\{a^n c a^n : n \geq 1\}$

Note : *Edges are labeled with Input symbol, stack symbol, written symbol on the stack.*

Solution :

The transition function δ is defined as follows :

$\delta(q_0,a,Z_0) = \{(q_0,aZ_0)\}$,

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

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

$\delta(q_1,a,a) = \{(q_1,\in)\}$, and

$\delta(q_1,\in,Z_0) = \{(q_2,Z_0)\}$

Following moves are carried out in order to check acceptability of string $aacaa$:

$$(q_0,aacaa,Z_0) \vdash (q_0,acaa,aZ_0)$$
$$\vdash (q_0,caa,aaZ_0)$$
$$\vdash (q_1,aa,aaZ_0)$$
$$\vdash (q_1,a,aZ_0)$$
$$\vdash (q_1,\in,Z_0)$$
$$\vdash (q_2,\in,Z_0)$$

Hence, $(q_0,aacaa,Z_0) \overset{*}{\vdash}_M (q_2,\in,Z_0)$.

Therefore, the string **aacaa** is accepted by M.

6. 2 CONSTRUCTION OF PDA

In this section, we shall see how PDA's can be constructed.

Example 1 : Obtain a PDA to accept the language $L(M) = \{ wCw^R \mid w \in (a+b)^* \}$ where W^R is reverse of W .

Solution:

It is clear from the language $L(M) = \{ wCw^R \}$ that if $w = abb$

then reverse of w denoted by W^R will be $W^R = bba$ and the language L will be wCw^R
i. e., abbCbba which is a string of palindrome.

To accept the string :

The sequence of moves made by the PDA for the string **aabCbaa** is shown below.

Initial ID

$(q_0, aabCbaa, Z_0) \vdash (q_0, abCbaa, aZ_0)$

$\vdash (q_0, bCbaa, aaZ_0)$

$\vdash (q_0, Cbaa, baaZ_0)$

$\vdash (q_1, baa, baaZ_0)$

$\vdash (q_1, aa, aaZ_0)$

$\vdash (q_1, a, aZ_0)$

$\vdash (q_1, \in, Z_0)$

$\vdash (q_2, \in, Z_0)$

(Final Configuration)

Since q_2 is the final state and input string is $\in$ in the final configuration, the string **aabCbaa** is accepted by the PDA.

To reject the string :

The sequence of moves made by the PDA for the string **aabCbab** is shown below.

Initial ID

$(q_0, aabCbab, Z_0) \vdash (q_0, abCbab, aZ_0)$

$\vdash (q_0, bCbab, aaZ_0)$

$\vdash (q_0, Cbab, baaZ_0)$

$\vdash (q_1, bab, baaZ_0)$

$\vdash (q_1, ab, aaZ_0)$

$\vdash (q_1, b, aZ_0)$

(Final Configuration)

Since the transition $\delta(q_1, b, a)$ is not defined, the string **aabCbab** is not a palindrome and the machine halts and the string is rejected by the PDA.

Example 2 : Obtain a PDA to accept the language $L = \{ a^n b^n \mid n \geq 1 \}$ by a final state.

Solution :

The machine should accept n number of a's followed by n number of b's.

6.3 DETERMINISTIC AND NONDETERMINISTIC PUSHDOWN AUTOMATA

In this section, we will discuss about the deterministic and nondeterministic behavior of pushdown automata.

6.3.1 Nondeterministic PDA (NPDA)

Like NFA, nondeterministic PDA (NPDA) has finite number of choices for its inputs. As we have discussed in the mathematical description that transition function δ which maps from $Q \times (\Sigma \cup \{\in\}) \times \Gamma$ to (**finite subset** of) $Q \times \Gamma^*$. A nondeterministic PDA accepts an input if a sequence of choices leads to some final state or causes PDA to empty its stack. Since, sometimes it has more than one choice to move further on a particular input ; it means, PDA guesses the right choice always, otherwise it will fail and will be in hang state.

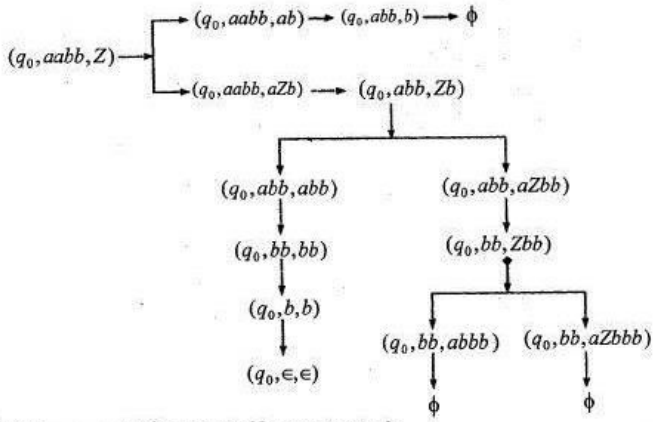
Example : consider a nondeterministic PDA $M = (\{q_0\},\{a,b\},\{a,b,Z\},\delta,q_0,Z,\phi)$, for the language $L = \{a^n b^n : n \geq 1\}$, where δ is defined as follows :

$\delta(q_0,\in,Z) = \{(q_0,ab),(q_0,aZb)\}$ (Two possible moves for input $\in$ on the tape and Z on the stack),

$\delta(q_0,a,a) = \{(q_0,\in)\}$, and $\delta(q_0,b,b) = \{(q_0,\in)\}$

Check whether string $w = aabb$ is accepted or not ?

Solution : Initial configuration is $(q_0,aabb,Z)$. Following moves are possible :



Hence, $w = aabb$ is accepted by empty stack.

One thing is noticeable here that only one move sequence leads to empty store and other don't. In other words, we say that some move sequence(s) leads to accepting configuration and other lead to hang state.

6.3.2 Deterministic PDA (DPDA)

Deterministic PDA (DPDA) is just like DFA, which has *at most one choice* to move for certain input. A PDA $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ is deterministic if it satisfies both the conditions given as follows :

1. For any $q \in Q$, $a \in (\Sigma \cup \{\in\})$, and $Z \in \Gamma$, $\delta(q, a, Z)$ has at most one choice of move.
2. For any $q \in Q$, and $Z \in \Gamma$, if $\delta(q, \in, Z)$ is defined i.e. $\delta(q, \in, Z) \neq \phi$, then $\delta(q, a, Z) = \phi$ for all $a \in \Sigma$

Example : Consider a DPDA $M = (\{q_0, q_1\}, \{a, c\}, \{a, Z_0\}, \delta, q_0, Z_0, \phi)$ accepting the language $\{a^n c a^n : n \geq 1\}$, where δ is defined as follows :

$\delta(q_0, a, Z_0) = \{(q_0, aZ_0)\}$

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

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

$\delta(q_1, a, a) = \{(q_1, \in)\}$, and $\delta(q_1, \in, Z_0) = \{(q_1, \in)\}$

Check whether the string $w = aacaa$ is accepted by empty stack or not ?

Solution :

We see that in each transition DPDA has at most one move. Initial configuration is $(q_0, aacaa, Z_0)$. Following are the possible moves.

$$(q_0, aacaa, Z_0) \rightarrow (q_0, acaa, aZ_0) \rightarrow (q_0, caa, aaZ_0) \rightarrow (q_1, aa, aaZ_0)$$

$$\downarrow$$

$$(q_1, \in, \in) \leftarrow (q_1, \in, Z_0) \leftarrow (q_1, a, aZ_0)$$

Hence, the string $w = aacaa$ is accepted by empty stack.

As we have discussed in earlier chapters that DFA and NFA are equivalent with respect to the language acceptance, but the same is not true for the PDA.

For example, language $L = \{ww^R : w \in (a \cup b)^*\}$ is accepted by nondeterministic PDA, can not by any deterministic PDA. A nondeterministic PDA can not be converted into equivalent deterministic PDA, but all DCFLs which are accepted by DPDA, are also accepted by NPDA. So, we say that deterministic PDA is a proper subset of nondeterministic PDA. Hence, the power of nondeterministic PDA is more as compared to deterministic PDA.

6.4 ACCEPTANCE OF LANGUAGE BY PDA

The language can be accepted by a Push Down Automata using two approaches.

