It is clear from the figure that all the IDs of NTM (represented by the nodes of the tree) are placed in the queue one after the other. Each ID inserted into the queue is also associated with the state and the next input symbol to be scanned. The first ID to be explored and examined will be at the front end of the queue. Given any ID in the queue, all IDs to the left of designated ID are assumed to be deleted (marked) and the IDs towards right are assumed to be explored or to be examined. For example, if current ID is ID3 then ID1 and ID2 are already explored and we need not consider them whereas the ID4, ID5 etc., are the nodes to be explored later. The steps carried by DTM are shown below:

1. The current ID is examined (For the first time current ID is ID1 and is marked/deleted) based on the state and the scanned symbol. If the state in the current ID is final state then DTM accepts the string and the machine halts.
2. If the state is non final state, the current ID(configuration) is explored and various IDs(configurations) obtained are inserted at the end of the queue
3. Step 1 and 2 are repeated until no more ID is examined or queue is empty.

The above steps can be clearly explained using the tree representation. The root node corresponds to the initial configuration (initial ID) and it is the only vertex of level 0. All the configurations (IDs) that are obtained by applying the transition function of NTM only once will be the children of the initial configuration (ID). These new vertices which are derived from the root are at level 1. In general, from the configurations (IDs) at level i, the configurations at level $i+1$ can be obtained. Since the configurations are finite, the number of children at various levels is finite.

The easiest way to simulate NTM using DTM is to traverse this tree using BFS (Breadth-First-Search) from the root until the halt state (the string w is accepted or rejected) is reached. At each level of the tree, the DTM applies the transition function corresponding to the NTM to each configuration (ID) at that level and computes its children (new IDs). These new IDs are the configurations of the next level and they are stored on the tape (if necessary a second tape may be used). If there is a halting state among these children, then DTM accepts the string and halts. It can be easily seen that DTM accepts a string if and only if NTM accepts it. Thus any language accepted by a NTM is also accepted by a DTM.

10.4 Turing machine with stay-option

In the Standard Turing Machine, after scanning the symbol and after replacing the symbol on the tape, the read/write head used to move either left or right based on the control mechanism. Apart from having the read/write either towards left or right, if there is one more option where in the read/write head stays in the same position after updating the symbol on the tape (no movement of read/write head either to the left or right), then the Turing machine is called TM with stay-option. Formally, the machine can be defined as follows:

The formal definition of TM with stay-option is provided below:

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 x Γ to Q x Γ x {L,R,S} indicating the TM may move towards left or right or stay in the same position after updating the symbol on the tape.
q_0 is the start state
B is a special symbol indicating blank character
$F \subseteq Q$ is set of final states.

Exercises:

1. Explain the general structure of Multi-tape machines. Show that they are equivalent to standard Turing machines
2. Define non-deterministic Turing machine and show that the language accepted by NTM is also accepted by DTM and they are equivalent
3. Define Turing machine with stay option

Summary

Now we know

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

11

Programming Techniques for Turing Machines

What we will know after reading this chapter?

- Subroutines
- Multiple tracks (Multi-track)
- Solutions for some typical examples

Achieving complicated tasks using TM

In the chapter 9, we have shown how the simple operations such as addition, subtraction, concatenation and comparison can be achieved using TM. These are the some of the basic operations found in all the computers. Using these basic operations, more complex operations can be achieved using the computers. On similar lines, more complex operations can be achieved using TM also using the primitive operations. To achieve this, let us use block diagrams, which contains boxes and arrows representing the action performed by each box and arrows representing flow of control. The functionality of each component in the block diagram is described, but the implementations are hidden or rather assumed to be implemented (in fact we can show how the functionality of each component can be implemented). Let us discuss how more complicated functions are performed by TMs using the primitive operations (which can be considered as subroutines or functions)

11.1 Multiple tracks/ Multi track)

In the Standard Turing Machine, the tape was divided into squares where each square holds only one symbol. It can be extended so that each tape consist of several tracks. This can be done by dividing the tape into number of tracks and each track is divided into a number of squares with each square holding only one symbol. If there are n-tracks and the read/write head points to m^{th} square, then all the symbols under different tracks beneath that read/write head are the symbols to be scanned. So, all the symbols under the read/write head can be considered as set of characters under one square. Such a TM is called Turing machine with multiple tracks.

11.2 Subroutines

We know that a program consists of zero or more functions (subroutines). On similar lines a Turing Machine can be a collection of zero or more Turing Machine subroutines. A Turing

Machine subroutine is a set of states that performs some pre-defined task. The TM subroutine has a start state and a state without any moves. This state which has no moves serves as the return state and passes the control to the state which calls the subroutine.

