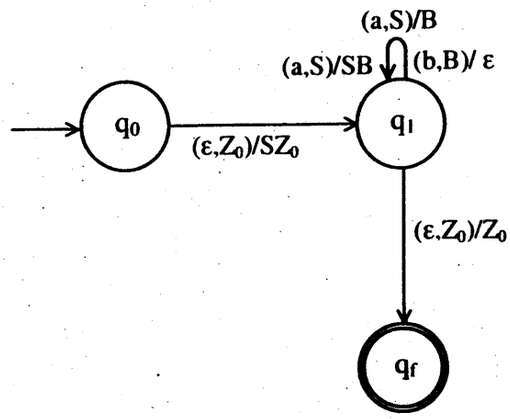
where δ is shown using the table. The corresponding transition diagram or the state diagram is also shown:

$\delta(q_0, \varepsilon, Z_0) = (q_1, SZ_0)$
$\delta(q_1, a, S) = (q_1, SB)$
$\delta(q_1, a, S) = (q_1, B)$
$\delta(q_1, b, B) = (q_1, \varepsilon)$
$\delta(q_1, \varepsilon, Z_0) = (q_f, Z_0)$



Example 7.21: Obtain a the PDA to accept the language $L = \{ww^R : |w| \geq 1 \text{ for } w \in (a + b)^*\}$

The equivalent CFG to generate the language is:

$S \rightarrow aSa \mid aa$
$S \rightarrow bSb \mid bb$

The corresponding grammar which is in GNF is

$S \rightarrow aSA \mid aA$
$S \rightarrow bSB \mid bB$
$A \rightarrow a$
$B \rightarrow b$

The equivalent transitions for the above productions are:

$\delta(q_1, a, S) = \{(q_1, SA), (q_1, A)\}$
$\delta(q_1, b, S) = \{(q_1, SB), (q_1, B)\}$
$\delta(q_1, a, A) = (q_1, \varepsilon)$
$\delta(q_1, b, B) = (q_1, \varepsilon)$

In section 7.7, we have seen that the transition

$\delta(q_0, \varepsilon, Z_0) = (q_1, SZ_0)$

is used to push S on to the stack initially and the last transition

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

is used to move to the final state. So, the PDA is given by

$$M = (\{q_0, q_1, q_f\}, \{a,b\}, \{S,A,B,Z_0\}, \delta, q_0, Z_0, q_f)$$

where δ is shown below:

$$\begin{aligned}
\delta(q_0, \varepsilon, Z_0) &= (q_1, SZ_0) \\
\delta(q_1, a, S) &= \{(q_1, SA), (q_1, A)\} \\
\delta(q_1, b, S) &= \{(q_1, SB), (q_1, B)\} \\
\delta(q_1, a, A) &= (q_1, \varepsilon) \\
\delta(q_1, b, B) &= (q_1, \varepsilon) \\
\delta(q_1, \varepsilon, Z_0) &= (q_f, Z_0)
\end{aligned}$$

7.9 PDA to CFG

As we have converted CFG to PDA, we can convert a given PDA to CFG. The general procedure for this conversion is shown below:

1. The input symbols of PDA will be the terminals of CFG
2. If the PDA moves from state to q_i to state q_j on consuming the input $a \in \Sigma$ when Z is the top of the stack, then the non-terminals of CFG are the triplets of the form (q_iZq_j)
3. If q_0 is the start state and q_f is the final state then (q_0Zq_f) is the start symbol of CFG.
4. The productions of CFG can be obtained from the transitions of PDA as shown below:
   a. For each transition of the form

      $$\delta(q_i, a, Z) = (q_j, AB)$$

      introduce the productions of the form

      $$(q_iZq_k) \rightarrow a\ (q_jAq_l)(q_lBq_k)$$

      where q_k and q_l will take all possible values from Q.

   b. For each transition of the form

      $$\delta(q_i, a, Z) = (q_j, \varepsilon)$$

      introduce the production

      $$(q_iZq_j) \rightarrow a$$

Note: Using this procedure, we may introduce lot of useless symbols, which in any way can be eliminated.

Example 7.22: Obtain a CFG for the PDA shown below:

$$\delta(q_0, a, Z) = (q_0, AZ)$$
$$\delta(q_0, a, A) = (q_0, A)$$
$$\delta(q_0, b, A) = (q_1, \varepsilon)$$
$$\delta(q_1, \varepsilon, Z) = (q_2, \varepsilon)$$

Note: To obtain a CFG from the PDA, all the transitions should be of the form

$$\delta(q_i, a, Z) = (q_j, AB)$$

or

$$\delta(q_i, a, Z) = (q_j, \varepsilon)$$

In the given transitions except the second transition, all transitions are in the required form. So, let us take the second transition

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

and convert it into the required form. This can be achieved if we have understood what the transition indicates. It is clear from the transition that when input symbol *a* is encountered and top of the stack is A, the PDA remains in state q_0 and contents of the stack are not altered. This can be interpreted as delete A from the stack and insert A onto the stack.
So, once A is deleted from the stack we enter into new state q_3. But, in state q_3 without consuming any input we add A on to the stack. The corresponding transitions are:

$$\delta(q_0, a, A) = (q_3, \varepsilon)$$
$$\delta(q_3, \varepsilon, Z) = (q_0, AZ)$$

So, the given PDA can be written using the following transitions

$$\delta(q_0, a, Z) = (q_0, AZ)$$
$$\delta(q_0, a, A) = (q_3, \varepsilon)$$
$$\delta(q_3, \varepsilon, Z) = (q_0, AZ)$$
$$\delta(q_0, b, A) = (q_1, \varepsilon)$$
$$\delta(q_1, \varepsilon, Z) = (q_2, \varepsilon)$$

Now, the transitions

$$\delta(q_0, a, A) = (q_3, \varepsilon)$$
$$\delta(q_0, b, A) = (q_1, \varepsilon)$$
$$\delta(q_1, \varepsilon, Z) = (q_2, \varepsilon)$$

can be converted into productions as shown below:

For δ of the form $\delta(q_i, a, Z) = (q_j, \varepsilon)$	Resulting Productions $(q_iZq_j) \rightarrow a$
$\delta(q_0, a, A) = (q_3, \varepsilon)$	$(q_0Aq_3) \rightarrow a$
$\delta(q_0, b, A) = (q_1, \varepsilon)$	$(q_0Aq_1) \rightarrow b$
$\delta(q_1, \varepsilon, Z) = (q_2, \varepsilon)$	$(q_1Zq_2) \rightarrow \varepsilon$

Now, the transitions

$$\delta(q_0, a, Z) = (q_0, AZ)$$
$$\delta(q_3, \varepsilon, Z) = (q_0, AZ)$$

can be converted into productions using rule 4.a as shown below:

For δ of the form $\delta(q_i,a,Z) = (q_j, AB)$	Resulting Productions $(q_iZq_k) \rightarrow a\ (q_iAq_l)(q_lBq_k)$
$\delta(q_0, a, Z) = (q_0, AZ)$	$(q_0Zq_0) \rightarrow a\ (q_0Aq_0)(q_0Zq_0) \mid a\ (q_0Aq_1)(q_1Zq_0) \mid a\ (q_0Aq_2)(q_2Zq_0) \mid a\ (q_0Aq_3)(q_3Zq_0)$ $(q_0Zq_1) \rightarrow a\ (q_0Aq_0)(q_0Zq_1) \mid a\ (q_0Aq_1)(q_1Zq_1) \mid a\ (q_0Aq_2)(q_2Zq_1) \mid a\ (q_0Aq_3)(q_3Zq_1)$ $(q_0Zq_2) \rightarrow a\ (q_0Aq_0)(q_0Zq_2) \mid a\ (q_0Aq_1)(q_1Zq_2) \mid a\ (q_0Aq_2)(q_2Zq_2) \mid a\ (q_0Aq_3)(q_3Zq_2)$ $(q_0Zq_3) \rightarrow a\ (q_0Aq_0)(q_0Zq_3) \mid a\ (q_0Aq_1)(q_1Zq_3) \mid a\ (q_0Aq_2)(q_2Zq_3) \mid a\ (q_0Aq_3)(q_3Zq_3)$
$\delta(q_3, \varepsilon, Z) = (q_0, AZ)$	$(q_3Zq_0) \rightarrow (q_0Aq_0)(q_0Zq_0) \mid (q_0Aq_1)(q_1Zq_0) \mid (q_0Aq_2)(q_2Zq_0) \mid (q_0Aq_3)(q_3Zq_0)$ $(q_3Zq_1) \rightarrow (q_0Aq_0)(q_0Zq_1) \mid (q_0Aq_1)(q_1Zq_1) \mid (q_0Aq_2)(q_2Zq_1) \mid (q_0Aq_3)(q_3Zq_1)$ $(q_3Zq_2) \rightarrow (q_0Aq_0)(q_0Zq_2) \mid (q_0Aq_1)(q_1Zq_2) \mid (q_0Aq_2)(q_2Zq_2) \mid (q_0Aq_3)(q_3Zq_2)$ $(q_3Zq_3) \rightarrow (q_0Aq_0)(q_0Zq_3) \mid (q_0Aq_1)(q_1Zq_3) \mid (q_0Aq_2)(q_2Zq_3) \mid (q_0Aq_3)(q_3Zq_3)$

The start symbol of the grammar will be q_0Zq_2.

Example 7.23: Obtain a CFG that generates the language accepted by PDA M = ($\{q_0,q_1\}$, {a, b}, {A, Z}, δ, q_0, Z, $\{q_1\}$), with the transitions

$$\delta(q_0, a, Z) = (q_0, AZ)$$
$$\delta(q_0, b, A) = (q_0, AA)$$
$$\delta(q_0, a, A) = (q_1, \varepsilon)$$

Now, the transition

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

can be converted into production as shown below:

For δ of the form $\delta(q_i, a, Z) = (q_j, \varepsilon)$	Resulting Productions $(q_iZq_j) \rightarrow a$
$\delta(q_0, a, A) = (q_1, \varepsilon)$	$(q_0Aq_1) \rightarrow a$

Now, the transitions

$$\delta(q_0, a, Z) = (q_0, AZ)$$
$$\delta(q_0, b, A) = (q_0, AA)$$

can be converted into productions using rule 4.a as shown below:

For δ of the form $\delta(q_i,a,Z) = (q_j, AB)$	Resulting Productions $(q_iZq_k) \rightarrow a\ (q_iAq_l)(q_lBq_k)$
$\delta(q_0, a, Z) = (q_0, AZ)$	$(q_0Zq_0) \rightarrow a\ (q_0Aq_0)(q_0Zq_0) \mid a\ (q_0Aq_1)(q_1Zq_0)$ $(q_0Zq_1) \rightarrow a\ (q_0Aq_0)(q_0Zq_1) \mid a\ (q_0Aq_1)(q_1Zq_1)$
$\delta(q_0, b, A) = (q_0, AA)$	$(q_0Zq_0) \rightarrow b(q_0Aq_0)(q_0Aq_0) \mid b(q_0Aq_1)(q_1Aq_0)$ $(q_0Zq_1) \rightarrow b(q_0Aq_0)(q_0Aq_1) \mid b(q_0Aq_1)(q_1Aq_1)$

The start symbol of the grammar will be q_0Zq_1.

Exercises:

1. What are the demerits of regular languages when compared to context free languages?
2. What are the demerits of DFA (or NFA) when compared with PDA?
3. Why FAs are less powerful than the PDAs?
4. What is the difference between NFA and PDA?
5. What is a PDA? Explain with an example.
6. What does each of the following transitions represent?
   a. $\delta(p, a, Z) = (q, aZ)$
   b. $\delta(p, a, Z) = (q, \varepsilon)$
   c. $\delta(p, a, Z) = (q, r)$
   d. $\delta(p, \varepsilon, Z) = (q, r)$
   e. $\delta(p, \varepsilon, \varepsilon) = (q, Z)$
   f. $\delta(p, \varepsilon, Z) = (q, \varepsilon)$
7. How the transition / move of a PDA defined?

8. What is an instantaneous description? Explain with respect to PDA.

9. When a language is accepted by a final state and when a language is accepted by an empty stack?

10. Obtain a PDA to accept the language L(M) = $\{wCw^R \mid w \in (a + b)^*\}$ where W^R is reveres of W. Show the sequence of moves made by the PDA for the strings aabCbaa, aabCbab.

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

12. Obtain a PDA to accept the language L(M) = $\{w \mid w \in (a+b)^*$ and $n_a(w) = n_b(w)$ i.e., number of a's in string w should be equal to number of b's in w.