1. **Acceptance by Final State :** The PDA accepts its input by consuming it and then it enters in the final state.

2. **Acceptance by empty stack :** On reading the input string from initial configuration for some PDA, the stack of PDA gets empty.

6.4.1 Equivalence of Empty Store and Final state acceptance

Theorem:

If $M_1 = (Q_1, \Sigma, \Gamma_1, \delta_1, p_1, Z_1, \phi)$ is a PDA accepting CFL L by empty store then there exists PDA $M_2 = (Q_2, \Sigma, \Gamma_2, \delta_2, p_2, Z_2, \{q_f\})$ which accepts L by final state.

Proof :

First we construct PDA M_2 based on PDA M_1 and then we prove that both accept L.

Step 1 : Construction of PDA M_2 based on given PDA M_1

Σ is same for both PDAs. We add a new initial state and a new final state with given PDA M_1.

So, $Q_2 = Q_1 \cup \{p_2 \cup q_f\}$

The stack alphabet Γ_2 of PDA M_2 contains one additional symbol Z_2 with Γ_1.

So, $\Gamma_2 = \Gamma_1 \cup \{Z_2\}$

The transition function δ_2 contains all the transitions of given PDA M_1 and two additional transitions (R_1 *and* R_3) as defined as follows :

$R_1 : \delta_2(p_2, \in, Z_2) = \{(p_1, Z_1 Z_2)\}$,

$R_2 : \delta_2(q, a, Z) = \delta_1(q, a, Z)$ for all (q, a, Z) in $Q_1 \times (\Sigma \cup \{\in\}) \times \Gamma_1$
(the original transitions of M_1), and

$R_3 : \delta_2(q, \in, Z_2) = \{(q_f, \in)\}$ for all $q \in Q_1$

By the R_1, M_2 moves from its initial ID $(p_2, \in, Z_2)$ to the initial ID of M_1. By R_2, M_2 uses all the transitions of M_1 after reaching the initial ID of M_1 and by using R_3 M_2 reaches the final state q_f.

The block diagram is shown in below figure.

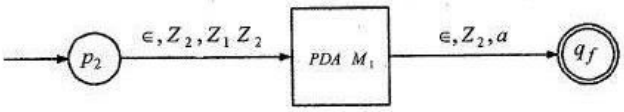


FIGURE : Block diagram of PDA M_2

Step 2 : The language accepted by PDA M_1 and PDA M_2

The behaviors of M_1 and M_2 are same except the two by $\in$ - moves defined by R_1 *and* R_3.

Let string $w \in L$ and accepted by M_1, then

$(p_1, w, Z_1) \left|\overset{*}{\overline{M_1}}\right. (q, \in, \in)$ where $q \in Q_1$ **(Result 1)**

For M_2, the initial ID is (p_2, w, Z_2) and it can be written as $(p_2, \in w \in, Z_2)$. So,

$(p_2, \in w \in, Z_2) \left|\overline{M_2}\right. (p_1, w, Z_1 Z_2)$ (This initial ID of M_1)

$\left|\overset{*}{\overline{M_2}}\right. (q, \in, Z_2)$ (by R_2 and Result 1)

$\left|\overset{*}{\overline{M_2}}\right. (q_f, \in, \alpha)$ $\alpha \in \Gamma_2^*$ (By R_3)

Thus, if M_1 accepts w, then M_2 also accepts it.

It means $L(M_2) \subseteq L(M_1)$ **(Result 2)**

Let string $w \in L$ and accepted by PDA M_2, then

$(p_2, \in w \in, Z_2)$ $\left|\overline{M_2}\right. (p_1, w, Z_1 Z_2)$ (By R_1) **(Result 3)**

$\left|\overset{*}{\overline{M_2}}\right. (q, \in, Z_2)$ (By R_2) **(Result 4)**

$\left|\overline{M_2}\right. (q_f, \in, \alpha)$ $\alpha \in \Gamma_2^*$ (By R_3)

Note : The Result 3 is the initial ID of M_1. The Result 4 shows the empty store for M_1 if symbol Z_2 is not there.

For M_1, the initial ID is (p_1, w, Z_1)

So, $(p_1, w, Z_1) \vdash_{M_2}^{*} (q, \in, \in)$, where $q \in Q_1$ (By Result 3 and Result 4) Thus, if M_2 accepts w, then M_1 also accepts it.

It means, $L(M_1) \subseteq L(M_2)$ **(Result 5)**

Therefore, $L = L(M_2) = L(M_1)$ (From Result 2 and Result 5)

Hence, the statement of theorem is proved.

Example: Consider a nondeterministic PDA $M_1 = (\{q_0\}, \{a,b\}, \{a,b,S\}, \delta, q_0, S, \phi)$ which accepts the language $L = \{a^n b^n : n \geq 1\}$ by empty store, where δ is defined as follows :

$\delta(q_0, \in, S) = \{(q_0, ab), (q_0, aSb)\}$ (Two possible moves),

$\delta(q_0, a, a) = \{(q_0, \in)\}$, and $\delta(q_0, b, b) = \{(q_0, \in)\}$

Construct an equivalent PDA M_2 which accepts L in final state and check whether string $w = aabb$ is accepted or not ?

Solution : Following moves are carried out by PDA M_1 in order to accept $w = aabb$:

$$(q_0, aabb, S) \vdash (q_0, aabb, aSb)$$
$$\vdash (q_0, abb, Sb)$$
$$\vdash (q_0, abb, abb)$$
$$\vdash (q_0, bb, bb)$$
$$\vdash (q_0, b, b)$$
$$\vdash (q_0, \in, \in)$$

Hence, $(q_0, aabb, S) \vdash_{M_1}^{*} (q_0, \in, \in)$

Therefore, $w = aabb$ is accepted by M_1.

UNIT-5

TURING MACHINES

After going through this chapter, you should be able to understand :

- Turing Machine
- Design of TM
- Computable functions
- Recursively Enumerable languages
- Church's Hypothesis & Counter machine
- Types of Turing Machines

7.1 INTRODUCTION

The Turing machine is a generalized machine which can recognize all types of languages viz, regular languages (generated from regular grammar), context free languages (generated from context free grammar) and context sensitive languages (generated from context sensitive grammar). Apart from these languages, the Turing machine also accepts the language generated from unrestricted grammar. Thus, Turing machine can accept any generalized language. This chapter mainly concentrates on building the Turing machines for any language.

7.2 TURING MACHINE MODEL

The Turing machine model is shown in below figure . It is a finite automaton connected to read - write head with the following components :

- Tape
- Read - write head
- Control unit

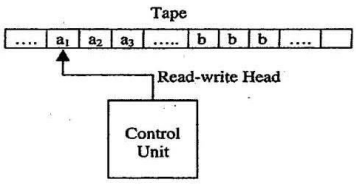


FIGURE : Turing machine model

Tape : It is a temporary storage and is divided into cells. Each cell can store the information of only one symbol. The string to be scanned will be stored from the left most position on the tape. The string to be scanned should end with infinite number of blanks.

Read - write head : The read - write head can read a symbol from where it is pointing to and it can write into the tape to where the read - write head points to.

Control Unit : The reading / writing from / to the tape is determined by the control unit. The different moves performed by the machine depends on the current scanned symbol and the current state. The read - write head can move either towards left or right i.e., movement can be on both the directions. The various moves performed by the machine are :

1. Change of state from one state to another state
2. The symbol pointing to by the read - write head can be replaced by another symbol.
3. The read - write head may move either towards left or towards right.

The Turing machine can be represented using various notations such as

- Transition table
- Instantaneous description
- Transition diagram

7.2.1 Transition Table

The table below shows the transition table for some Turing machine. Later sections describe how to obtain the transition table.