Example 11.1: Let *x* and *y* are two positive integers represented using unary notation. Design a TM that computes the function

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

The block diagram to add *x* and *y* can be written as shown in figure 11.1

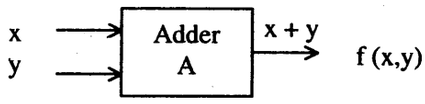


Figure 11.1 Block diagram (subroutine) to add x and y

where *x* and *y* are the inputs to the adder, the output of which will be *x+y* which can be easily implemented (example 9.11). The block diagram to concatenate x with x can be written as shown in figure 11.2.

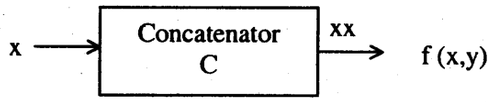


Figure 11.2 Block diagram (subroutine) to concatenate x with x

For this concatenator, x is the only input. The output will be the integer *xx* which can be implemented (example 9.12). It is clear from the given function that the addition should be performed if $x \geq y$ and concatenation of *x* with *x* has to performed whenever $x < y$. The comparator also can be implemented as shown in example 9.13 the block diagram for which can be written as shown in figure 11.3.

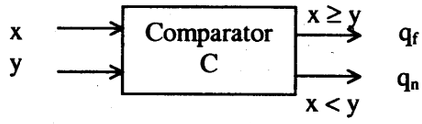


Figure 11.3 Block diagram (subroutine) to compute x+y or xx

Whenever $x \geq y$, the comparator enters into the state q_f which acts as a trigger to invoke the Adder A to add *x* and *y*. When $x < y$, the comparator enters into the state q_n which can act as trigger to the concatanator which concatenates *x* with *x*. The complete high-level block diagram to compute the function f(x,y) using Turing machine is shown in figure 11.4.

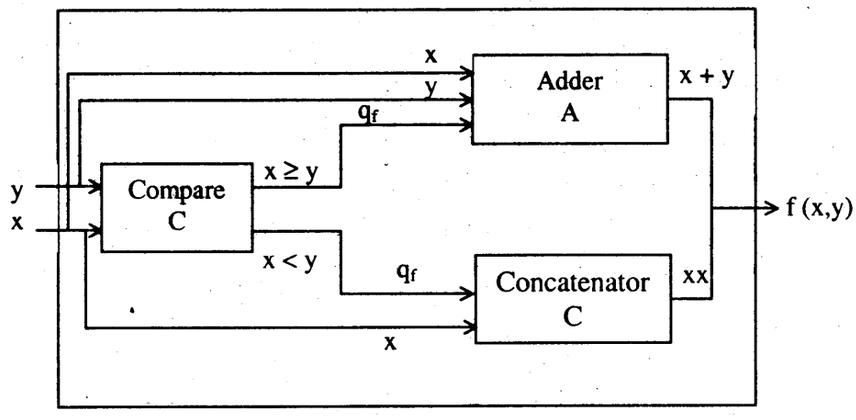


Figure 11.4 Block diagram (subroutine) to compare x and y

Note: The TM can also be represented as the combination of blocks which each block representing a TM which can do some primitive operation (like functions in C). All these TMs that can do some primitive operations can be invoked by another TM, which can be considered as the main program.

Example 11.2: Obtain a TM to multiply two unary numbers separated by the delimiter 1.

Note: Let us assume we have two unary numbers x and y such that x has m number of 0's and y has n number of 0's separated by the delimiter 1 as shown below:

$$0^m10^n$$

The product of x and y should be stored on the tape and the original numbers should not be destroyed. This can be visualized as shown below:

$$0^m10^n10^{mn}$$

To start with let us assume $x = 00$ and $y = 0000$ and are separated by delimiter 1. Assume y ends with 1 which can act as the delimiter for the input string which in turn is followed by infinite number of blanks (B's) as shown below:

$$\overbrace{00}^{x}\,1\,\overbrace{0000}^{y}\,1\,BBBBBB\ldots.$$

The output should be of the form

$$\overbrace{00}^{x}\,1\,\overbrace{0000}^{y}\,1\,\overbrace{00000000}^{xy}\,BBBB\ldots.$$

NOTE: It is clearly visible from the above figure that the unary number y is copied two times (equal to the number of 0's in the number x) in the result xy (which is 0 0 0 0 0 0 0 0). So, to start with we think of how to copy the unary number y once so that we can call this TM (which is represented as a subroutine) repeatedly m (the number of 0's in x) times to achieve the result.

General Procedure:(Algorithm)

1. To start with let us assume a tape will have a string of the form $0^m10^n1BBBBBBBB\ldots.$ which is divided into three groups as shown below:

   First group: 0^m 1 Second group: 0^n 1 last group: BBBBBBBB....

   where $x = 0^m$ and $y = 0^n$ and are separated by the delimiter 1. So, if the input is 0^m10^n then this string is followed by 1 (which acts as delimiter between the input and the result. Note that all B's will be replaced by the result 0^{mn} later). Now, change a 0 in the first group (x) to B.