13. Obtain a PDA to accept a string of balanced parentheses. The parentheses to be considered are (,), [,], { and }.

14. Obtain a PDA to accept the language L = $\{w \mid w \in (a, b)^*$ and $n_a(w) > n_b(w)\}$

15. Obtain a PDA to accept the language L = $\{a^n b^{2n} \mid n \geq 1\}$

16. Obtain a PDA to accept the language L = $\{ww^R \mid w \in (a + b)^*\}$

17. When the PDA is deterministic and when it is called non-determinstic?

18. Is the PDA to accept the language L(M) = $\{wCw^R \mid w \in (a + b)^*\}$ is deterministic?

19. Is the PDA corresponding to the language L = $\{a^n b^n \mid n \geq 1\}$ by a final state is deterministic?

20. Is the PDA to accept the language L(M) = $\{w \mid w \in (a+b)^*$ and $n_a(w) = n_b(w)$ is deterministic?

21. Is the PDA to accept the language consisting of balanced parentheses is deterministic?

22. Is the PDA to accept the language L = $\{w \mid w \in (a, b)^*$ and $n_a(w) > n_b(w)\}$ is deterministic?

23. Is the PDA to accept the language L = $\{a^n b^{2n} \mid n \geq 1\}$ is deterministic?

24. Is the PDA to accept the language L = $\{ww^R \mid w \in (a + b)^*\}$ is deterministic?

25. What is the procedure to convert a CFG to PDA?

26. For the grammar

$$\begin{aligned} S &\rightarrow aABC \\ A &\rightarrow aB|a \\ B &\rightarrow bA|b \\ C &\rightarrow a \end{aligned}$$

Obtain the corresponding PDA

27. For the grammar

$$\begin{aligned} S &\rightarrow aABB|\ aAA \\ A &\rightarrow aBB|\ a \\ B &\rightarrow bBB|\ A \\ C &\rightarrow a \end{aligned}$$

Obtain the corresponding PDA

28. What is the application of GNF notation of a CFG?

29. Obtain a the PDA to accept the language $L = \{a^n b^n \mid n \geq 1\}$

30. Obtain a the PDA to accept the language $L = \{ww^R : |w| \geq 1 \text{ for } w \in (a + b)^*\}$

31. What is the general procedure used to convert from PDA to CFG?

32. Obtain a CFG for the PDA shown below:

$$\begin{aligned} \delta(q_0, a, Z) &= (q_0, AZ) \\ \delta(q_0, a, A) &= (q_0, A) \\ \delta(q_0, b, A) &= (q_1, \varepsilon) \\ \delta(q_1, \varepsilon, Z) &= (q_2, \varepsilon) \end{aligned}$$

33. Obtain a CFG that generates the language accepted by PDA $M = (\{q_0, q_1\}, \{a, b\}, \{A, Z\}, \delta, q_0, Z, \{q_1\})$, with the transitions

$$\begin{aligned} \delta(q_0, a, Z) &= (q_0, AZ) \\ \delta(q_0, b, A) &= (q_0, AA) \\ \delta(q_0, a, A) &= (q_1, \varepsilon) \end{aligned}$$

Summary

Now!! we know

- Difference between finite automaton (FA) and pushdown automaton (PDA)
- Pushdown automaton
- The transition diagram and moves of PDA
- Actions performed by PDA
- Instantaneous description
- Languages accepted by PDA by a final state
- Languages accepted by PDA by an empty stack
- Various ways of constructing PDA for the given languages
- Deterministic and non-deterministic PDA
- To obtain PDA from CFG, the method and solution to various types of problems
- Applications of GNF
- To obtain CFG from PDA
- Solutions to more than 23 problems of various nature

8

Properties of Context Free Languages

What we will know after reading this chapter?

- Pumping Lemma for CFLs
- Proof of Pumping Lemma for CFLs
- Applications of Pumping lemma for CFLs
- Solution to various problems and to show that the specified languages are not context free.
- CFLs are not closed under union
- CFLs are not closed under concatenation
- CFLs are not closed under star-closure (kleene-closure)
- CFLs are not closed under intersection
- CFLs are not closed under complementation
- Solutions to more than 20 problems of various nature

As we have discussed in previous chapters that some of the non-regular languages can be represented using context free grammars from which we can obtain the context free languages. For example, the non-regular language such as $L = \{a^n b^n \mid n \geq 1\}$ can be easily generated using context free grammars. There are so many other instances such as matching parentheses, to match the nested if statements, whether a statement is syntactically correct or not and so on most of which can be easily represented using CFGs. So, it is very important for us to learn the properties of context free languages. The different closure properties covered in this chapter are union, concatenation, star-closure, intersection, complementation etc. We have to remember that just because the regular languages are closed under union, concatenation, *-closure, intersection, complementation etc., it is not true that they are also closed under all these operations. This also covers pumping lemma, which is a very useful concept in determining whether the given language is context free or not.

Note:

1. In the derivation process, if a non terminal A occurs in some sentential form starting from the start symbol and if a string of terminals can be derived from this sentential form, then the non terminal A is useful. Otherwise, it is useless.

2. The non-terminal A can be recursive if and only if it can generate a string containing itself. For example, Consider the derivation

$$S \overset{*}{\Rightarrow} uAy$$

The non terminal A is recursive

a. If there is a production of the form

$$A \rightarrow x_1 A x_2$$

for some strings $x_1, x_2 \in (V \cup T)^*$ (Direct recursion)

or

b. If there is a production of the form

$$A \rightarrow ....X1...$$
$$X1 \rightarrow ....X2...$$
$$X2 \rightarrow ....X3...$$
$$X3 \rightarrow ....X4...$$
$$\vdots$$
$$Xn \rightarrow ....A...\text{(indirect recursion)}$$

Now let us see how to prove that certain languages are not context free similar to the proof as we did for regular languages using Pumping Lemma. The Pumping Lemma for Context Free Languages (CFL) can be stated as follows:

8.1 Pumping Lemma

Pumping Lemma for Context Free Languages: Let L be the context free language and is infinite. Let z is sufficiently long string and $z \in L$ so that $|z| \geq n$ where n is some positive integer. If the string z can be decomposed into combinations of strings

$$z = uvwxy$$

such that $|vwx| \leq n$, $|vx| \geq 1$, then $uv^iwx^iy \in L$ for $i = 0, 1, 2, ...$

Note: The following observations can be made from the Pumping Lemma.

1. n is the length of the longest string that can be generated by the parse tree where the same non terminal never occurs twice on the same path through the tree.
2. The string z is sufficiently long so that it can be decomposed into various sub strings u, v, w, x and y in that sequence.
3. The two sub strings v and x are present somewhere in z.
4. The sub string u appears before v, the sub string w is in between v and x and the sub string y appears after x.
5. The string w in between v and x cannot be too long since $|vwx| \leq n$ for some positive integer n.
6. Both the sub strings v and x cannot be empty since $|vx| \geq 1$. One of them can be empty.
7. If all the points mentioned from 1 to 5 are satisfied, and if we duplicate sub string v and x same number of times, the resultant string will definitely be in L and the string $z \in L$ is context free. Otherwise, the string $z \in L$ is not context free.

Proof: According to Pumping Lemma, it is assumed that string $z \in L$ is finite and is context free language. We know that z is string of terminals which is derived by applying series of productions. Proof of this theorem leads to the following two cases.

Case 1: To generate a sufficiently long string z(it is assumed that the string is infinite), one or more variables (non-terminals) must be recursive (Note that infinite string can be generated if the grammar has some non-terminal A such that

$$A \overset{*}{\Rightarrow} \alpha A \beta$$

for some α and β) and should be applied more than once.

Case 2: $z \in L$ implies that the after applying some/all productions some number of times, we get finally string of terminals and the derivation stops. If we can prove these two cases, we have the proof for Pumping Lemma.

Proof of case 1: To generate a sufficiently long string z(it is assumed that the string is infinite), one or more variables (non-terminals) must be recursive. Let us assume that the language is finite and the grammar has a finite number of variables (assume that all are useful variables) and each production has finite length. The only way to derive sufficiently long string using such productions is that the grammar should have one or more recursive variables. Assume that no variable is recursive.

Since no non-terminal (variable) is recursive, each variable must be defined only in terms of terminal(s) and/or other variables. Since those variables are also non-recursive, they have to be defined in terms of terminals and other variables and so on. If we keep applying the productions like this, there are no variables at all in the final derivation and finally we get string of terminals and the generated string is finite. From this we conclude that there is a limit on the length of the string that is generated from the start symbol S. This contradicts our assumption that the language

is finite. Therefore, the assumption that one or more variables are non-recursive is incorrect. It means that one or more variables are recursive and hence the proof.

Proof of case 2: $z \in L$ implies that after applying some/all productions some number of times, we get finally string of terminals and the derivation stops. Let $z \in L$ is sufficiently long string and so the derivation must have involved recursive use of some non-terminal A and the derivations must have the form

$$S \overset{*}{\Rightarrow} uAy$$

Note that any derivation should start from the start symbol S. Since A is used recursively, the derivation can take the following form:

$$S \overset{*}{\Rightarrow} uAy \overset{*}{\Rightarrow} uvAxy$$

and the final derivation should be of the form

$$S \overset{*}{\Rightarrow} uAy \overset{*}{\Rightarrow} uvAxy \overset{*}{\Rightarrow} uvwxy = z$$

It implies that the following derivations

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

and

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

are also possible. From this we can easily conclude that the derivation

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

must also be possible. Next we have to prove that the longest string *vwx* is generated without recursion since it is assumed that $|vwx| \leq n$. This can be easily proved since CFG that generates CFL does not contain ε-productions or unit productions. It shows that every derivation step either increases the length of the sentential form (using recursive variable) or introduces a terminal. The derivation

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

used earlier clearly shows that

$$|vx| \geq 1.$$

Note from the derivation

$$S \overset{*}{\Rightarrow} uAy \overset{*}{\Rightarrow} uvAxy$$

that uvAxy occurs in the derivation, and

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

and

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

are also possible, it follows that

$$uv^iwx^iy \in L$$

and hence the proof.

8.2 Applications of Pumping Lemma for CFLs

The Pumping Lemma for CFLs is used to prove that certain languages are not context free languages. Note that Pumping Lemma can not be used to prove that certain languages are context free. In this section let us show that certain languages are not context free using Pumping lemma (similar to the problems by showing that certain languages are non-regular languages).

The general strategy used to prove that a given language is not context free is shown below.

1. Assume that the language L is infinite and it is context free.
2. Select the string say z and break it into sub strings u, v, w, x and y such that z = uvwxy where $|vwx| \leq n$ and $|vx| \geq 1$.
3. Find any i such that $uv^iwx^iy \notin L$. According to pumping lemma, $uv^iwx^iy \in L$. So, the result is a contradiction to the assumption that the language is context free. Therefore, the given language L is not context free.

Example 8.1: Show that $L = \{a^nb^nc^n \mid n \geq 0\}$ is not context free.

Step1: Let L is context free and is infinite. Let $z = a^nb^nc^n \in L$.

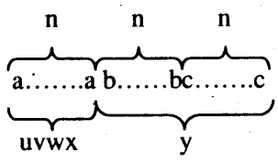
Step2: Note that $|z| > n$ and so we can split z into uvwxy such that

$$|vwx| \leq n \text{ and } |vx| \geq 1$$

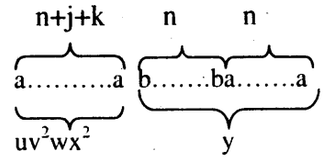
and so $uv^iwx^iy \in L$ for for i = 0, 1, 2....(This is according to Pumping Lemma). Let us take the various cases

Case 1: The string vwx is within a^n.

Let $v = a^j$, $x = a^k$ where $|vx| = j + k \geq 1$ and $|vwx| <= n$ which can be shown pictorially as



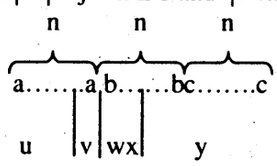
Now, according to Pumping Lemma, $uv^2wx^2y \in L$ for i = 2 and the language generated is as shown below:



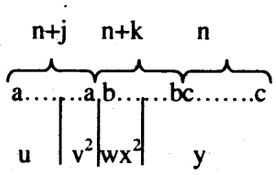
Note that $uv^2wx^2y = a^{n+j+k}b^nc^n = \notin L$ when $j + k >= 1$ (Note that the string should have some number of a's followed by equal number of b's and c's). But according to pumping lemma, $uv^2wx^2y \in L$, which is the contradiction.

Case 2: The string vwx is a^nb^n.

Let $v = a^j$, $x = b^k$ where $|vx| = j + k \geq 1$ and $|vwx| <= n$ which can be shown pictorially as



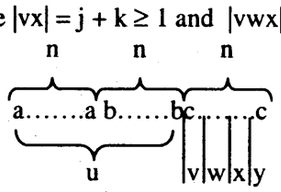
Now, according to Pumping Lemma, $uv^2wx^2y \in L$ for $i = 2$ and the language generated is as shown below:



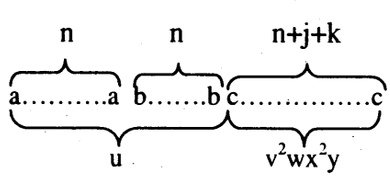
Note that $uv^2wx^2y = a^{n+j}b^{n+k}c^n = \notin L$ when $j + k >= 1$ (Note that the string should have some number of a's followed by equal number of b's and c's). But according to pumping lemma, $uv^2wx^2y \in L$, which is the contradiction.

Case 3: The string vwx is within c^n.

Let $v = c^j$, $x = c^k$ where $|vx| = j + k \geq 1$ and $|vwx| <= n$ which can be shown pictorially as



Now, according to Pumping Lemma, $uv^2wx^2y \in L$ for $i = 2$ and the language generated is as shown below:



Note that $uv^2wx^2y = a^nb^nc^{n+j+k} \notin L$ when $j + k >= 1$ (Note that the string should have some number of a's followed by equal number of b's and c's). But according to pumping lemma, $uv^2wx^2y \in L$, which is the contradiction.