δ	Tape Symbols (Γ)				
States	a	b	X	Y	B
q_0	(q_1, X, R)	-	-	(q_3, Y, R)	-
q_1	(q_1, a, R)	(q_2, Y, L)	-	(q_1, Y, R)	-
q_2	(q_2, a, L)	-	(q_0, X, R)	(q_2, Y, L)	-
q_3	-	-	-	(q_3, Y, R)	(q_4, B, R)
q_4	-	-	-	-	-

Note that for each state q, there can be a corresponding entry for the symbol in Γ. In this table the symbols a and b are input symbols and can be denoted by the symbol Σ. Thus $\Sigma \subseteq \Gamma$ excluding the symbol B. The symbol B indicates a blank character and usually the string ends with infinite number of B's i. e., blank characters. The undefined entries indicate that there are no - transitions defined or there can be a transition to dead state. When there is a transition to the dead state, the machine halts and the input string is rejected by the machine. It is clear from the table that

$$\delta : Q \times \Gamma \text{ to } (Q \times \Gamma \times \{L, R\})$$

where $Q = \{q_0, q_1, q_2, q_3, q_4\}$; $\Sigma = \{a, b\}$

$\Gamma = \{a, b, X, Y, B\}$

q_0 is the initial state; B is a special symbol indicating blank character

$F = \{q_4\}$ which is the final state.

Thus, a Turing Machine M can be defined as follows.

Definition : The Turing Machine $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ where

Q is set of finite states

Σ is set of input alphabets

Γ is set of tape symbols

δ is transition function $Q \times \Gamma$ to $(Q \times \Gamma \times \{L, R\})$

q_0 is the initial state

B is a special symbol indicating blank character

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

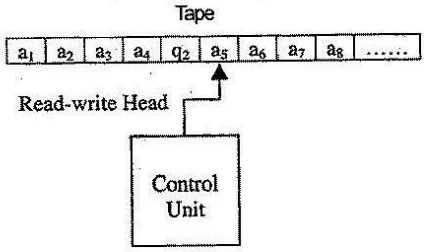
7.2.2 Instantaneous description (ID)

Unlike the ID described in PDA, in Turing machine (TM), the ID is defined on the whole string (not on the string to be scanned) and the current state of the machine.

Definition :

An ID of TM is a string in $\alpha q \beta$, where q is the current state, $\alpha \beta$ is the string made from tape symbols denoted by Γ i. e., α and $\beta \in \Gamma^*$. The read - write head points to the first character of the substring β. The initial ID is denoted by $q\alpha\beta$ where q is the start state and the read - write head points to the first symbol of α from left. The final ID is denoted by $\alpha\beta qB$ where $q \in F$ is the final state and the read - write head points to the blank character denoted by B.

Example : Consider the snapshot of a Turing machine



In this machine, each $a_i \in \Gamma$ (i. e., each a_i belongs to the tape symbol). In this snapshot, the symbol a_5 is under read - write head and the symbol towards left of a_5 i. e., q_2 is the current state. Note that, in the Turing machine, the symbol immediately towards left of the read - write head will be the current state of the machine and the symbol immediately towards right of the state will be the next symbol to be scanned. So, in this case an ID is denoted by

$$a_1a_2a_3a_4q_2a_5a_6a_7a_8.....$$

where the substring $a_1a_2a_3a_4$ towards left of the state q_2 is the left sequence, the substring $a_5a_6a_7a_8$..... towards right of the state q_2 is the right sequence and q_2 is the current state of the machine. The symbol a_5 is the next symbol to be scanned.

Assume that the current ID of the Turing machine is $a_1a_2a_3a_4q_2a_5a_6a_7a_8$..... as shown in snapshot of example.

Suppose, there is a transition $\delta(q_2, a_5) = (q_3, b_1, R)$

It means that if the machine is in state q_2 and the next symbol to be scanned is a_5, then the machine enters into state q_3 replacing the symbol a_5 by b_1 and R indicates that the read - write head is moved one symbol towards right. The new configuration obtained is

$$a_1a_2a_3a_4b_1q_3a_6a_7a_8.....$$

This can be represented by a move as $a_1a_2a_3a_4q_2a_5a_6a_7a_8.... \vdash a_1a_2a_3a_4b_1q_3a_6a_7a_8.....$

Similarly if the current ID of the Turing machine is $a_1a_2a_3a_4q_2a_5a_6a_7a_8.....$

and there is a transition

$$\delta(q_2, a_5)=(q_1, c_1, L)$$

means that if the machine is in state q_2 and the next symbol to be scanned is a_5, then the machine enters into state q_1 replacing the symbol a_5 by c_1 and L indicates that the read - write head is moved one symbol towards left. The new configuration obtained is

$$a_1a_2a_3q_1a_4c_1a_6a_7a_8.....$$

This can be represented by a move as $a_1a_2a_3a_4q_2a_5a_6a_7a_8\ldots. \vdash a_1a_2a_3q_1a_4c_1a_6a_7a_8\ldots.$

This configuration indicates that the new state is q_1, the next input symbol to be scanned is a_4. The actions performed by TM depends on

1. The current state.
2. The whole string to be scanned
3. The current position of the read - write head

The action performed by the machine consists of

1. Changing the states from one state to another
2. Replacing the symbol pointed to by the read - write head
3. Movement of the read - write head towards left or right.

7.2.3 The move of Turing Machine M can be defined as follows

Definition : Let $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ be a TM. Let the ID of M be $a_1a_2a_3\ldots\ldots a_{k-1}qa_ka_{k+1}\ldots\ldots a_n$ where $a_j \in \Gamma$ for $1 \le j \le n-1$, $q \in Q$ is the current state and a_k as the next symbol to scanned. If there is a transition $\delta(q, a_k) = (p, b, R)$ then the move of machine M will be $a_1a_2a_3\ldots\ldots a_{k-1}qa_ka_{k+1}\ldots\ldots a_n \vdash a_1a_2a_3\ldots\ldots a_{k-1}bpa_{k+1}\ldots\ldots a_n$

If there is a transition $\delta(q, a_k) = (p, b, L)$ then the move of machine M will be

$$a_1a_2a_3\ldots\ldots a_{k-1}qa_ka_{k+1}\ldots\ldots a_n \vdash a_1a_2a_3\ldots\ldots a_{k-2}pa_{k-1}ba_{k+1}\ldots\ldots a_n$$

7.2.4 Acceptance of a language by TM

The language accepted by TM is defined as follows.

Definition :

Let $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ be a TM. The language L(M) accepted by M is defined as

$$L(M) = \{w \mid q_0w \vdash^* \alpha_1\ p\ \alpha_2 \text{ where } w \in \Sigma^*, p \in F \text{ and } \alpha_1, \alpha_2 \in \Gamma^*\}$$

i.e., set of all those words w in Σ^* which causes M to move from start state q_0 to the final state p. The language accepted by TM is called recursively enumerable language.

The string w which is the string to be scanned, should end with infinite number of blanks. Initially, the machine will be in the start state q_0 with read - write head pointing to the first symbol of w from left. After some sequence of moves, if the Turing machine enters into the final state and halts, then we say that the string w is accepted by Turing machine.

7.2.5 Differences between TM and PDA

Push Down Automa :

1. A PDA is a nondeterministic finite automaton coupled with a stack that can be used to store a string of arbitrary length.
2. The stack can be read and modified only at its top.
3. A PDA chooses its next move based on its current state, the next input symbol and the symbol at the top of the stack.
4. There are two ways in which the PDA may be allowed to signal acceptance. One is by entering an accepting state, the other by emptying its stack.
5. ID consisting of the state, remaining input and stack contents to describe the "current condition" of a PDA.
6. The languages accepted by PDA's either by final state or by empty stack, are exactly the context - free languages.
7. A PDA languages lie strictly between regular languages and CSL's.

Turing Machines :

1. The TM is an abstract computing machine with the power of both real computers and of other mathematical definitions of what can be computed.
2. TM consists of a finite - state control and an infinite tape divided into cells.
3. TM makes moves based on its current state and the tape symbol at the cell scanned by the tape head.
4. The blank is one of tape symbols but not input symbol.
5. TM accepts its input if it ever enters an accepting state.
6. The languages accepted by TM's are called Recursively Enumerable (RE) languages.
7. Instantaneous description of TM describes current configuration of a TM by finite - length string.
8. Storage in the finite control helps to design a TM for a particular language.
9. A TM can simulate the storage and control of a real computer by using one tape to store all the locations and their contents.

7.3 CONSTRUCTION OF TURING MACHINE (TM)

In this section, we shall see how TMs can be constructed.

Example 1 : Obtain a Turing machine to accept the language $L = \{ 0^n 1^n \mid n \geq 1 \}$.

Solution : Note that n number of 0's should be followed by n number of 1's. For this let us take an example of the string $w = 00001111$. The string w should be accepted as it has four zeroes followed by equal number of 1's.

General Procedure :

Let q_0 be the start state and let the read - write head points to the first symbol of the string to be scanned. The general procedure to design TM for this case is shown below :

1. Replace the left most 0 by X and change the state to q_1 and then move the read - write head towards right. This is because, after a zero is replaced, we have to replace the corresponding 1 so that number of zeroes matches with number of 1's.
2. Search for the leftmost 1 and replace it by the symbol Y and move towards left (so as to obtain the leftmost 0 again). Steps 1 and 2 can be repeated.

Consider the situation

XX00YY11

↑

q_0

where first two 0's are replaced by Xs and first two 1's are replaced by Ys. In this situation, the read - write head points to the left most zero and the machine is in state q_0. With this as the configuration, now let us design the TM.

Step 1 : In state q_0, replace 0 by X, change the state to q_1 and move the pointer towards right. The transition for this can be of the form

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

The resulting configuration is shown below.

XXX0YY11

↑

q_1

Step 2 : In state q_1, we have to obtain the left - most 1 and replace it by Y. For this, let us move the pointer to point to leftmost one. When the pointer is moved towards 1, the symbols encountered may be 0 and Y. Irrespective what symbol is encountered, replace 0 by 0, Y by Y, remain in state q_1 and move the pointer towards right. The transitions for this can be of the form

$$\delta(q_1,0)=(q_1,0,R)$$
$$\delta(q_1,Y)=(q_1,Y,R)$$

When these transitions are repeatedly applied, the following configuration is obtained.

XXX0YY11

↑

q_1

Step 3 : In state q_1, if the input symbol to be scanned is a 1, then replace 1 by Y, change the state to q_2 and move the pointer towards left. The transition for this can be of the form

$$\delta(q_1,1)=(q_2,Y,L)$$

and the following configuration is obtained.

XXX0YYY1

↑

q_2

Note that the pointer is moved towards left. This is because, a zero is replaced by X and the corresponding 1 is replaced by Y. Now, we have to scan for the left most 0 again and so, the pointer was move towards left.

Step 4 : Note that to obtain leftmost zero, we need to obtain right most X first. So, we scan for the right most X. During this process we may encounter Y's and 0's . Replace Y by Y, 0 by 0, remain in state q_2 only and move the pointer towards left. The transitions for this can be of the form

$$\delta(q_2,Y)=(q_2,Y,L)$$

$$\delta(q_2,0)=(q_2,0,L)$$

The following configuration is obtained

XXX0YYY1

↑

q_2

Step 5 : Now, we have obtained the right most X. To get leftmost 0, replace X by X, change the state to q_0 and move the pointer towards right. The transition for this can be of the form

$$\delta(q_2,X)=(q_0,X,R)$$

and the following configuration is obtained

XXX0YYY1

↑

q_0

Now, repeating the steps 1 through 5, we get the configuration shown below :

XXXXYYYY

↑

q_0

Step 6 : In state q_0, if the scanned symbol is Y, it means that there are no more 0's. If there are no zeroes we should see that there are no 1's. For this we change the state to q_3, replace Y by Y and move the pointer towards right. The transition for this can be of the form

$$\delta(q_0, Y) = (q_3, Y, R)$$

and the following configuration is obtained

XXXXYYYY

↑

q_3

In state q_3, we should see that there are only Ys and no more 1's. So, as we can replace Y by Y and remain in q_3 only. The transition for this can be of the form

$$\delta(q_3, Y) = (q_3, Y, R)$$

Repeatedly applying this transition, the following configuration is obtained .

XXXXYYYYB

↑

q_3

Note that the string ends with infinite number of blanks and so, in state q_3 if we encounter the symbol B, means that end of string is encountered and there exists n number of 0's ending with n number of 1's. So, in state q_3, on input symbol B, change the state to q_4, replace B by B and move the pointer towards right and the string is accepted. The transition for this can be of the form $\delta(q_3, B) = (q_4, B, R)$

The following configuration is obtained

XXXXYYYYBB

↑

q_4

So, the Turing machine to accept the language $L = \{a^n b^n \mid n \geq 1\}$

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

where

$Q = \{q_0, q_1, q_2, q_3\}$; $\Sigma = \{0, 1\}$; $\Gamma = \{0, 1, X, Y, B\}$

$q_0 \in Q$ is the start state of machine ; $B \in \Gamma$ is the blank symbol.

$F = \{q_4\}$ is the final state.

δ is shown below.

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

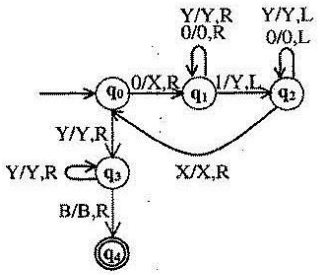
$$\delta(q_1, 0) = (q_1, 0, R)$$

$$\delta(q_1, Y) = (q_1, Y, R)$$
$$\delta(q_1, 1) = (q_2, Y, L)$$
$$\delta(q_2, Y) = (q_2, Y, L)$$
$$\delta(q_2, 0) = (q_2, 0, L)$$
$$\delta(q_2, X) = (q_0, X, R)$$
$$\delta(q_0, Y) = (q_3, Y, R)$$
$$\delta(q_3, Y) = (q_3, Y, R)$$
$$\delta(q_3, B) = (q_4, B, R)$$

The transitions can also be represented using tabular form as shown below.

δ	Tape Symbols (Γ)				
States	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	-	-	-	-	-

The transition table shown above can be represented as transition diagram as shown below :



To accept the string :

The sequence of moves or computations (IDs) for the string 0011 made by the Turing machine are shown below :

Initial ID

$q_0 0011$ $\vdash X q_1 011$ $\vdash X 0 q_1 11$

$\vdash X q_2 0Y1$ $\vdash q_2 X 0Y1$

$\vdash X q_0 0Y1$ $\vdash XX q_1 Y1$

$\vdash XXY q_1 1$ $\vdash XX q_2 YY$

$\vdash X q_2 XYY$ $\vdash XX q_0 YY$

$\vdash XXY q_3 Y$ $\vdash XXY Y q_3$

$\vdash XXYYB q_4$

(Final ID)

Example 2 : Obtain a Turing machine to accept the language $L(M) = \{ 0^n 1^n 2^n \mid n \geq 1 \}$

Solution : Note that n number of 0's are followed by n number of 1's which in turn are followed by n number of 2's. In simple terms, the solution to this problem can be stated as follows :

Replace first n number of 0's by X's, next n number of 1's by Y's and next n number of 2's by Z's. Consider the situation where in first two 0's are replaced by X's , next immediate two 1's are replaced by Y's and next two 2's are replaced by Z's as shown in figure 1(a).

XX00YY11ZZ22	XXX0YY11ZZ22	XXX0YY11ZZ22
↑	↑	↑
q_0	q_1	q_1
(a)	(b)	(c)

FIGURE 1 : Various Configurations

Now, with figure 1(a). a as the current configuration, let us design the Turing machine. In state q_0, if the next scanned symbol is 0 replace it by X, change the state to q_1 and move the pointer towards right and the situation shown in figure 1(b) is obtained . The transition for this can be of the form

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

In state q_1, we have to search for the leftmost 1. It is clear from figure 1(b) that, when we are searching for the symbol 1, we may encounter the symbols 0 or Y. So, replace 0 by 0 , Y by Y and move the pointer towards right and remain in state q_1 only. The transitions for this can be of the form

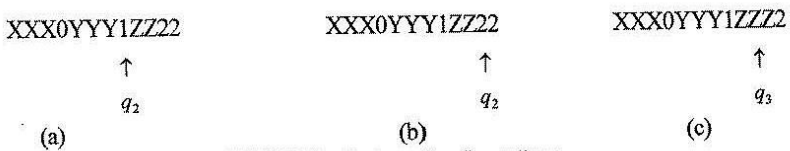
$$\delta(q_1, 0) = (q_1, 0, R)$$

$$\delta(q_1, Y) = (q_1, Y, R)$$

The configuration shown in figure 1(c) is obtained. In state q_1, on encountering 1 change the state to q_2, replace 1 by Y and move the pointer towards right. The transition for this can be of the form

$$\delta(q_1,1)=(q_2,Y,R)$$

and the configuration shown in figure 2(a) is obtained



(a) (b) (c)