But, according to pumping lemma, n number of a's should be followed by n number of b's which in turn should be followed by n number of c's. In all the three cases we get contradiction to the assumption that the language is context free.

So, the language $L = \{a^n b^n c^n \mid n \geq 0\}$ is not context free.

Example 8.2: Show that $L = \{w \mid w \in \{a,b,c\}^* \text{ where } n_a(w) = n_b(w) = n_c(w)\}$ is not context free.

The language

$$L_1 = \{a^n b^n c^n \mid n \geq 0\}$$

is obtained by the intersection of L and the regular language represented by the regular expression a*b*c* i.e.,

$$\{a^n b^n c^n \mid n \geq 0\} = \{a^*b^*c^* \cap \{w \mid w \in \{a,b,c\}^* \text{ where } n_a(w) = n_b(w) = n_c(w)\}$$

We know that intersection of context free language and regular language is also a context free. But, we have already proved that the language

$$L_1 = \{a^n b^n c^n \mid n \geq 0\}$$

is not context free. Since L1 is not context free, it implies that the given language

$$L = \{w \mid w \in \{a,b,c\}^* \text{ where } n_a(w) = n_b(w) = n_c(w)\} \text{ is not context free.}$$

is also not context free.

Example 8.3: Show that $L = \{ww \mid w \in \{a,b\}^*\}$ is not context free.

Step1: Assume that L is context free and is infinite. Let $z = a^n b^n a^n b^n \in L$.

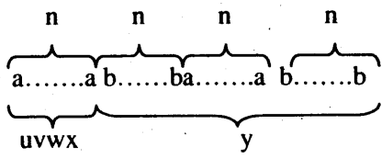
Step 2: Since $|z| > n$, according to Pumping Lemma we can split z into u, v, w, x and y such that

$$|vwx| <= n \text{ and } |vx| >= 1$$

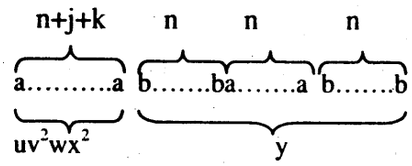
and $uv^iwx^iy \in L$ for for i = 0, 1, 2....Let us take the various cases of splitting the string into u, v, w, x and y.

Case 1: The string vwx is within first a^n.

Let $v = a^j$, $x = a^k$ where $|vx| = j + k \geq 1$ and $|vwx| <= n$ which can be shown pictorially as



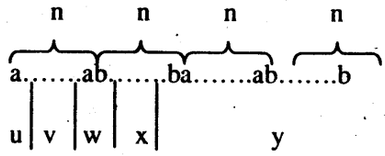
Now, according to Pumping Lemma, $uv^2wx^2y \in L$ for i = 2.



Note that $a^{n+j+k}b^n a^n b^n = uv^2wx^2y \notin L$ when j+k >=1 (Note that the string ww of the form is not generated). But by pumping lemma, $uv^2wx^2y \in L$, which is the contradiction.

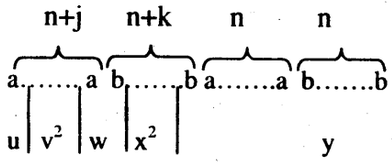
Case 2: Let v is in first a^n and x is in first b^n.

Let $v = a^j$, $x = b^k$ such that $|vx| = j + k >= 1$ and $|vwx| \leq n$



Note: w has a's and b's

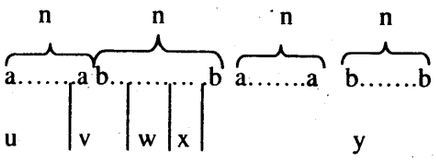
Now, according to Pumping Lemma, $uv^2wx^2y \in L$ for i = 2. The string uv^2wx^2y generated is as shown below:



Note that $a^{n+j}b^{n+k}a^n b^n = uv^2wx^2y \notin L$ when j+k >=1 (Note that the string ww of the form is not generated). But by pumping lemma, $uv^2wx^2y \in L$, which is the contradiction.

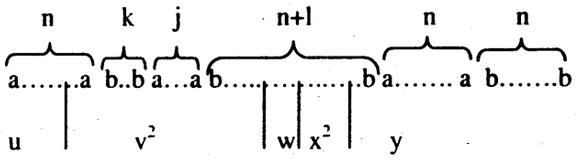
Case 3: Let v overlaps the first $a^n b^n$ and x is the first b^n.

Let $v = a^j b^k$ and $x = b^l$ such that $|vx| = j + k + l \geq 1$ and $|vwx| \leq n$ as shown below.



Note: v has a's and b's, w and x has b's, y has b's followed by $a^m b^m$.

According to Pumping Lemma, $uv^2wx^2y \in L$ for i = 2. The string uv^2wx^2y generated is as shown below:



It is very clear from the above figure that $uv^2wx^2y \notin L$ when j+k+l ≥ 1. (Note that the string ww of the form is not generated). But by pumping lemma $uv^2wx^2y \in L$ which is contradiction.

In all the three cases we get the contradiction. Even in one of the cases we get the contradiction, we can say that the language generated is not context free. So, it has been shown that the language

$$L = \{ww \mid w \in \{a,b\}^*\}$$

is not context free.

Example 8.4: Show that $L = \{a^{n!} \mid n \geq 0\}$ is not context free.

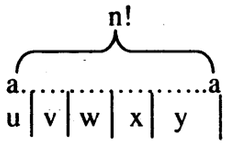
Step1: Assume that L is context-free and is infinite. Let $z = a^{n!} \in L$ where $|a^{n!}| > n$.

Step 2: Since |z| > n, according to Pumping Lemma we can split $z = a^{n!}$ into u, v, w, x and y such that

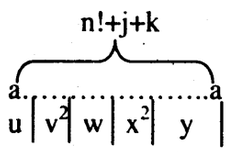
$$|vwx| \leq n \text{ and } |vx| \geq 1$$

so that $uv^iwx^iy \in L$ for for i = 0, 1, 2....

The splitting of the string z is shown in figure below.



Let $v = a^j$, $x = a^k$ such that $|vx| = j + k \geq 1$ and $|vwx| \leq n$. The string uv^2wx^2y can be generated and is shown in the figure below.



$$uv^2wx^2y = a^{n!+j+k} \text{ whenever } j+k \geq 1$$

When n = 2,

$$n!+j+k = n! + j + k \leq n!+n$$
$$< n! + n!n$$
$$= n!(n+1)$$
$$= (n+1)!$$

$$n! < n!+j+k < (n+1)!$$

Since n!+j+k it lies between n! and (n+1)!, the string generated

$$uv^2wx^2y = a^{n!+j+k} \notin L$$

which is a contradiction. So, the language

$$L = \{a^{n!} \mid n \geq 0\}$$

is not context free.

Example 8.5: Show that $L = \{a^p b^q \mid p = q^2\}$ is not context free.

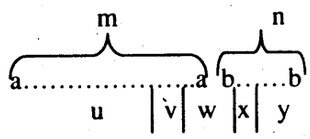
Step 1: Assume that L is context-free and is infinite. So, we can apply Pumping Lemma. Let $z = a^m b^n \in L$ where $m = n^2$ and $|a^m b^n| > n$.

Step2: Since $|z| > n$, according to Pumping Lemma we can split the string $z = uvwxy = a^m b^n$ such that

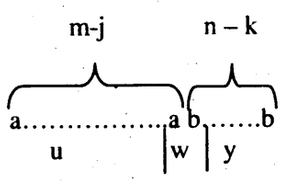
$$|vwx| \leq n \text{ and } |vx| \geq 1$$

so that $uv^iwx^iy \in L$ for for i = 0, 1, 2....

Step 3: Assume that the string vwx is within $a^m b^n$. Let $v = a^j$ $x = b^k$ such that $|vx| = j+k \geq 1$.



when $j+k \geq 1$, and $i = 0$, then we have uv^iwx^iy as shown below:



i.e.,

$$a^{m-j}b^{n-k} = uwy \text{ and } j \neq 0 \text{ and } k \neq 0$$

$$\Downarrow \quad \text{(implies)}$$

$$(n-k)^2 \leq (n-1)^2$$
$$= n^2 - 2n + 1$$
$$= m - 2n + 1 \text{ (since } m = n^2)$$
$$< m - j$$

$$\Downarrow \quad \text{(implies)}$$

$$m-j \neq (n-k)^2$$

It means that m-j is not a perfect square. According to Pumping Lemma it should be a perfect square. So,

$$a^{m-j}b^{n-k} = uwy \notin L$$

which is contradiction

Step 4: So, the language

$$L = \{a^pb^q \mid p = q^2\}$$

is not context free.

8.3 CFLs are closed under Union, concatenation and star

Theorem: If L_1 and L_2 are CFLs, then $L_1 \cup L_2$, $L_1.L_2$ and L_1^* also denote the CFLs and so the Context Free Languages are closed under union, concatenation, start-closure.

Proof: Let L_1 and L_2 are two CFLs generated by the CFGs

$$G_1 = (V_1, T_1, P_1, S_1)$$

and

$$G_2 = (V_2, T_2, P_2, S_2)$$

respectively and assume that V_1 and V_2 are disjoint.

Case 1: Union of two CFLs is CFL

Now, let us consider the language L_3 generated by the grammar

$$G_3 = (V_1 \cup V_2 \cup S_3, T_1 \cup T_2, P_3, S_3)$$

where

- S_3 is a the start symbol for the grammar G_3 and $S_3 \notin (V_1 \cup V_2)$
- $P_3 = P_1 \cup P_2 \cup \{S_3 \rightarrow S_1 \mid S_2\}$

It is clear from this that the grammar G_3 is context free and the language generated by this grammar is context free. It is easy to prove that

$$L_3 = L_1 \cup L_2$$

If we assume $w \in L_1$, then the possible derivation from S_3 is

$$S_3 \Rightarrow S_1 \overset{*}{\Rightarrow} w$$

On similar lines if we assume $w \in L_2$, then the possible derivation from S_3 is

$$S_3 \Rightarrow S_2 \overset{*}{\Rightarrow} w$$

So, if $w \in L_3$, one of the derivations

$$S_3 \Rightarrow S_1$$

or

$$S_3 \Rightarrow S_2$$

is possible. In the first case, all the variables in V_1 and all the terminals in T_1 may be used to get the derivation

$$S_1 \overset{*}{\Rightarrow} w$$

which uses only the productions in P_1. Similarly all the variables in V_2 and all the terminals in T_2 may be used to get the derivation

$$S_2 \overset{*}{\Rightarrow} w$$

which uses only the productions in P_2 and it follows that

$$L_3 = L_1 \cup L_2$$

Thus, it is proved that context free languages are closed under union

Case 2: Concatenation of two CFLs is CFL

Now, let us consider the language L_4 generated by the grammar

$$G_4 = (V_1 \cup V_2 \cup S_4, T_1 \cup T_2, P_4, S_4)$$

where

- S_4 is a the start symbol for the grammar G_4 and $S_4 \notin (V_1 \cup V_2)$
- $P_4 = P_1 \cup P_2 \cup \{S_4 \rightarrow S_1 S_2\}$

It is clear from this that the grammar G_4 is context free and the language generated by this grammar is context free and so

$$L_3 = L_1 L_2$$

Thus, it is proved that context free languages are closed under concatenation.

Case 3: CFLs are closed under star-closure

Now, let us consider the language L_5 generated by the grammar

$$G_5 = (V_1 \cup S_5, T_1, P_5, S_5)$$

where

- S_5 is a the start symbol for the grammar G_5
- $P_5 = P_1 \cup \{S_5 \rightarrow S_1 S_5 | \varepsilon\}$

It is clear from this that the grammar G_5 is context free and the language generated by this grammar is context free and so

$$L_5 = L_5^*$$

Thus, it is proved that context free languages are closed under star-closure.

Thus, we have proved that the context free languages are closed under union, concatenation and star-closure.

8.4 CFLs are not closed under intersection

Theorem: The CFLs are not closed under intersection. If L_1 and L_2 are context free languages, it is not always true that L1 $\cap$ L2 is context free language.

Proof: Let us prove this theorem by taking counter examples. Consider the two languages

$$L_1 = \{a^n b^n c^m \mid n \geq 0, m \geq 0\}$$

and

$$L_2 = \{a^n b^m c^m \mid n \geq 0, m \geq 0\}$$

The two languages are context free, as we can easily obtain the corresponding context free grammars

$$\begin{aligned} S &\rightarrow S_1 S_2 \\ S_1 &\rightarrow aS_1 b \mid \varepsilon \\ S_2 &\rightarrow CS_2 \mid \varepsilon \end{aligned}$$

and

$$\begin{aligned} S &\rightarrow aS \mid S_1 \\ S_1 &\rightarrow bS_1 c \mid \varepsilon \end{aligned}$$

Now, let us take

$$L_1 \cap L_2 = \{a^n b^n c^n \mid n \geq 0\}$$

We have already proved earlier that this language is not context free. Thus we can prove that the family of context free languages is not closed under intersection.

8.5 CFLs are not closed under complementation

Theorem: The CFLs are not closed under complementation. If L is context free language, it is not true that complement of L is context free language.