FIGURE 2 : Various Configurations

In state q_2, we have to search for the leftmost 2. It is clear from figure 2(a) that, when we are searching for the symbol 2, we may encounter the symbols 1 or Z. So, replace 1 by 1, Z by Z and move the pointer towards right and remain in state q_2 only and the configuration shown in figure 2(b) is obtained. The transitions for this can be of the form

$$\delta(q_2,1)=(q_2,1,R)$$
$$\delta(q_2,Z)=(q_2,Z,R)$$

In state q_2, on encountering 2, change the state to q_3, replace 2 by Z and move the pointer towards left. The transition for this can be of the form

$$\delta(q_2,2)=(q_3,Z,L)$$

and the configuration shown in figure 2(c) is obtained. Once the TM is in state q_3, it means that equal number of 0's, 1's and 2's are replaced by equal number of X's, Y's and Z's respectively. At this point, next we have to search for the rightmost X to get leftmost 0. During this process, it is clear from figure 2(c) that the symbols such as Z's, 1,s, Y's, 0's and X are scanned respectively one after the other. So, replace Z by Z, 1 by 1, Y by Y, 0 by 0, move the pointer towards left and stay in state q_3 only. The transitions for this can be of the form

$$\delta(q_3,Z)=(q_3,Z,L)$$
$$\delta(q_3,1)=(q_3,1,L)$$
$$\delta(q_3,Y)=(q_3,Y,L)$$
$$\delta(q_3,0)=(q_3,0,L)$$

Only on encountering X, replace X by X, change the state to q_0 and move the pointer towards right to get leftmost 0. The transition for this can be of the form

$$\delta(q_3,X)=(q_0,X,R)$$

All the steps shown above are repeated till the following configuration is obtained.

XXXXYYYYZZZZ

↑

q_0

In state q_0, if the input symbol is Y, it means that there are no 0's . If there are no 0's we should see that there are no 1's also. For this to happen change the state to q_4, replace Y by Y and move the pointer towards right. The transition for this can be of the form

$$\delta(q_0, Y) = (q_4, Y, R)$$

In state q_4 search for only Y's, replace Y by Y, remain in state q_4 only and move the pointer towards right. The transition for this can be of the form

$$\delta(q_4, Y) = (q_4, Y, R)$$

In state q_4, if we encounter Z, it means that there are no 1's and so we should see that there are no 2's and only Z's should be present. So, on scanning the first Z, change the state to q_5, replace Z by Z and move the pointer towards right. The transition for this can be of the form

$$\delta(q_4, Z) = (q_5, Z, R)$$

But, in state q_5 only Z's should be there and no more 2's. So, as long as the scanned symbol is Z, remain in state q_5, replace Z by Z and move the pointer towards right. But, once blank symbol B is encountered change the state to q_6, replace B by B and move the pointer towards right and say that the input string is accepted by the machine. The transitions for this can be of the form

$$\delta(q_5, Z) = (q_5, Z, R)$$

$$\delta(q_5, B) = (q_6, B, R)$$

where q_6 is the final state.

So, the TM to recognize the language $L = \{ 0^n 1^n 2^n \mid n \geq 1 \}$ is given by

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

where

$Q = \{q_0, q_1, q_2, q_3, q_4, q_5, q_6\}$; $\Sigma = \{ 0, 1, 2 \}$

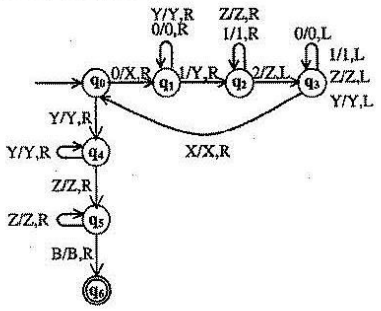
$\Gamma = \{ 0, 1, 2, X, Y, Z, B \}$; q_0 is the initial state

B is blank character ; F = { q_6 } is the final state

δ is shown below using the transition table.

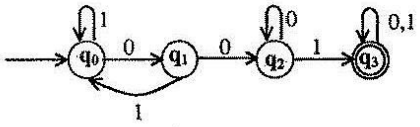
	Γ						
States	0	1	2	Z	Y	X	B
q_0	q_1, X, R				q_4, Y, R		
q_1	q_1, 0, R	q_2, Y,R			q_1, Y,R		
q_2		q_2, 1,R	q_3, Z,L	q_2, Z,R			
q_3	q_3, 0,L	q_3, 1,L		q_1, Z,L	q_3, Y,L	q_0, X,R	
q_4				q_5, Z,R	q_4, Y,R		
q_5				q_5, Z,R			(q_6, B, R)
q_6							

The transition diagram for this can be of the form



Example 3 : Obtain a TM to accept the language $L = \{w \mid w \in (0+1)^*\}$ containing the substring 001.

Solution : The DFA which accepts the language consisting of strings of 0's and 1's having a sub string 001 is shown below :



The transition table for the DFA is shown below :

	0	1
q_0	q_1	q_0
q_1	q_2	q_0
q_2	q_2	q_3
q_3	q_3	q_3

We have seen that any language which is accepted by a DFA is regular. As the DFA processes the input string from left to right in only one direction, TM also processes the input string in only one direction (unlike the previous examples, where the read - write header was moving in both the directions). For each scanned input symbol (either 0 or 1), in whichever state the DFA was in, TM also enters into the same states on same input symbols, replacing 0 by 0 and 1 by 1 and the read - write head moves towards right. So, the transition table for DFA and TM remains same (the format may be different. It is evident in both the transition tables). So, the transition table for TM to recognize the language consisting of 0's and 1's with a substring 001 is shown below :

	0	1	B
q_0	q_1,0, R	q_0, 1, R	-
q_1	q_2, 0, R	q_0, 1, R	-
q_2	q_2, 0, R	q_3, 1, R	-
q_3	q_3, 0, R	q_3, 1, R	q_4, B, R
q_4			

The TM is given by

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

where

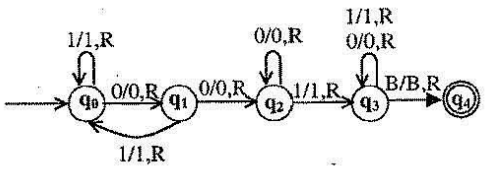
$Q = \{ q_0, q_1, q_2, q_3, q_4 \}$; $\Sigma = \{0, 1\}$

$\Gamma = \{0, 1\}$; δ – is defined already

q_0 is the initial state ; B blank character

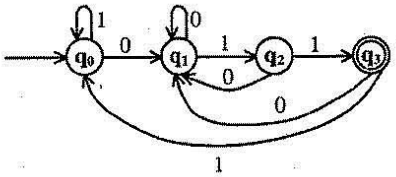
F = { q_4 } is the final state

The transition diagram for this is shown below.



Example 4 : Obtain a Turing machine to accept the language containing strings of 0's and 1's ending with 011.