Proof: Let us prove this theorem by contradiction. Suppose that context-free languages *are* closed under complementation. So, if L_1 and L_2 are context-free languages, then $\overline{L}_1$ and $\overline{L}_2$ are also context free. We have already proved that CFLs are closed under union.

So,

$$\overline{L}_1 \cup \overline{L}_2$$

must be context free. Since we have assumed that the CFLs are closed under complementation,

$$\overline{\overline{L}_1 \cup \overline{L}_2}$$

must be context free. But, according to de Morgan's law

$$\overline{\overline{L}_1 \cup \overline{L}_2} = L_1 \cap L_2$$

So,

$$L_1 \cap L_2$$

must be context free which is a contradiction (As we have already proved that the context free languages are not closed under intersection). Since the context free languages are not closed under intersection, our assumption that the CFLs are closed under complementation is not true. So, the family of context free languages are closed under complementation.

Note: We have seen that the following languages

1. $L = \{a^n b^n c^n \mid n \geq 0\}$
2. $L = \{w \mid w \in \{a,b,c\}^* \text{ where } n_a(w) = n_b(w) = n_c(w)\}$
3. $L = \{ww \mid w \in \{a,b\}^*\}$
4. $L = \{a^{n!} \mid n \geq 0\}$
5. $L = \{a^p b^q \mid p = q^2\}$

are not context free and it is not possible to represent using context free grammars and so we can not have the corresponding PDA for these types of languages. Even then some of the important points we can make at this stage are:

1. The regular languages are the subset of context free languages and so context free languages are more powerful than the regular languages.
2. The statement 1 automatically implies that PDAs are more powerful than the finite automaton.
3. But, the PDAs are not strong enough to accept some of the languages as pointed out earlier that are not context free and so we need much more powerful automaton than the PDA such as Linear bounded Automat or Turing Machine. Let us concentrate on Turing Machines in the next chapter.

Exercises:

1. State and prove Pumping Lemma for context free languages?
2. What are the applications of Pumping Lemma?
3. Show that $L = \{a^n b^n c^n \mid n \geq 0\}$ is not context free.
4. Show that $L = \{w \mid w \in \{a,b,c\}^* \text{ where } n_a(w) = n_b(w) = n_c(w)\}$ is not context free
5. Show that $L = \{ww \mid w \in \{a,b\}^*\}$ is not context free.
6. Show that $L = \{a^{n!} \mid n \geq 0\}$ is not context free.
7. Show that $L = \{a^p b^q \mid p = q^2\}$ is not context free
8. Prove that CFLs are closed under union, concatenation and star-closure
9. Prove that CFLs are not closed under intersection
10. Prove that CFLs are not closed under complementation

Summary

Now!! We know

- Pumping Lemma for CFLs
- Proof of Pumping Lemma for CFLs
- Applications of Pumping lemma for CFLs
- Solution to various problems and to show that the specified languages are not context free.
- CFLs are not closed under union
- CFLs are not closed under concatenation
- CFLs are not closed under star-clousre (kleene-closure)
- CFLs are not closed under intersection
- CFLs are not closed under complementation
- How to design Turing machines for various types of problems.

9

Turing Machines

What we will know after reading this chapter?

- Concept of Turing Machine model
- Definition of Turing machine (Standard Turing machine)
- Definition of Instantaneous description of TM
- Moves of TM
- Languages accepted by TM
- Recursively enumerable language
- Constructing TMs for varieties of languages
- TM as transducer

Turing Machine (TM) is modified version of the PDA and it is much more powerful than PDA. Instead of using stack as in PDA, the TM uses the tape to store the symbols. The Turing machine is a generalized machine which can recognize all types of languages viz, regular languages (generated from regular grammar also known as type 3 grammar), context free languages (generated from context free grammars also known as type 2 grammars) and context sensitive language (generated from context sensitive grammar also known as type 1 grammar). Apart from these languages, the turing machine also accepts the language generated from type 0 grammar (also known as unrestricted grammar). Thus, Turing machine can accept any language. This chapter mainly concentrates on building the turing machines for any language.

9.1 Turing machine Model

The Turing machine model is shown in figure 9.1. It is a finite automaton connected to read-write head with the following components:

- Tape
- Read-write head
- Control unit

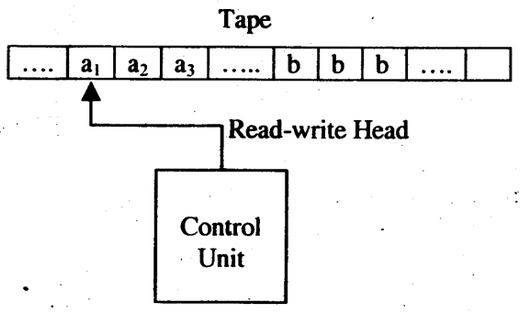


Fig 9.1 Turing machine model

Tape is used to store the information 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 leftmost position on the tape. The string to be scanned should end with blanks. The tape is assumed to be infinite both on left side and right side of the string.

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 it points to.

Control Unit The reading from the tape or writing into 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 control unit consults *action table* i.e., *transition table* and carry out the tasks.

The read-write head can move either towards left or right i.e., movement can be on both the directions. The various actions 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.

If there is no entry in the table for the current combination of symbol and state, then the machine will halt. The Turing machines can be represented using various notations such as

- Transition tables
- Instantaneous descriptions
- Transition diagram

9.2 Transition Table

Consider the transition table shown in table 9.1. Later sections describe how to obtain the transition table 9.1. 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. The symbols a, b, X, Y and B are denoted by Γ and $\Sigma \subseteq \Gamma$. 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 in the table 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.

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	-	-	-	-	-

Table 9.1 Transition table

The transitions shown in the table can also be written as

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

δ: Q x Γ to Qx Γ x{L or R})

In general, δ can be defined as follows:

δ: Q x Γ to (Q x Γ x {L, R}) i.e., cross product of Q, Γ and {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 start state
B is a special symbol indicating blank character
$F = \{q_4\}$ which is the final state.

Thus, formally 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 from $Q \times \Gamma$ to $Q \times \Gamma \times \{L,R\}$
q_0 is the start state
B is a special symbol indicating blank character
$F \subseteq Q$ is set of final states.

Since there can be several variations of TM (which we see in the coming chapters), the TM that we discuss now can be called **standard Turing Machine** with the following features:

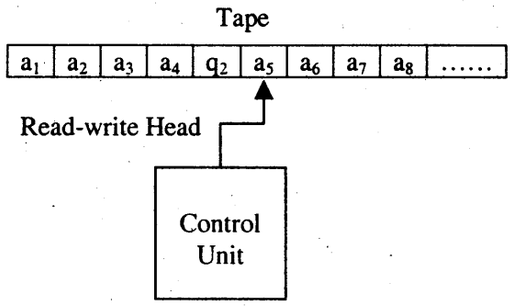
1. The Turing machine has a tape that is divided into number of cells with each cell capable of storing only one symbol. The tape is unbounded (i.e., no boundary on the left as well as on the right) with any number of left or right moves.
2. The machine is deterministic. It can have either zero or one transition for each configuration.
3. Some of the symbols on the tape can be considered as the input. The symbols on the tape (some symbols or all the symbols) can be considered as the output whenever the TM halts.

9.3 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. The formal definition of *instantaneous description* (ID) in case of TM is defined as shown below:

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 9.1: 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 which is the next symbol to be scanned and the symbol towards left of a_5 i.e., q_2 is the current state. So, in this case an ID is denoted by

$$\underbrace{a_1a_2a_3a_4}_{\alpha}\underbrace{q_2a_5a_6a_7a_8}_{\beta}\ldots.$$

α q β general form of ID

where the substring

$$a_1a_2a_3a_4$$

towards left of the state q_2 is the left sequence, the substring

$$a_5a_6a_7a_8\ldots.$$

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

$$a_1a_2a_3a_4q_2a_5a_6a_7a_8\ldots.$$

as shown in snapshot of example 9.1. 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

$$a_1a_2a_3a_4b_1q_3a_6a_7a_8\ldots.$$

This can be represented by a *move* as shown below:

$$a_1a_2a_3a_4q_2a_5a_6a_7a_8\ldots. \vdash a_1a_2a_3a_4b_1q_3a_6a_7a_8\ldots.$$

Similarly if the current ID of the Turing machine is

$$a_1a_2a_3a_4q_2a_5a_6a_7a_8\ldots.$$

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

$$a_1a_2a_3q_1a_4c_1a_6a_7a_8\ldots.$$

This can be represented by a *move* as shown below:

$$a_1a_2a_3a_4q_2a_5a_6a_7a_8\ldots. \vdash a_1a_2a_3q_1a_4c_1a_6a_7a_8\ldots.$$

This configuration indicates that the new state is q_1, the next input symbol to be scanned is a_4. In general, 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.

The formal definition of *move* for TM is shown below:

Definition: Let $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ be a TM. Let the ID of M be

$$a_1a_2a_3\ldots a_{k-1}qa_ka_{k+1}\ldots.a_n$$

where $a_j \in \Gamma$ for $1 \leq j \leq n$, $q \in Q$ is the current state and a_k as the next symbol to be scanned. If there is a transition

$$\delta(q, a_k) = (p, b, R)$$

then the *move* of machine M will be

$$a_1a_2a_3\ldots a_{k-1}qa_ka_{k+1}\ldots.a_n \vdash a_1a_2a_3\ldots a_{k-1}bpa_{k+1}\ldots.a_n$$

If there is a transition

$$\delta(q, a_k) = (p, b, L)$$

then the *move* of machine M will be

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

9.4 Acceptance of a language by TM

Note: The Turing Machine can do one of the following things:

a. Halt and accept by entering into final state.
b. Halt and reject. This is possible if the transition is not defined i.e., $\delta(q, a)$ is not defined.
c. TM will never halt and enters into an infinite loop.

It is true that there is no algorithm to determine and tell whether a given machine always halts

The language accepted by TM is defined as follows.

Definition: Let M = (Q, Σ, Γ, δ, q_0, B, F) be a TM. The language L(M) accepted by M is defined as

$$L(M)= \{w \mid q_0w \vdash^* \alpha_1p\alpha_2 \text{ where } w \in \Sigma^*, p \in F \text{ and } \alpha_1, \alpha_2 \in \Gamma^*\}$$

where q_0w is the initial ID and $\alpha_1p\alpha_2$ is the final ID. The set of all those words w in Σ^* which causes M to move from start state q_0 to the final state p.

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 final state and halts, then we say that the string *w* is accepted by Turing machine. The language accepted by TM is called **recursively enumerable language** or **RE language**. The formal definition is shown below:

Definition: A language L is *recursively enumerable*, if it is accepted by a TM i.e., given a string *w* which is input to TM, the machine halts and outputs Yes if it belongs to the language. If w does not belong to the lanauge L, the TM halts and outputs NO.

The languages with Turing Machine which will always halts and output yes if it belongs to the language or output no if it does not belong to the language are called *decidable languages* or *recursive languages*. The Turing machines that always halt irrespective of whether they accept or not are a good model for an algorithm. If an algorithm exists to solve a given problem, then the problem is *decidable* otherwise it is un-decidable problem.

9.5 Construction of Turing Machine (TM)

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

Example 9.2: Obtain a Turing machine to accept the language

$$L = \{0^n1^n \mid n \geq 1\}$$

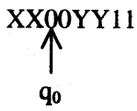
It is given that the language accepted by TM should have *n* number of 0's followed by *n* number of 1's. For this let us take an example of the string *w* = 00001111. The string *w* should be accepted as it has four zeroes followed by four 1's.

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.

Design: Consider the situation

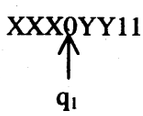


where first two 0's are replaced by Xs and first two 1's are replaced by Ys. In this situation, the read-write head points to the left most zero and the machine is in state q_0. With this as the configuration, now let us design the TM.

Step1: 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

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

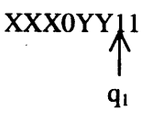
The resulting configuration is shown below.



Step2: In state q_1, we have to obtain the left-most 1 and replace it by Y. So, let us move the pointer to point to leftmost 1. 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

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

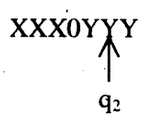
When these transitions are repeatedly applied, the following configuration is obtained.



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

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

The resulting configuration is shown below.

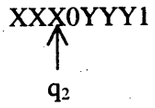


Note that the pointer should be moved towards left. This is because, a zero is replaced by X and the corresponding 1 is replaced by Y. Now, we have to scan for the left most 0 and so, the pointer was move towards left.

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

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

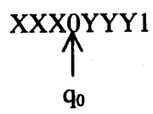
The following configuration is obtained.



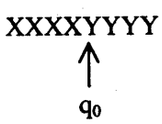
Step 5: Now, we have obtained the right most X. To get leftmost 0, replace X by X, change the state to q_0 and move the pointer towards right. The transition for this can be of the form

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

and the following configuration is obtained.



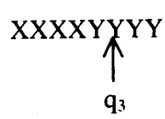
Now, repeating the steps 1 through 5, we get the configuration shown below:



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 0's 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

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

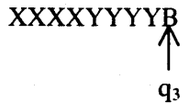
and the following configuration is obtained.