Solution : The DFA which accepts the language consisting of strings of 0's and 1's ending with the string 001 is shown below :



The transition table for the DFA is shown below :

δ	0	1
q_0	q_1	q_0
q_1	q_1	q_2
q_2	q_1	q_3
q_3	q_1	q_0

We have seen that any language which is accepted by a DFA is regular. As the DFA processes the input string from left to right in only one direction, TM also processes the input string in only one direction. For each scanned input symbol (either 0 or 1), in whichever state the DFA was in, TM also enters into the same states on same input symbols, replacing 0 by 0 and 1 by 1 and the read - write head moves towards right. So, the transition table for DFA and TM remains same (the format may be different. It is evident in both the transition tables). So, the transition table for TM to recognize the language consisting of 0's and 1's ending with a substring 001 is shown below :

δ	0	1	B
q_0	q_1,0, R	q_0, 1, R	-
q_1	q_1, 0, R	q_2, 1, R	-
q_2	q_1, 0, R	q_3, 1, R	-
q_3	q_1, 0, R	q_0, 1, R	q_4, B, R
q_4	-	-	-

The TM is given by $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$
where

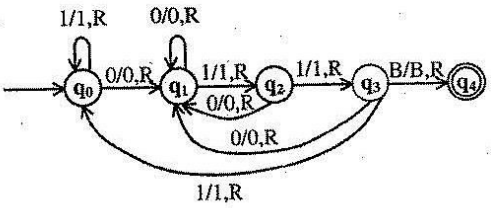
$Q = \{ q_0, q_1, q_2, q_3 \}$; $\Sigma = \{0, 1\}$; $\Gamma = \{0, 1\}$

δ – is defined already

q_0 is the initial state ; B does not appear

$F = \{ q_4 \}$ is the final state

The transition diagram for this is shown below :

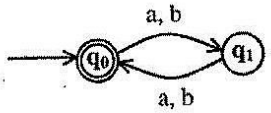


Example 5 : Obtain a Turing machine to accept the language

$L = \{ w | w \text{ is even and } \Sigma = \{ a, b \} \}$

Solution :

The DFA to accept the language consisting of even number of characters is shown below.



The transition table for the DFA is shown below :

	a	b
q_0	q_1	q_1
q_1	q_0	q_0

We have seen that any language which is accepted by a DFA is regular. As the DFA processes the input string from left to right in only one direction, TM also processes the input string in only one direction. For each scanned input symbol (either a or b), in whichever state the DFA was in, TM also enters into the same states on same input symbols, replacing a by a and b by b and the read - write head moves towards right. So, the transition table for DFA and TM remains same (the format may be different). So, the transition table for TM to recognize the language consisting of a's and b's having even number of symbols is shown below :

δ	a	b	B
q_0	q_1, a, R	q_1, b, R	q_2, B, R
q_1	q_0, a, R	q_0, b, R	-
q_2	-	-	-

The TM is given by

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

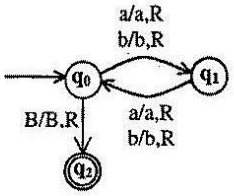
where

$Q = \{ q_0, q_1 \}$; $\Sigma = \{a, b\}$; $\Gamma = \{a, b\}$

δ – is defined already ; q_0 is the initial state

B does not appear ; $F = \{ q_2 \}$ is the final state

The transition diagram of TM is given by



Example 6 : Obtain a Turing machine to accept a palindrome consisting of a's and b's of any length.

Solution : Let us assume that the first symbol on the tape is blank character B and is followed by the string which in turn ends with blank character B. Now, we have to design a Turing machine which accepts the string, provided the string is a palindrome. For the string to be a palindrome, the first and the last character should be same. The second character and last but one character in the string should be same and so on. The procedure to accept only string of palindromes is shown below. Let q0 be the start state of Turing machine.

Step 1 : Move the read - write head to point to the first character of the string. The transition for this can be of the form $\delta(q_0, B) = (q_1, B, R)$

Step 2 : In state q_1, if the first character is the symbol a, replace it by B and change the state to q_2 and move the pointer towards right. The transition for this can be of the form

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

Now , we move the read - write head to point to the last symbol of the string and the last symbol should be a . The symbols scanned during this process are a's , b's and B. Replace a by a, b by b and move the pointer towards right. The transitions defined for this can be of the form

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

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

But, once the symbol B is encountered, change the state to q_3, replace B by B and move the pointer towards left. The transition defined for this can be of the form

$$\delta(q_2, B) = (q_3, B, L)$$

In state q_3, the read - write head points to the last character of the string. If the last character is a, then change the state to q_4, replace a by B and move the pointer towards left. The transitions defined for this can be of the form

$$\delta(q_3, a) = (q_4, B, L)$$

At this point, we know that the first character is a and last character is also a. Now, reset the read - write head to point to the first non blank character as shown in step5.

In state q_3, if the last character is B (blank character), it means that the given string is an odd palindrome. So, replace B by B change the state to q_7 and move the pointer towards right. The transition for this can be of the form

$$\delta(q_3, B) = (q_7, B, R)$$

Step 3 : If the first character is the symbol b, replace it by B and change the state from q_1 to q_5 and move the pointer towards right. The transition for this can be of the form

$$\delta(q_1, b) = (q_5, B, R)$$

Now, we move the read - write head to point to the last symbol of the string and the last symbol should be b. The symbols scanned during this process are a's, b's and B. Replace a by a, b by b and move the pointer towards right. The transitions defined for this can of the form

$$\delta(q_5, a) = (q_5, a, R)$$
$$\delta(q_5, b) = (q_5, b, R)$$

But, once the symbol B is encountered, change the state to q_6, replace B by B and move the pointer towards left. The transition defined for this can be of the form

$$\delta(q_5, B) = (q_6, B, L)$$

In state q_6, the read - write head points to the last character of the string. If the last character is b, then change the state to q_6, replace b by B and move the pointer towards left. The transitions defined for this can be of the form

$$\delta(q_6, b) = (q_4, B, L)$$

At this point, we know that the first character is b and last character is also b. Now, reset the read - write head to point to the first non blank character as shown in step 5.

In state q_6, If the last character is B (blank character), it means that the given string is an odd palindrome. So, replace B by B, change the state to q_7 and move the pointer towards right. The transition for this can be of the form

$$\delta(q_6, B) = (q_7, B, R)$$

Step 4 : In state q_1, if the first symbol is blank character (B), the given string is even palindrome and so change the state to q_7, replace B by B and move the read - write head towards right. The transition for this can be of the form

$$\delta(q_1, B) = (q_7, B, R)$$

Step 5 : Reset the read - write head to point to the first non blank character. This can be done as shown below.

If the first symbol of the string is a, step 2 is performed and if the first symbol of the string is b, step 3 is performed. After completion of step 2 or step 3, it is clear that the first symbol and the last symbol match and the machine is currently in state q_4. Now, we have to reset the read - write head to point to the first nonblank character in the string by repeatedly moving the head towards left and remain in state q_4. During this process, the symbols encountered may be a or b or B (blank character). Replace a by a, b by b and move the pointer towards left. The transitions defined for this can be of the form $\delta(q_4, a) = (q_4, a, L)$

$$\delta(q_4, b) = (q_4, b, L)$$

But, if the symbol B is encountered , change the state to q_1, replace B by B and move the pointer towards right. the transition defined for this can be of the form

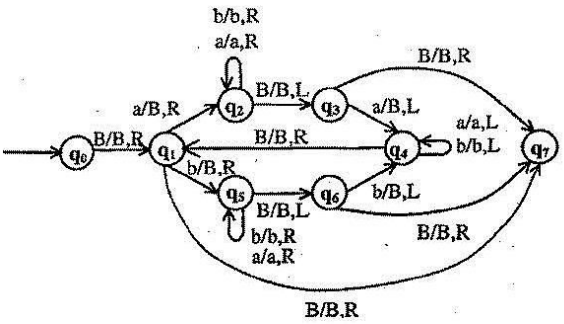
$$\delta(q_4, B) = (q_1, B, R)$$

After resetting the read - write head to the first non - blank character, repeat through step 1.

So, the TM to accept strings of palindromes over { a, b } is given by $M = (Q, \Sigma, \delta, q_0, B, F)$ where $Q = \{q_0, q_1, q_2, q_3, q_4, q_5, q_6, q_7\}$; $\Sigma = \{a, b\}$; $\Gamma = \{a, b, B\}$; q_0 is the initial state B is the blank character ; $F = \{q_7\}$; δ is shown below using the transition table

	Γ		
δ	a	b	B
q_0	-	-	q_1, B, R
q_1	q_2, B, R	q_5, B, R	q_7, B, R
q_2	q_2, a, R	q_2, b, R	q_3, B, L
q_3	q_4, B, L	-	q_7, B, R
q_4	q_4, a, L	q_4, b, L	q_1, B, R
q_5	q_5, a, R	q_5, b, R	q_6, B, L
q_6	-	q_4, B, L	q_7, B, R
q_7	-	-	-

The transition diagram to accept palindromes over { a, b } is given by

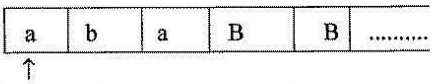


The reader can trace the moves made by the machine for the strings abba, aba and aaba and is left as an exercise.

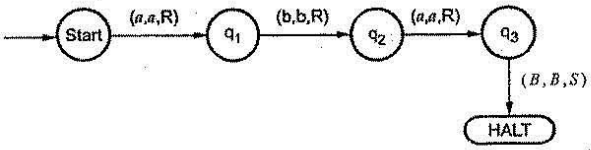
Example 7 : Construct a Turing machine which accepts the language of aba over $\Sigma=\{a,b\}$.

Solution : This TM is only for L = { aba }

We will assume that on the input tape the string 'aba' is placed like this

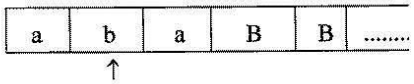


The tape head will read out the sequence upto the B character if 'aba' is readout the TM will halt after reading B.

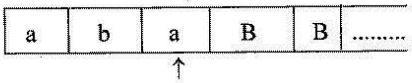


The triplet along the edge written is (input read, output to be printed, direction)

Let us take the transition between start state and q_1 is (a, a, R) that is the current symbol read from the tape is a then as a output a only has to be printed on the tape and then move the tape head to the right. The tape will look like this



Again the transition between q_1 and q_2 is (b, b, R). That means read b, print b and move right. Note that as tape head is moving ahead the states are getting changed.



The TM will accept the language when it reaches to halt state. Halt state is always a accept state for any TM. Hence the transition between q_3 and halt is (B, B, S). This means read B, print B and stay there or there is no move left or right. Eventhough we write (B, B, L) or (B, B, R) it is equally correct. Because after all the complete input is already recognized and now we simply want to enter into a accept state or final state. Note that for invalid inputs such as abb or ab or bab ..... there is either no path reaching to final state and for such inputs the TM gets stucked in between. This indicates that these all invalid inputs can not be recognized by our TM.

The same TM can be represented by another method of transition table

	a	b	B
Start	(q_1, a, R)	-	-
q_1	-	(q_2, b, R)	-
q_2	(q_3, a, R)	-	-
q_3	-	-	(HALT, B, S)
HALT	-	-	-

In the given transition table, we write the triplet in each row as :

(Next state, output to be printed, direction)

Thus TM can be represented by any of these methods.

Example 8 : Design a TM that recognizes the set $L = \{0^{2^n}1^n \mid n \geq 0\}$.

Solution : Here the TM checks for each one whether two 0's are present in the left side. If it match then only it halts and accept the string.

The transition graph of the TM is ,

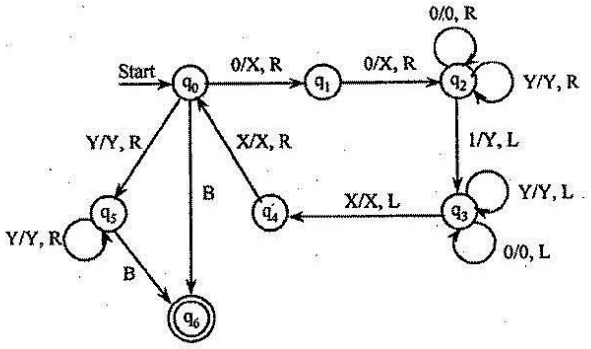


FIGURE : Turing Machine for the given language $L = \{0^{2^n}1^n \mid n \geq 0\}$

Example 11 : What does the Turing Machine described by the 5 - tuples,

$(q_0,0,q_0,1,R),(q_0,1,q_1,0,r),(q_0,B,q_2,B,R)$,

$(q_1,0,q_1,0,R)$, $(q_1,1,q_0,1,R)$ *and* (q_1,B,q_2,B,R) .Do when given a bit string as input ?

Solution : The transition diagram of the TM is ,

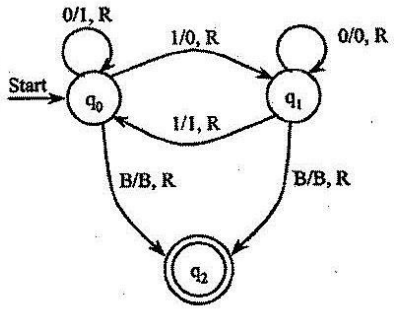


FIGURE : Transition Diagram for the given TM

The TM here reads an input and starts inverting 0's to 1's and 1's to 0's till the first 1. After it has inverted the first 1, it read the input symbol and keeps it as it is till the next 1. After encountering the 1 it starts repeating the cycle by inverting the symbol till next 1. It halts when it encounters a blank symbol.

7.4 COMPUTABLE FUNCTIONS

A Turing machine is a language acceptor which checks whether a string x is accepted by a language L. In addition to that it may be viewed as computer which performs computations of functions from integers to integers. In traditional approach an integer is represented in unary, an integer $i \geq 0$ is represented by the string 0^i .

Example 1 : 2 is represented as 0^2 . If a function has k arguments, $i_1, i_2, \ldots\ldots i_k$, then these integers are initially placed on the tape separated by 1's, as $0^{i_1}10^{i_2}1 \ldots\ldots 10^{i_k}$.

If the TM halts (whether in or not in an accepting state) with a tape consisting of 0's for some m, then we say that $f(i_1, i_2, \ldots\ldots i_k) = m$, where f is the function of k arguments computed by this Turing machine.

$\delta(q_4,1) = (q_4, B, L)$

$\delta(q_4,0) = (q_4, 0, L)$

$\delta(q_4,0) = (q_6, 0, R)$

If in state q_2 a B is encountered before a 0, we have situation (i) described above. Enter state q_4 and move left, changing all 1's to B 's until encountering a 'B'. This B is changed back to a 0, state q_6 is entered, and M halts.

6. $\delta(q_0,1) = (q_5, B, R)$

$\delta(q_5,0) = (q_5, B, R)$

$\delta(q_5,1) = (q_5, B, R)$

$\delta(q_5,B) = (q_6, B, R)$

If in state q_0 a 1 is encountered instead of a 0, the first block of 0's has been exhausted, as in situation (ii) above. M enters state q_5 to erase the rest of the tape, then enters q_6 and halts.

Example 4 : Design a TM which computes the addition of two positive integers.

Solution : Let TM $M = (Q, \{0, 1, \#\}, \delta, s)$ computes the addition of two positive integers m and n. It means, the computed function f (m, n) defined as follows :

$$f(m,n) = \begin{cases} m+n (If\ m, n \geq 1) \\ 0 \qquad (m = n = 0) \end{cases}$$

1 on the tape separates both the numbers m and n. Following values are possible for m and n.

1. $m = n = 0$ (# 1 # ..... is the input),
2. m = 0 and $n \neq 0$ (#10^n# ..... is the input),
3. $m \neq 0$ and n = 0 (#0^m1# ... is the input), and
4. $m \neq 0$ and $n \neq 0$ (#0^m10^n# ..... is the input)

Several techniques are possible for designing of M, some are as follows :

(a) M appends (writes) m after n and erases the m from the left end.

(b) M writes 0 in place of 1 and erases one zero from the right or left end . This is possible in case of $n \neq 0$ or $m \neq 0$ only. If m =0 or n = 0 then 1 is replaced by #.

We use techniques (b) given above. M is shown in below figure.

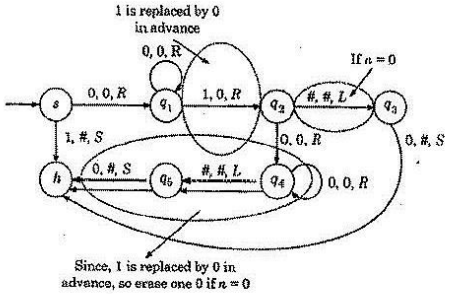


FIGURE : TM for addition of two positive integers

7.5 RECURSIVELY ENUMERABLE LANGUAGES

A language L over the alphabet Σ is called recursively enumerable if there is a TM M that accept every word in L and either rejects (crashes) or loops for every word in language L' the complement of L.

Accept (M) = L

Reject (M) + Loop (M) = L'

When TM M is still running on some input (of recursively enumerable languages) we can never tell whether M will eventually accept if we let it run for long time or M will run forever (in loop).