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

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

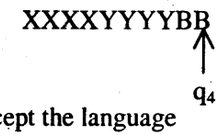
Repeatedly applying this transition, the following configuration is obtained.



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

$$\delta(q_3, B) = (q_4, B, R)$$

where q_4 is the final state and the following configuration is obtained.



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.

$$\begin{aligned}
\delta(q_0, 0) &= (q_1, X, R)\\
\delta(q_1, 0) &= (q_1, 0, R)\\
\delta(q_1, Y) &= (q_1, Y, R)\\
\delta(q_1, 1) &= (q_2, Y, L)\\
\delta(q_2, Y) &= (q_2, Y, L)\\
\delta(q_2, 0) &= (q_2, 0, L)\\
\delta(q_2, X) &= (q_0, X, R)
\end{aligned}$$

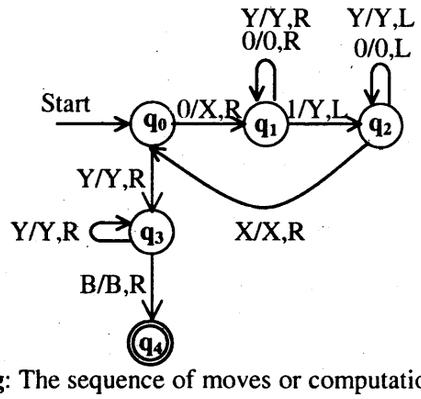
$\delta(q_0, Y) = (q_3, Y, R)$
$\delta(q_3, Y) = (q_3, Y, R)$
$\delta(q_3, B) = (q_4, B, R)$

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	-	-	-	-	-

9.6 Transition diagram for Turing Machine (TM)

The Turing Machine can be represented using transition diagram. The transition diagram consists of nodes corresponding to the states of Turing Machine. An edge from state *q* to state *p* will have a label of the form (X / Y, D) where X and Y are tape symbols and D is the direction either 'L' or 'R' where 'L' stands for *left* and 'R' stands for right i.e., the movement of the head can be either left or right. Here, X is the scanned symbol and Y is the symbol written on to the tape. The start state of the Turing Machine is indicated by an arrow entering the state with label 'Start'. The final states are represented by two concentric circles. The transition diagram for the example 9.2 is shown below:



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

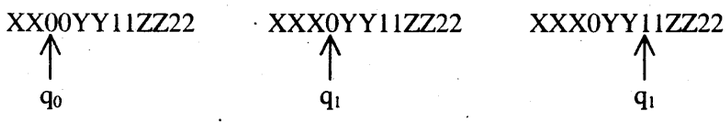
(Initial ID) $q_0 0011 \vdash X q_1 011 \vdash X 0 q_1 11 \vdash X q_2 0 Y 1 \vdash q_2 X 0 Y 1 \vdash X q_0 0 Y 1 \vdash X X q_1 Y 1 \vdash X X Y q_1 1 \vdash X X q_2 Y Y \vdash X q_2 X Y Y \vdash X X q_0 Y Y \vdash X X Y q_3 Y \vdash X X Y Y q_3 \vdash X X Y Y B q_4$(Final ID)

Since the final state q_4 is reached, the string 0011 is accepted.

Example 9.3: Obtain a Turing machine to accept the language

$$L(M) = \{0^n1^n2^n \mid n \geq 1\}$$

It is given that the language should consist of *n* number of 0's followed by *n* number of 1's which in turn should be followed by *n* number of 2's. Let us consider the string 000011112222 and we shall see how to design the Turing Machine. To design the Turing Machine, consider the situation where first two 0's are replaced by X's, first two 1's are replaced by Y's and first two 2's are replaced by Z's as shown in fig.9.2.a.



(a) (b) (c)

Fig. 9.2 Various configurations

Now, with fig 9.2.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 fig.9.2.b is obtained. The transition for this can be of the form

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

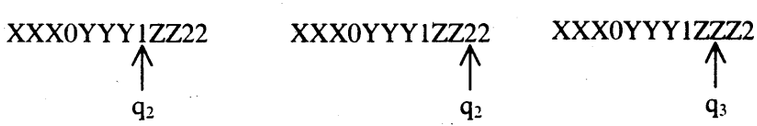
In state q_1, we have to search for the leftmost 1. It is clear from fig. 9.2.b that, when we are searching for the symbol 1, we may encounter the symbols 0 or Y. So, replace 0 by 0, Y by Y and move the pointer towards right and remain in state q_1 only. The transitions for this can be of the form

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

The configuration shown in fig. 9.2.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 fig.9.3.a is obtained.



(a) (b) (c)

Fig. 9.3 Various configurations

In state q_2, we have to search for the leftmost 2. It is clear from fig. 9.3.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 fig.9.3.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 fig.9.3.c is obtained. Once the TM is in state q_3, it means that first 0 is replaced by X, first 1 is replaced by Y and first 2 is replaced by Z. At this point, we have to search for the rightmost X to get leftmost 0. During this process, it is clear from fig.9.3.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. This is 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 $L = \{0^n1^n2^n \mid n \geq 1\}$ is given by

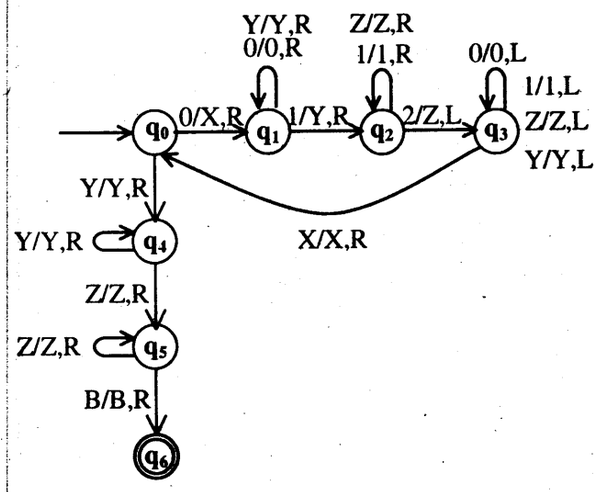
$$M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$$

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 start state
B is blank character
$F = \{q_6\}$ is the final state
δ is shown below using the transitional table.

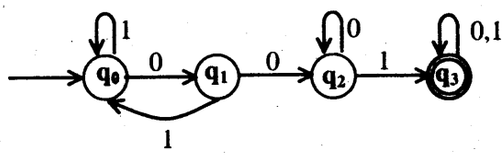
	Γ						
states	0	1	2	Z	Y	X	B
q_0	q_1,X,R				q_4,Y,R		
q_1	q_1,0,R	q_2,Y,R			q_1,Y,R		
q_2		q_2,1,R	q_3,Z,L	q_2,Z,R			
q_3	q_3,0,L	q_3,1,L		q_3,Z,L	q_3,Y,L	q_0, X,R	
q_4				q_5,Z,R	q_4,Y,R		
q_5				q_5,Z,R			(q_6,B,R)
q_6							

The transition diagram for this can be of the form



Example 9.4: Obtain a TM to accept the language L = {w | w ∈ (0+1)*} containing the substring 001

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 in chapter 3 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			

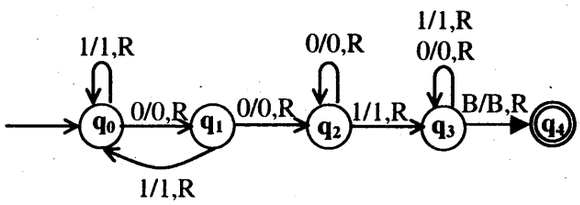
The TM is given by

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

where

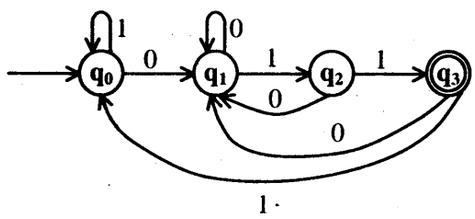
$Q = \{q_0, q_1, q_2, q_3, q_4\}$
$\Sigma = \{0, 1\}$
$\Gamma = \{0, 1\}$
δ- is shown in the form of transition table above
q_0 is the start state
B blank character
$F = \{q_4\}$ is the final state

The transition diagram for this is shown below.



Example 9.5: Obtain a Turing machine to accept the language containing strings of 0's and 1's ending with 011.

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 in chapter 3 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(similar to the example 9.5.). 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	-	-	-

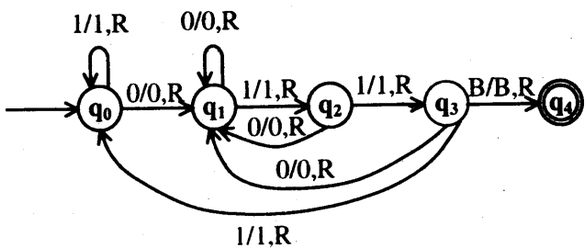
The TM is given by

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

where

$Q = \{q_0, q_1, q_2, q_3\}$
$\Sigma = \{0, 1\}$
$\Gamma = \{0, 1, B\}$
δ- is defined already
q_0 is the start state
B blank character
$F = \{q_4\}$ is the final state

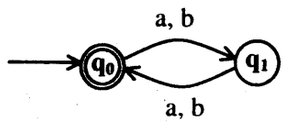
The transition diagram for this is shown below.



Example 9.6: Obtain a Turing machine to accept the language

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

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 in chapter 3 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(similar to the example 9.5.). 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	-	-	-

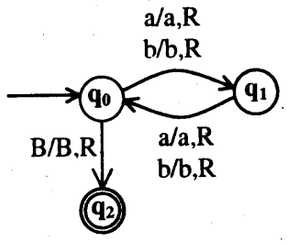
The TM is given by

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

where

$Q = \{q_0, q_1\}$
$\Sigma = \{a, b\}$
$\Gamma = \{a, b, B\}$
δ- is defined in the form of table above
q_0 is the start state
B blank character
$F = \{q_2\}$ is the final state

The transition diagram of TM is given by



Example 9.7: Obtain a TM to compute the function – which is called *monus* or proper subtraction and is defined by m – n = max(m-n, 0)

Note: The monus operation is defined as

$$m - n = m - n \text{ if } m \geq n$$

and

$$m - n = 0 \text{ if } m < n.$$

It is clear from this definition that the Turing Machine is supposed to perform *monus* operation and will not accept anything and so the concept of final state will not come into picture. To start with the tape consists of 0^m10^n which is surrounded by blanks and the machine halts with 0^{m-n} on its tape surrounded by blanks. Here, *m* number of 0's and *n* number of 0's are replaced by the delimiter 1.

General Procedure: The sequence of 0's is partitioned into first group with *m* number of 0's followed by a 1 and followed by second group with *n* number of 0's. The machine finds the leftmost 0 and is replaced by blank B. Then move towards right to search for 1. After finding 1, it searches leftmost 0 in the second group and is replaced by 1 and move towards left to get leftmost 0 in the first group. This procedure is repeated till one of the following conditions are satisfied:

- When searching for a 0 in second group, if B is encountered it means that *n* number 0's in the second group are replaced by 1's and n+1 zeros in the first group are changed to B's. Now, the second group will have n+1 ones. The machine replaces n+1 1's by one 0 and n B's and observe that only m – n 0's exists on the tape
- If the first group if the machine M can not find a 0 (since first m 0's have already been changed to B's) it means that m < n and so no 0's and 1's should be there on the tape.

The brief description of each state to achieve the above task is shown below:

In state q_0: On encountering a 0, change the state to q_1, replace 0 by B and move the head towards right using the transition

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

On encountering a 1 (means that all 0's in the first portion are replaced by B's) change the state to q_5 as shown below:

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

In state q1: we search for leftmost 1. Keep updating the head towards right till we encounter 1 replacing 0 by 0 and remaining in q_0. On encountering 1 change the state to q2 and move towards right using the transition

$$\delta(q_1, 0) = (q_1, 0, R)$$
$$\delta(q_1, 1) = (q_2, 1, R)$$

In state q2: if we encounter 0 replace it by 1, change the state to q_3 (to get leftmost 0 in left portion) and move the header towards left as shown below:

$$\delta(q_2, 0) = (q_3, 1, L)$$

If we encounter 1 replace it by 1, remain in q_2 and move the head towards right to obtain leftmost 0 as shown below:

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

If we encounter B, it means that no more 0's are found and change the state to q_4 which indicates n 0's out of m 0's are cancelled and subtraction is complete.

$$\delta(q_2, B) = (q_4, B, L)$$

Now, in state q4 we have to convert all 1's to blanks.

In state q3: To get leftmost 0, replace 1 by 1, replace 0 by 0, remain in q_3 and move the head towards left using the transitions:

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

On encountering B, change the state to q_0 and move the head towards right using

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

In state q4: Let us convert all 1's to blanks using the following transitions

$$\delta(q_4, 1) = (q_4, B, L)$$
$$\delta(q_4, 0) = (q_4, 0, L)$$

Note that out of m 0's, n+1 0's are replaced by blanks. But, we are supposed to replace only n 0's. So, one blank should be replaced by 0 and halt the machine by entering into state q_6.

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

In state q_5: In state q5, the output should be 0. The tape should not have any symbols, except B's. So, replace all 1's and all remaining 0's with B's using the transitions

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

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

On encountering B, change the state to q_6.