Example : Consider a language (a + b) * bb (a + b) *.

TM for this language is ,

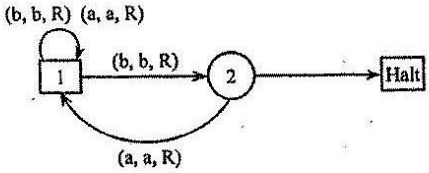


FIGURE : Turing Machine for (a + b) * bb (a + b) *

Here the inputs are of three types.

1. All words with bb = accepts (M) as soon as TM sees two consecutive b's it halts.
2. All strings without bb but ending in b = rejects (M). When TM sees a single b, it enters state2. If the string is ending with b, TM will halt at state 2 which is not accepting state. Hence it is rejected.
3. All strings without bb ending in 'a' or blank 'B' = loop (M) here when the TM sees last a it enters state 1. In this state on blank symbol it loops forever.

Recursive Language

A language L over the alphabet Σ is called recursive if there is a TM M that accepts every word in L and rejects every word in L' i. e.,

accept (M) = L
reject (M) = L'
loop (M) = ϕ.

Example :Consider a language b (a + b) * . It is represented by TM as :

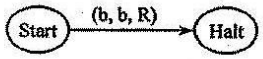


FIGURE : Turing Machine for b (a + b) *

This TM accepts all words beginning with 'b' because it enters halt state and it rejects all words beginning with a because it remains in start state which is not accepting state.

A language accepted by a TM is said to be recursively enumerable languages. The subclass of recursively enumberable sets (r. e) are those languages of this class are said to be recursive sets or recursive language.

7.6 CHURCH'S HYPOTHESIS

According to church's hypothesis, all the functions which can be defined by human beings can be computed by Turing machine. The Turing machine is believed to be ultimate computing machine.

The church's original statement was slightly different because he gave his thesis before machines were actually developed. He said that any machine that can do certain list of operations will be able to perform all algorithms. TM can perform what church asked, so they are possibly the machines which church described.

Church tied both recursive functions and computable functions together. Every partial recursive function is computable on TM. Computer models such as RAM also give rise to partial recursive functions. So they can be simulated on TM which confirms the validity of churches hypothesis.

Important of church's hypothesis is as follows .

1. First we will prove certain problems which cannot be solved using TM.

2. If churches thesis is true this implies that problems cannot be solved by any computer or any programming languages we might every develop.

3. Thus in studying the capabilities and limitations of Turing machines we are indeed studying the fundamental capabilities and limitations of any computational device we might even construct.

It provides a general principle for algorithmic computation and, while not provable, gives strong evidence that no more powerful models can be found.

7.7 COUNTER MACHINE

Counter machine has the same structure as the multistack machine, but in place of each stack is a counter. Counters hold any non negative integer, but we can only distinguish between zero and non zero counters.

Counter machines are off - line Turing machines whose storage tapes are semi - infinite, and whose tape alphabets contain only two symbols, Z and B (blank). Furthermore the symbol Z, which serves as a bottom of stack marker, appears initially on the cell scanned by the tape head and may never appear on any other cell. An integer i can be stored by moving the tape head i cells to the right of Z. A stored number can be incremented or decremented by moving the tape head right or left. We can test whether a number is zero by checking whether Z is scanned by the head, but we cannot directly test whether two numbers are equal.

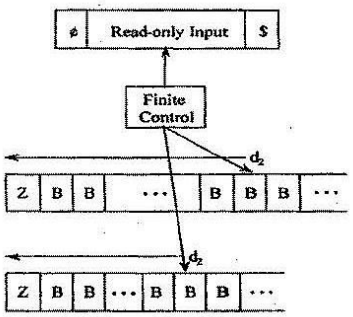


FIGURE : Counter Machine

¢ and $ are customarily used for end markers on the input. Here Z is the non blank symbol on each tape. An instantaneous description of a counter machine can be described by the state, the input tape contents, the position of the input head, and the distance of the storage heads from the symbol Z (shown here as d_1 and d_2). We call these distances the counts on the tapes. The counter machine can only store a count an each tape and tell if that count is zero.

Power of Counter Machines

- Every language accepted by a counter Machine is recursively enumerable.
- Every language accepted by a one - counter machine is a CFL so a one - counter machine is a special case of one - stack machine i. e., a PDA

7.8 TYPES OF TURING MACHINES

Various types of Turing Machines are :

i. With multiple tapes.
ii. With one tape but multiple heads.
iii. With two dimensional tapes.
iv. Non deterministic Turing machines.

It is observed that computationally all these Turing Machines are equally powerful. That means one type can compute the same that other can. However, the efficiency of computation may vary.

1. Turing machine with Two - Way Infinite Tape :

This is a TM that have one finite control and one tape which extends infinitely in both directions.

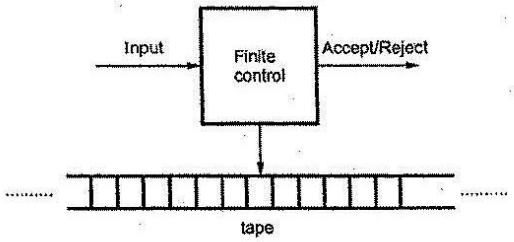


FIGURE : TM with infinite Tape

It turns out that this type of Turing machines are as powerful as one tape Turing machines whose tape has a left end.

2. Multiple Turing Machines :

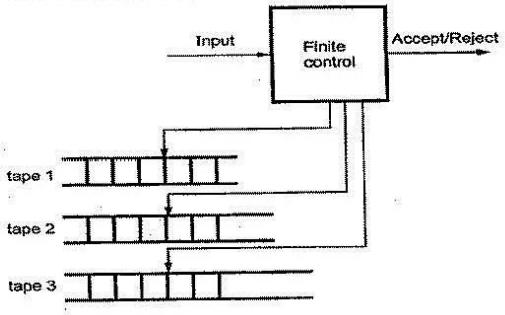


FIGURE : Multiple Turing Machines

A multiple 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 the finite control and the symbol scanned by each of the tape heads, the machine can

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

Initially, the input appears on the first tape and the other tapes are blank.

3. Nondeterministic Turing Machines :

A nondeterministic Turing machine is a device with a finite control and a single, one way infinite tape. For a given state and tape symbol scanned by the tape head, the machine has a finite number of choices for the next move. Each choice consists of a new state, a tape symbol to print, and a direction of head motion. Note that the non deterministic TM is not permitted to make a move in which the next state is selected from one choice, and the symbol printed and / or direction of head motion are selected from other choices. The non deterministic TM accepts its input if any sequence of choices of moves leads to an accepting state.

As with the finite automaton, the addition of nondeterminism to the Turing machine does not allow the device to accept new languages.

4. Multidimensional Turing Machines :

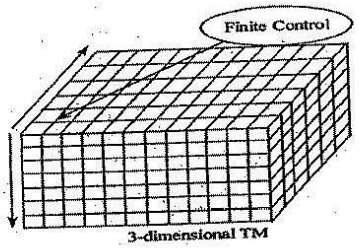


FIGURE : Multidimensional Turing Machine

The multidimensional Turing machine 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 2 k directions, either positively or negatively, along one of the k axes. Initially, the input is along one axis, and the head is at the left end of the input.At any time, only a finite number of rows in any dimension contains nonblank symbols, and these rows each have only a finite number of nonblank symbols

5. Multihead Turing Machines :

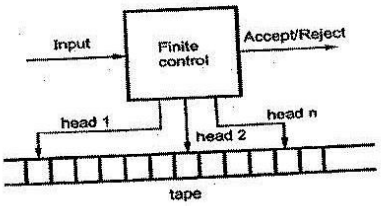


FIGURE : Multihead Turing Machine

A k - head Turing machine has some fixed number, k, of heads. The heads are numbered 1 through k, and a 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, right or remain stationary.

6. Off - Line Turing Machines :

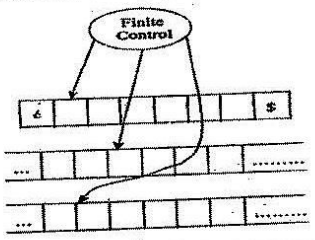


FIGURE : Off - line Turing Machine