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

So, the TM for monus function is shown below:

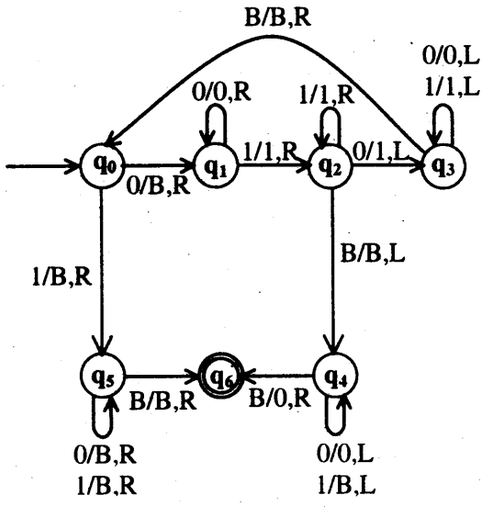
$$M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$$

where

$Q = \{q_0, q_1, q_2, q_3, q_4, q_5, q_6\}$
$\Sigma = \{0, 1\}$
$\Gamma = \{0, 1, B\}$
q_0 is the start state
B is the blank character
$F = \phi$
δ is shown below using the transition table

δ	0	1	B
q_0	q_1, B, R	q_5, B, R	
q_1	q_1, 0, R	q_2, 1, R	
q_2	q_3, 1, L	q_2, 1, R	q_4, B, L
q_3	q_3, 0, L	q_3, 1, L	q_0, B, R
q_4	q_4, 0, L	q_4, B, L	q_6, 0, R
q_5	q_5, B, R	q_5, B, R	q_6, B, R
$*q_6$			

The transition diagram for this can be of the form



The sequence of moves made by the TM for the string 0000100B outputting m-n is shown below:

$Bq_00000100B \vdash BBq_1000100B \vdash BB0q_100100B \vdash BB00q_10100B \vdash$
$BB000q_1100B \vdash BB0001q_200B \vdash BB000q_3110B \vdash BB00q_30110B \vdash$
$BB0q_300110B \vdash BBq_3000110B \vdash Bq_3B000110B \vdash BBq_0000110B \vdash$
$BBBq_100110B \vdash BBB0q_10110B \vdash BBB00q_1110B \vdash BBB001q_210B \vdash$
$BBB0011q_20B \vdash BBB001q_311B \vdash BBB00q_3111B \vdash BBB0q_30111B \vdash$
$BBBq_300111B \vdash BBq_3B00111B \vdash BBBq_000111B \vdash BBBBq_10111B \vdash$
$BBBB0q_1111B \vdash BBBB01q_211B \vdash BBBB011q_21B \vdash BBBB0111q_2B \vdash$
$BBBB011q_41B \vdash BBBB01q_41BB \vdash BBBB0q_41BBB \vdash BBBBq_40BBBB \vdash$
$BBBq_4B0BBBB \vdash BBB0q_60BBBB$

Since q_6 on 0 is not defined, the Turing Machine halts. Observe that number of 0's on the tape is 2 which is 4-2. The sequence of moves made by the machine for the string B0100B is shown below:

(Initial ID) $Bq_00100B \vdash BBq_100B \vdash BB1q_200B \vdash BBq_3110B \vdash Bq_3B110B \vdash$
$BBq_0110B \vdash BBBq_510B \vdash BBBBq_50B \vdash BBBBBq_5B \vdash BBBBBBq_6$

Since the transition is not defined for the state q_6, the Turing machine halts. Observe that no zeros are present on the tape since number of 0's in first portion is less than the number of 0's in the second portion.

Example 9.8: Obtain a TM to accept a string w of a's and b's such that $N_a(w)$ is equal to $N_b(w)$ i.e., the number of a's and b's in the string w should be equal.

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 which can either be *a* or *b*. The general procedure to design a TM will result in three cases depending on the next input symbol to be scanned namely:

1. On encountering B
2. On encountering *a*
3. On encountering *b*

Case 1: On encountering B

Change the state from q_0 to q_f, replace B by B and move the pointer towards right and the Turing machine halts. The transition for this is shown below:

$$\delta(q_0, B) = (q_f, B, R)$$

Case 2: On encountering a

General procedure: In state q_0, if we encounter *a*, we skip all the subsequent symbols till we encounter *b*. Then come back to the next leftmost symbol and repeat any of the three cases based on the next symbol to be scanned.

Detail procedure: The first *a* is replaced by X and the first *b* is replaced by Y. For example, consider the string *aaababbb* and consider the scenario where first two *a*'s replaced by X's and first two *b*'s are replaced by Y's and the read-write head points to the next symbol to be scanned as shown below:

XXaYaYbb
↑
q_0

- In state q_0, on encountering *a*, change the state to q_1, replace *a* by X and move the pointer towards right to get leftmost *b*. The corresponding transition is
$$\delta(q_0, a) = (q_1, X, R)$$

- It is clear from the figure that when we search for leftmost *b*, we may get *a* or Y. In such cases, the head should move towards right replacing *a* by *a*, Y by Y and remaining in state q_1. The corresponding transitions are:

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

- In state q_1 on encountering *b*, replace *b* by Y, change the state to q_2 and move the pointer towards left to get the next rightmost X. The corresponding transition is:

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

- When searching for X, we may encounter Y's or *a*'s. In such cases remain in q_2 only and move the head towards left. The corresponding transitions are:
$$\delta(q_2, Y) = (q_2, Y, L)$$
$$\delta(q_2, a) = (q_2, a, L)$$

- In state q_2 on encountering X change the state to q_0, replace X by X and move the pointer towards right using the transition

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

- In state q_0, on encountering Y it indicates that so far the number of a's and b's are equal and so simply move the pointer towards right using the transition

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

- Repeat one of the three cases

Case 3: On encountering b

General procedure: In state q_0, if we encounter *b,* we skip all the subsequent symbols till we encounter *a*. Then come back to the next leftmost symbol and repeat any of the three cases based on the next symbol to be scanned.

Detail procedure: The first *b* is replaced by X and the first *a* is replaced by Y. For example, consider the string *bbbabaaa* and consider the scenario where first two *b*'s replaced by X's and first two *a*'s are replaced by Y's and the read-write head points to the next symbol to be scanned as shown below:

XXbYbYaa
  ↑
  q_0

- In state q_0, on encounter change the state to q_3, replace *b* by X and move the pointer towards right to get leftmost *a*. The corresponding transition is

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

- It is clear from the figure that when we search for leftmost *a*, we may get *b* or Y. In such cases, the head should move towards right replacing *b* by *b*, Y by Y and remaining in state q_3. The corresponding transitions are:

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

- On encountering *a*, replace *a* by Y, change the state to q_4 and move the pointer towards left to get the next rightmost X. The corresponding transition is:

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

- When searching for X, we may encounter Y's or *b*'s. In such cases remain in q_4 only and move the head towards left. The corresponding transitions are:

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

- In state q_4 on encountering X change the state to q_0, replace X by X and move the pointer towards right using the transition

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

- In state q_0, on encountering Y it indicates that so far the number of a's and b's are equal and so simply move the pointer towards right using the transition

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

- Repeat one of the three cases

So, the TM to accept strings of a's and b's such that number of a's is equal to number of b's is

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

where

$Q = \{q_0, q_1, q_2, q_3, q_4, q_f\}$
$\Sigma = \{a, b\}$
$\Gamma = \{a, b, X, Y, B\}$
q_0 is the start state
B is the blank character
$F = \{q_f\}$
δ is shown below using the transition table

	Γ				
δ	a	b	X	Y	B
→ q_0	q_1, X, R	q_3, X, R		q_0, Y, R	q_f, B, 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_4, Y, L	q_3, b, R		q_3, Y, R	
q_4		q_4, b, L	q_0, X, R	q_4, Y, L	
*q_f	Final state				

The sequence of moves made by the Turing Machine for the string bbabaa is shown below:

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

(Initial ID) $q_0bbabaa \vdash Xq_3babaa \vdash Xbq_3abaa \vdash Xq_4bYbaa \vdash q_4XbYbaa \vdash$

$Xq_0bYbaa \vdash XXq_3Ybaa \vdash XXYq_3baa \vdash XXYbq_3aa \vdash XXYq_4bYa \vdash$

$XXq_4YbYa \vdash Xq_4XYbYa \vdash XX\,q_0YbYa \vdash XXYq_0bYa \vdash XXYXq_3Ya \vdash$

$XXYXYq_3a \vdash XXYXq_4YY \vdash XXYq_4XYY \vdash XXYXq_0YY \vdash XXYXYq_0Y \vdash$

$XXYXYYq_fB$ (Final ID)

Similarly the sequence of moves for the string *bbaaab* is shown below:

(Initial ID) $q_0bbaaab \vdash Xq_3baaab \vdash Xbq_3aaab \vdash Xq_4bYaab \vdash q_4XbYaab \vdash$

$Xq_0bYaab \vdash XXq_3Yaab \vdash XXYq_3aab \vdash XXYaq_3ab \vdash$ ..... and so on.

Example 9.9: Obtain a Turing machine to accept a palindrome consisting of a's and b's of any length.

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 should be same and so on. The procedure to accept only string of palindromes is shown below. Let q_0 be the start state of Turing machine.

Step1: Move the read-write head to point to the first character of the string. The transition for this can be of the form

$$\delta(q_0, B) = (q_1, B, R)$$

Step2: In state q_1, if the first character is *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 be 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 step5.

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 step1.

So, the TM to accept strings of palindromes over {a,b} is given by

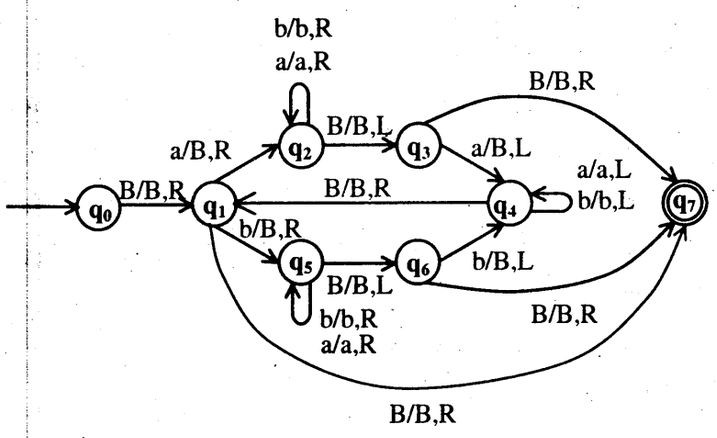
$$M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$$

where

$Q = \{q_0, q_1, q_2, q_3, q_4, q_5, q_6, q_7\}$
$\Sigma = \{a, b\}$
$\Gamma = \{a, b, B\}$
q_0 is the start state
B is the blank character
$F = \{q_7\}$
δ is shown below using the transition table

	Γ		
δ	a	b	B
$\rightarrow q_0$	-	-	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 9.10: Obtain a TM to accept the language L = {ww^R | w ∈ (a+b)* }

Note: ww^R is nothing but a palindrome but of even length. So, it is same as the previous problem except that from states q_3 and q_6 on B no transitions are defined as shown below:

	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_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	-	-	-

9.7 Transducers

A transducer accepts some input and transform that input into the desired output. In this sense, the TM can be called as a transducer. The primary purpose of any computer is to accept some input and transform into the desired output. Using Turing machines, an abstract model of a digital computer can be obtained. The input for Turing machine will be the non-blank symbols on the tape and after processing, the output will be the symbols on the tape. So, the transducer for a Turing machine is a function *f* defined by

$$f(w) = w'$$

where w is the input before computation and w' is the output after computation such that

$$q_0 w \vdash^* q_f w' \text{ for } q_f \in F$$

Definition: Let $M = (Q, \Sigma, \delta, q_0, B, F)$ be a Turing machine. The function f is Turing computable (also called computable) if and only if

$$q_0 w \vdash^* q_f w' \text{ for } q_f \in F \text{ and } w \in \Gamma^*$$

The arithmetic operations such as addition, subtraction etc. including the common mathematical functions are Turing computable. Some of the operations covered are:

- Addition
- Concatenation of two strings
- Arithmetic comparison

Example 9.11: Let x and y are two positive integers. Obtain a Turing machine to perform x + y

Let us see how to represent positive integers. We know that binary digits are 0 and 1. On similar lines, we can have a unary number which is made up of only one digit. Let us assume that 1 is the unary digit. So, a number is made up of only 1's. Let x and y are two unary numbers over $\{1\}^+$. Assume that both the unary integers x and y are stored on the tape one after the other separated by a 0. For example, if x is 1111 and y is 111111 then store x on the tape, end with a 0 and then store integer y as shown below

$$\underbrace{1111}_{x}\ 0\ \underbrace{111111}_{y}$$

If this is the input to the transducer, the output should be of the form

$$\underbrace{1111}_{x}\ \underbrace{111111}_{y}\ 0$$

In general the moves made by the Turing machine should be of the form

$$q_0 x 0 y \vdash^* q_f x y 0 \text{ where } q_f \text{ is the final state}$$

It is clear from the problem definition that to solve this problem the following steps are performed:

General Procedure

Keep updating the pointer till a 0 is encountered. Replace the symbol 0 by 1 and move till the last 1 is reached. Replace last 1 by 0 and reset the read-write head to point the first 1 on the tape.

Let q_0 be the start state and assume that the integers x and y are separated by 0 and enclosed between two B's as shown below

BxOyB

and the read-write head points to the first 1 in the integer x. The TM can be constructed as shown below:

Keep updating read-write head till a 0 is encountered. While scanning for a 0, we encounter 1's in x. So, replace 1 by 1 and move the read-write head towards right and stay in the state q_0. The transition for this can be of the form

$$\delta(q_0,1) = (q_0,1,R)$$

On encountering a 0, change the state to q_1, replace 0 by 1 and move the pointer towards right. The transition for this can be of the form

$$\delta(q_0,0) = (q_1, 1, R)$$

Now, the read-write head points to the first 1 of integer y. Now, move the read-write head to point to the last 1 of integer y. To achieve this, replace 1 by 1, move the read-write head towards right and stay in q_1 only. The transition for this can be of the form

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

On encountering B, change the state to q_2, replace B by B and move the pointer towards left. The transition for this can be of the form

$$\delta(q_1, B) = (q_2, B, L)$$

Now, the read-write head points to the last 1 of integer y. Now, change that 1 to 0, change the state to q_3 and move the head towards left. The transition for this can be of the form

$$\delta(q_2, 1) = (q_3, 0, L)$$

Now, we have the pattern

xy0

on the tape. But, we should move the pointer to the first 1 of the integer x. To achieve this, scan each symbol, replace 1 by 1, move the pointer towards left and remain in state q_3. The transition for this can be of the form

$$\delta(q_3, 1) = (q_3, 1, L)$$

Once the symbol B is encountered, replace B by B, change the state to q_4 and move the read-write towards right. The transition for this is

$$\delta(q_3, B) = (q_4, B, R)$$

So, the TM to achieve x + y is given by

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

where

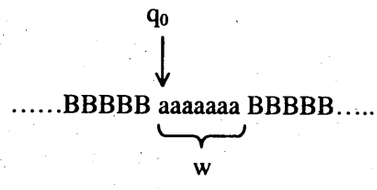
$Q = \{q_0, q_1, q_2, q_3, q_4\}$
$\Sigma = \{1\}$
$\Gamma = \{1, 0, B\}$
q_0 is the start state
B is blank character
$F = \{q_4\}$
δ is shown below using the transition table

δ	Γ		
	1	0	B
→ q_0	q_0,1,R	q_1, 1, R	-
q_1	q_1, 1, R	-	q_2, B, L
q_2	q_3, 0, L	-	-
q_3	q_3, 1, L	-	q_4, B, R
*q_4	-	-	-

It is left to the reader to take an example and show the sequence of moves made by the TM. By looking at the transitional table we can easily write the transition diagram which is also left to the reader as an exercise.

Example 9.12: Given a string *w*, design a TM that generates the string ww where $w \in a^+$.

Let q_0 be the start state and assume that the string *w* is enclosed between infinite number of B's and the read/write head points to the first *a* of the string *w* as shown below:



The general procedure to concatenate the string *w* with itself is shown below:

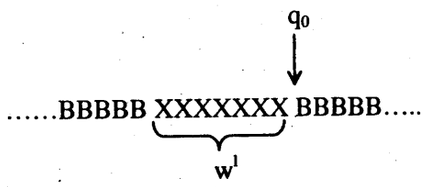
1. Replace each symbol in *w* with x.
2. Find the rightmost x
3. Replace rightmost x by the symbol *a*.
4. Move to the right of rightmost *a* and replace it by *a*.
5. Find the rightmost x
6. Repeat through step 3 till we find no more x's

Let us obtain the transitions for each of the steps shown above:

Step1: Replace each symbol in *w* with x. This can be easily done by replacing each *a* by the symbol X and then move the read/write head towards right till we get the symbol 'B'. The transitions defined to achieve this are:

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

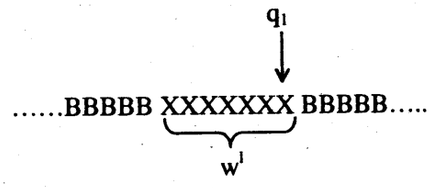
The contents of the tape and position of read/write head after applying these transitions will be



Step 2: Find the rightmost x. This is achieved only after all a's are replaced by X's as shown in figure above. In state q_0, once we encounter the symbol 'B' as the input, change the state to q_1, replace B by B and move the pointer towards left. The corresponding transition will be

$$\delta(q_0,B) = (q_1,B,L)$$

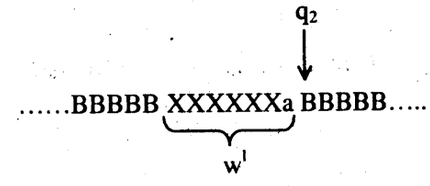
Now, the pointer points to the rightmost X as shown in the figure below:



Step 3: Replace rightmost x by the symbol *a*. Since the pointer points to the rightmost x, replace this X by *a*, change the state to q_2 and move the pointer towards right. The transition for this will be

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

The contents of the tape and position of read/write head after applying these transitions will be



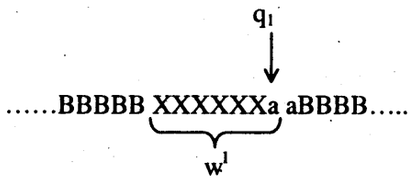
Step 4: Move to the right of rightmost *a* and replace it by *a*. This can be achieved by repeatedly replacing they symbol *a* by *a*, remain in the same state q_2 and move the pointer towards right using the transition

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

and when the symbol B is encountered, change the state to q_1, replace B by *a* and move the pointer towards left using the transition

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

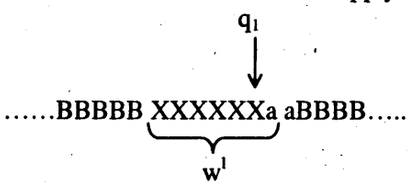
The contents of the tape and position of read/write head after applying these transitions will be



Step 5: Find the rightmost x. Now, to get the rightmost X, as we encounter *a* in the input, remain state q_1, replace *a* by *a* and move the pointer towards left. The transition for this will be

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

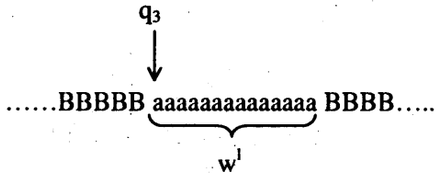
The contents of the tape and position of read/write head after applying these transitions will be



Step 6: Repeating the steps through step 3, there will not be any more X's and the left of the leftmost 1 will be B. Once this B is encountered, change the state to q_3 which is the final state, replace B by B and move the pointer towards right. The transition will be

$$\delta(q_1,B) = (q_3,B,R)$$

The final contents of the tape and position of read/write head is shown below:



So, given the string *w*, the TM to obtain the string *ww* is given by

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

where

$Q = \{q_0, q_1, q_2, q_3\}$
$\Sigma = \{a\}$
$\Gamma = \{a, X, B\}$
q_0 is the start state
B is blank character
$F = \{q_3\}$
δ is shown below using the transition table

	δ	Γ		
		a	X	B
→	q_0	(q_0,X,R)	-	(q_1,B,L)
	q_1	(q_1,a,L)	(q_2,a,R)	(q_3,B,R)
	q_2	(q_2,a,R)	-	(q_1,a,L)
	$*q_3$	-	-	-

It is left to the reader to take an example and show the sequence of moves made by the TM. By looking at the transitional table we can easily write the transition diagram which is also left to the reader as an exercise.

Example 9.13: Construct a TM that stays in the final state q_f whenever $x \geq y$ and non-final state q_n whenever $x < y$ where x and y are positive integers represented in unary notation.

Let q_0 be the start state and assume the two unary integers *x* and *y* are separated by #. Also, assume the string x#y is enclosed between B's as shown below:

$$Bx\#yB$$

The initial ID will be of the form

$$Bq_0x\#yB$$

with the read/write head pointing to the first leftmost digit of x and the final configuration will be either

$$Bq_fx\#yB \text{ whenever } x \geq y$$

or

$$Bq_nx\#yB \text{ whenever } x < y$$

In other words,

$$Bq_0x\#yB \vdash^* Bq_fx\#yB \text{ whenever } x \geq y$$
$$Bq_0x\#yB \vdash^* Bq_nx\#yB \text{ whenever } x < y$$

Note: While designing the TM for the language $L = a^nb^n$ in example 9.2, each leftmost symbol *a* was matched with leftmost symbol *b*. On similar lines we can solve this problem also.

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

1. Replace the left most digit of integer x by X and then move the read/write head till we get the leftmost digit of integer y.
2. Replace it by X and move towards left till the leftmost 1 of integer x is obtained.
3. Repeat through step 1 till one of the condition is satisfied:
   a. No more 1's in integer x and y.
   b. More 1's in integer x which results in no 1's in integer y.
   c. More 1's in integer y which results in no 1's in integer x.

So, final contents of the tape will have one of

1. XXXXX#XXXXXB whenever x = y with output q_f.
2. XXXXX11#XXXXXB whenever x > y with output q_f.
3. XXXXX#XXXXX11B whenever x < y with output q_n.

If the condition shown in step 3.a or 3.b is encountered, change the state to q_f; Otherwise change the state to q_n. Now, consider the situation

XX11#XX11B
↑
q_0

where first two 1's of integer x are replaced by X's and first two 1's of integer y are also replaced by X's. In this situation, the read-write head points to the left most 1 of integer x 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, when the digit 1 is encountered, change the state to q_1, replace 1 by X and move pointer towards right using the transition

$$\delta(q_0,1) = (q_1,X,R)$$

But, in state q_0, if the symbol # is encountered, it means that there are no 1's in integer x and change the state to q_5 using the transition

$$\delta(q_0,\#) = (q_5,\#,R)$$

Step 2: In state q_1, the pointer should move towards right till # is encountered and then change the state to q_2 which can be done using the transitions

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

Step 3: In state q2, TM may encounter X's when searching for leftmost 1 in y. If so, the TM should replace X by X and move the pointer towards right using

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

Once the machine encounters 1, it should change the state to q_3, replace 1 by X and move the pointer towards left using

$$\delta(q_2,1) = (q_3,X,L)$$

But, if the symbol B is encountered, it means that x > y and the machine should enter into final state q_6 using the transition

$$\delta(q_2,B) = (q_6,B,L)$$

and finally from q_6 we can enter into state q_f.

Step 4: In state q_3, every X should be replaced by X and the pointer should be moved towards left using the transition

$$\delta(q_3,X) = (q_3,X,L)$$

But, once # is encountered, move the pointer towards left, change the state to q_4 to search for leftmost 1 in *x* using the transition

$$\delta(q_3,\#) = (q_4,\#,L)$$

Step 5: In state q_4, replace 1 by 1 and move towards left using

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

But, once X is encountered, change the state to q_0, replace X by X and move the pointer towards right using

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

Step 6: Whenever the machine is in q_5, it means that there are no 1's in x. If there are no 1's in y, then y will have only X's followed by B in that case the machine should enter into state q_6. The transitions will be

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

and from state q_6 we can reach the final state q_f. But, in state q_5, if 1's are encountered, it means that x < y and the machine goes to state q_7 from which non-final state q_n is reached using the transition

$$\delta(q_5,1) = (q_7,1,L)$$

Step 7: From state q_6, the machine should enter into final state q_f and the pointer should point first digit of x which can be done using

$$\delta(q_6,1) = (q_6,1,L)$$
$$\delta(q_6,X) = (q_6,X,L)$$
$$\delta(q_6,\#) = (q_6,\#,L)$$
$$\delta(q_6,B) = (q_f,B,R)$$

Step 8: From state q_7, the machine should enter into final state q_n and the pointer should point first digit of x which can be done using

$$\delta(q_7,X) = (q_7,X,L)$$
$$\delta(q_7,\#) = (q_7,\#,L)$$
$$\delta(q_7,B) = (q_n,B,R)$$

So, the final TM is given by

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

where

$Q = \{q_0, q_1, q_2, q_3, q_4, q_5, q_6, q_7, q_n, q_f\}$
$\Sigma = \{1,\#\}$
$\Gamma = \{1, X, \#, B\}$
q_0 is the start state
B is blank character
$F = \{q_f, q_n\}$
δ is shown below using the transition table

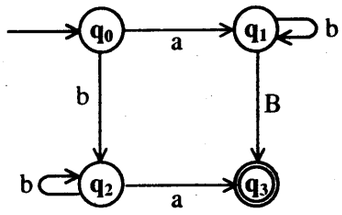
	δ	Γ			
		1	#	X	B
→	q_0	(q_1,X,R)	$(q_5,\#,R)$	-	-
	q_1	$(q_1,1,R)$	$(q_2,\#,R)$		
	q_2	(q_3,X,L)	-	(q_2,X,R)	(q_6,B,L)
	q_3	-	$(q_4,\#,L)$	(q_3,X,L)	-
	q_4	$(q_4,1,L)$	-	(q_0,X,R)	-
	q_5	$(q_7,1,L)$	-	(q_5,X,R)	(q_6,B,L)
	q_6	$(q_6,1,L)$	$(q_6,\#,L)$	(q_6,X,L)	(q_f,B,R)
	q_7	-	$(q_7,\#,L)$	(q_7,X,L)	(q_n,B,R)
	$*q_n$	-	-	-	-
	$*q_f$	-	-	-	-

Example 9.14: What language is accepted by the machine $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ where $Q = \{q_0, q_1, q_2, q_3\}$, $\Sigma = \{a, b\}$, $\Gamma = \{a, b, B\}$, q_0 is start state, B is blank character, $q_f = \{q_3\}$ where δ is defined as follows:

$$\delta(q_0,a) = (q_1,a,R)$$
$$\delta(q_0,b) = (q_2,b,R)$$
$$\delta(q_1,b) = (q_1,b,R)$$
$$\delta(q_1,B) = (q_3,B,R)$$

$\delta(q_2,b) = (q_2,b,R)$
$\delta(q_2,a) = (q_3,a,R)$

Note: It is clear from the TM that the movement of the read/write pointer is only towards right and it can be compared with the FA. So, the equivalent FA for these transitions can be written as



It is clear from the graph that to reach the final state the FA can take only two paths yielding the language containing either bb*a or ab* (Leaving B as it is not the input symbol) which can be defined as

$$L = \{bb^*a + ab^*\}$$

which is the string consisting of at least one *b* followed by one *a* or a single *a* followed by zero or more *b*'s.

9.8 Church Turing Hypothesis (Church's/Church-Turing thesis)

Church's thesis: Various formal models of computations such as *Recursive functions* and *Post systems* were established by three prominent persons A.Church, S.C.Kleene and E.Post.

A function is called *primitive recursive* if and only if it can be constructed from the basic functions by successive composition and primitive recursion.

A *Post system* is similar to unrestricted grammar consisting of an alphabet and some production rules by which successive strings can be derived.

In addition to recursive functions and Post systems, many other formal computations models have been proposed. On examination it was found that though the computational models looked quite different, they expressed the same thing. This observation was formalized in ***Church's thesis*** which is stated as follows:

Any "effective computation" or "any algorithmic" procedure that can be carried out by a human being or a team of human beings or a computer, can be carried out by some Turing machine. In other words, there is an effective procedure to solve a decision problem P if and only if there is a Turing machine that answers *yes* on inputs $w \in P$ and *no* for $w \notin P$.

This theory maintains that all the models of computations those are proposed and yet to be proposed, are equivalent in their power to recognize languages or compute functions. This thesis predicts that it is unable to construct models of computation more powerful than the existing ones.

The above statement is known as "Church's thesis" named after the logician A.Church. Since the *Church's thesis* is closely related to Turing's thesis which states that we can not go beyond Turing machines or their equivalent, it is also called **Church-Turing thesis**.

Since there is no precise definition for "effective computation" or there is no precise definition for "algorithmic procedure", Church's thesis is not a mathematically precise statement. So, this statement is not proved at the same time it has been not been disproved. Even though it is simply stated and not proved, now majority of scientists have accumulated enough evidence over the years that has caused Church's thesis to be generally accepted.

Exercises:

1. Explain the Turing machine model
2. Define Turing machine
3. What is an ID with respect to TM?
4. Define move of a TM
5. What language is accepted by TM?
6. What is a recursively enumerable language?
7. Obtain a Turing machine to accept the language $L = \{0^n1^n \mid n \geq 1\}$
8. Obtain a Turing machine to accept the language $L(M) = \{0^n1^n2^n \mid n \geq 1\}$
9. Obtain a TM to accept the language $L = \{w \mid w \in (0+1)^*\}$ containing the sub string 001
10. Obtain a Turing machine to accept the language containing strings of 0's and 1's ending with 011
11. Obtain a Turing machine to accept the language $L = \{w \mid w \text{ is even and } \Sigma = \{a,b\}\}$
12. Obtain a Turing machine to accept a palindrome consisting of a's and b's of any length.
13. On what basis we say that TM is a transducer?
14. What is Turing computable?
15. Let x and y are two positive integers. Obtain a Turing machine to perform x + y
16. Given a string w, design a TM that generates the string ww where $w \in a^+$
17. Construct a TM that stays in the final state q_f whenever $x \geq y$ and non-final state q_n whenever $x < y$ where x and y are positive integers represented in unary notation
18. What language is accepted by the machine $M = (Q, \Sigma, \delta, q_0, B, F)$ where $Q = \{q_0, q_1, q_2, q_3\}$, $\Sigma = \{a, b\}$, $\Gamma = \{a, b, B\}$, q_0 is start state, B is blank character, $q_f = \{q_3\}$ where δ is defined as follows:

$$\delta(q_0,a) = (q_1,a,R)$$
$$\delta(q_0,b) = (q_2,b,R)$$
$$\delta(q_1,b) = (q_1,b,R)$$
$$\delta(q_1,B) = (q_3,B,R)$$
$$\delta(q_2,b) = (q_2,b,R)$$
$$\delta(q_2,a) = (q_3,a,R)$$

19. How to achieve complex tasks using TM?
20. Let x and y are two positive integers represented using unary notation. Design a TM that computes the function

$$f(x, y) = x + y \quad \text{if } x \geq y$$
$$f(x,y) = xx \quad \text{if } x < y$$

21. What are the various variations of TM?
22. Define the following
    - Turing machine with stay-option
    - Turing machine with multiple tracks
    - Turing machine with semi-infinite tape
    - Off-line Turing machine
    - Multi-tape Turing machine
    - Linear bounded Automaton
23. What is a multi-tape Turing machine? Show how it can be simulated using single tape Turing machine.

Summary

Now!! We know

- Concept of Turing Machine model
- Definition of Turing machine (Standard Turing machine)
- Definition of Instantaneous description w.r.t TM
- Moves of TM
- Languages accepted by TM
- Recursively enumerable language
- Constructing TMs for varieties of languages
- TM as transducer
- Solution to more than 10 problems of various nature.

10

Extensions of Turing Machines

What we will know after reading this chapter?

- Multi-tape Turing machine
- Equivalence of single tape and multi-tape TM's
- Non-deterministic Turing Machine
- Turing machine with stay-option

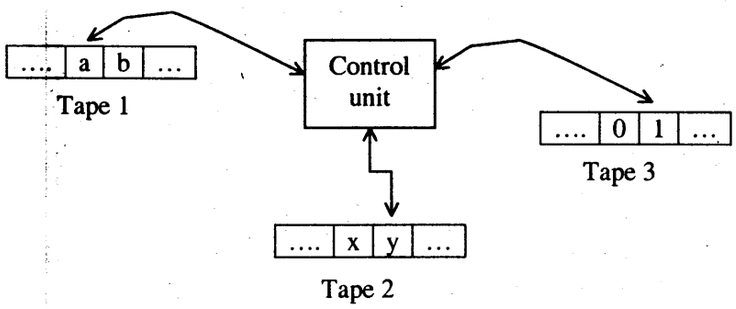
So far in the previous sections, we have discussed the concepts of Standard Turing Machines. Now, we shall concentrate on the variations or extensions of the Standard Turing machine and using simulators we show that the extensions of Turing machines in fact are equivalent to Standard Turing machines. Instead of providing the complete simulation, we shall provide only broad outline to show that the machines are equivalent. We can have so many variations of Standard Turing machines. With minor modification we can have the following Turing machines:

- Multi-tape Turing Machine
- Non-deterministic Turing Machine

This section discusses these two variations of TM. Other variations by imposing certain restrictions (restricted TMs) are discussed in the chapter 12.

10.1 Multi-tape Turing Machines

A multi-tape Turing Machine is nothing but a standard Turing Machine with more number of tapes. Each tape is controlled independently with independent read-write head. The pictorial representation of multi-tape Turing machine is shown in figure below:



The various components of multi-tape Turing Machine are:

a. Finite control
b. Each tape having its own symbols and read/write head.

Each tape is divided into cells which can hold any symbol from the given alphabet. To start with the TM should be in start state q_0. If the read/write head pointing to tape 1 moves towards right, the read/write head pointing to tape 2 and tape 3 may move towards right or left depending on the transition. The formal definition of Multi-tape Turing machine can be defined as follows.

Definition: The Multi-tape Turing Machine is an n-tape 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 from $Q \times \Gamma^n$ to $Q \times \Gamma^n \times \{L,R\}^n$
q_0 is the start state
B is a special symbol indicating blank character
$F \subseteq Q$ is set of final states

The move of the multi-tape TM depends on the current state and the scanned symbol by each of the tape heads. For example, if number of tapes in TM is 3 as shown in the figure and if there is a transition

$$\delta(q, a, b, c) = (p, x, y, z, L, R, S)$$

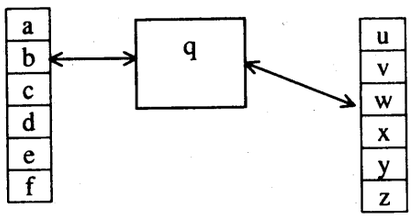
where *q* is the current state. The transition can be interpreted as follows. The TM in state *q* will be moved to state *p* only when the first read/write head points to *a*, the second read-write head points to *b* and third read/write head points to *c* and the read/write head will be moved to left in the first case and right in the second case. But, the read/write head with respect to third tape will not be altered. At the same time, the symbols *a*, *b* and *c* will be replaced by *x*, *y* and *z*. It can be easily shown that the n-tape TM in fact is equivalent to the single tape Standard Turing Machine as shown below.

10.2 Equivalence of single tape and multi-tape TM's

Theorem: Every language accepted by a multi-tape TM is recursively enumerable.

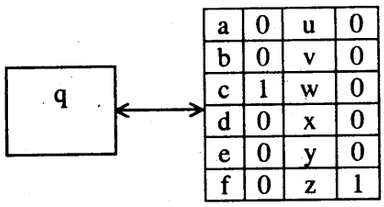
Note: The theorem clearly indicates that every language accepted by a multi-tape TM is also accepted by a standard TM.

Proof: This can be shown by simulation. For example, consider a TM with two tapes as shown below:



Multi-tape

The above 2-tape TM can be simulated using single tape TM which has four tracks as shown in figure below:



Single tape Multiple tracks

The first and third tracks consist of symbols from first and second tape respectively. The second and fourth track consists of the positions of the read/write head with respect to first and second tape respectively. The value 1 indicates the position of the read/write head. It is clear from the above figure that, the machine in state q and when the first read/write head points to the symbol *c*, the second read/write head points to the symbol *z*, then the machine enters into state p, if and only if this transition is defined for TM with multi-tapes. So, whatever transitions have been applied for multi-tape TM, if we apply the same transitions for the new machine that we have constructed, then the two machines are equivalent.

10.3 Nondeterministic Turing Machines

The difference between nondeterministic TM and deterministic TM lies only in the definition of δ. The formal definition of *nondeterministic* TM is shown below:

Definition: The *nondeterministic* 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 from $Q \times \Gamma$ to $2^{Q \times \Gamma \times \{L,R\}}$
q_0 is the start state
B is a special symbol indicating blank character
$F \subseteq Q$ is set of final states.

It is clear from the definition of δ that for each state q and tape symbol X, $\delta(q,X)$ is a set of triples

$$\{(q_1,X_1,D), (q_2,X_2,D), (q_3,X_3,D),\ldots\ldots, (q_i,X_i,D)\}$$

where i is a finite integer and D is the direction with 'L' indicating left or 'R' indicating right. The machine can choose any of the triples as the next move. The language accepted by TM is defined as follows.

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

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

The language is thus a set of all those words w in Σ^* which causes M to move from start state q_0 to the final state p.

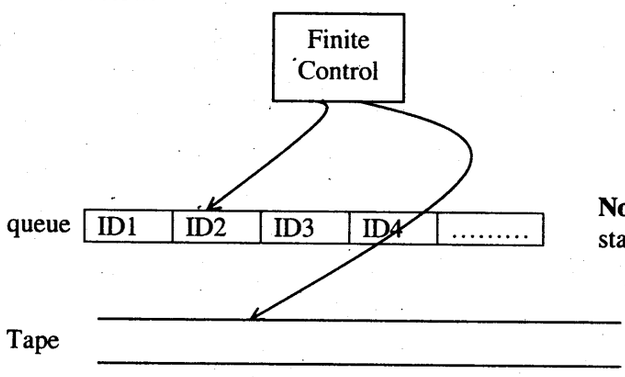
A nondeterministic TM may have many moves that does not lead to a final state. But, we are interested in only those moves that leads to the final states. The nondeterministic TM in fact is no more powerful than the deterministic TM. Any language accepted by nondeterministic TM can be accepted by deterministic and both are equivalent. We can simulate a deterministic TM from a nondeterministic TM as shown below:

Theorem: For every nondeterministic TM (NTM) there exists a deterministic TM (DTM) such that L(NTM) = L(DTM)

Proof : Given a string *w*, NTM starts at the initial configuration(initial ID) and goes through a sequence of configurations(IDs) until it reaches one of the conditions:

- Final state is reached and the machine halts
- The transition is not defined and the machine halts
- Goes into an infinite loop

To go to the next configuration(ID), the NTM has to choose from a finite set of configurations(IDs). All these configurations(IDs) which can be obtained by NTM for a given string *w* can be represented by a tree. The way NTM is simulated by DTM is shown using the figure shown below:



Note: Each ID is associated with a state and the next input symbol