2. Copy n number 0's from the second group to the last group by replacing n number of B's by n number of 0's.

3. It is clear from second step that "we copy a group of n 0's (replace n B's) to the last group whenever a 0 is the first group is changed to B". When all 0's in the first group are changed to B's there will definitely be mn number of 0's in the last group (which is the product of first and second group i.e., product of x and y).

4. Once first three steps are completed, the contents of the tape will be of the form

   First group: B^m 1 Second group: 0^n 1 last group: 0^{mn}BBBB.....

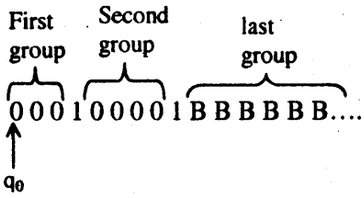
   So, by replacing the string 10^n1 which is enclosed between the first group and the last group by B's we will have only 0^{mn} number of 0's on the tape with ID q_i0^{mn} which is the result.

Now, we shall implement all these steps one after the other in detail.

Step 1 (details) : To implement step 1 in detail, let us take a string of the form:

000100001BBBBBB....

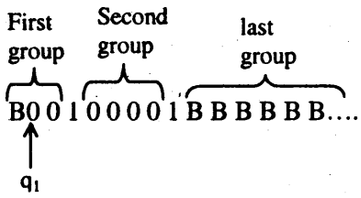
where the first group, second group and last group are identified as shown below:



Assume q_0 is the start state. As per the algorithm in step 1, replace 0 by B, change the state to q_1 and move the head towards right and the corresponding transition for this will be of the form:

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

After applying the above transition, the situation will be of the form:



Now, we should copy *n* 0's from the second group to the last group. This can be achieved by moving the pointer head towards right till we encounter 1 repeatedly using the transition

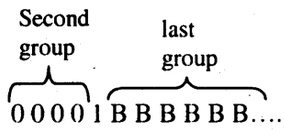
$$\delta(q_1, 0) = (q_1, 0, R)$$

On encountering 1, change the state to q2 and the corresponding transition is:

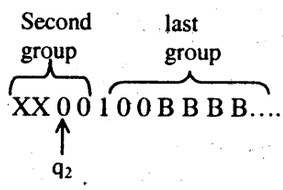
$$\delta(q_1, 1) = (q_2, 1, R)$$

Now, the pointer points to first zero in the second group and we shall copy n 0's from the second group to the last group. The details are shown in step2.

Step 2 (details): (Copy function) This second step of the algorithm (procedure) can be implemented using a subroutine called *copy*. Now, let us think of how to replace *n* number of B's in the last group to *n* number of 0's from the second group. For this reason, let us take only the second and last group as shown below:



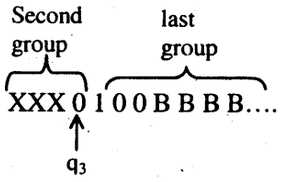
We have to copy n number of 0's from the second group to the last group. This can be achieved by replacing a B by 0 in the last group immediately after changing a 0 to X in the second group. To obtain the required transitions let us assume the situation shown below:



In this situation, two leftmost 0's in second group are replaced by X and the corresponding two leftmost B's are replaced by two 0's in the last group. Let us assume the machine is in state q_2 and the next symbol to be scanned is the symbol pointed to by q_2. It is clear from the above figure that in state q_2 on input 0, change the state to q_3, replace 0 by X and move the pointer towards right using the transition

$$\delta(q_2, 0) = (q_3, X, R)$$

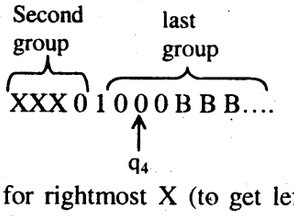
After applying the above transition, the situation is shown below:



Now, in state q_3 on any number of 0's or 1's we can stay in q_3 and simply move the head towards right. But, on encountering a B, change the state to q_4, replace B by 0 and move the head towards left (to replace leftmost zero by X in the second group) using the following transitions:

$$\delta(q_3, 0) = (q_3, 0, R)$$
$$\delta(q_3, 1) = (q_3, 1, R)$$
$$\delta(q_3, B) = (q_4, 0, L)$$

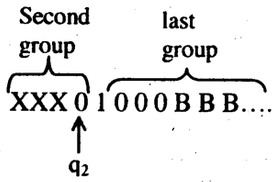
After these transitions, the situation will be of the form shown below:



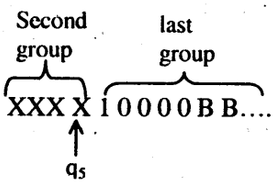
In state q_4, we should search for rightmost X (to get leftmost 0). So, keep updating the head towards left on encountering 0's or 1's and on encountering an X, change the state to q_2, replacing X by X and move the pointer towards right. The corresponding transitions are:

$$\delta(q_4, 0) = (q_4, 0, L)$$
$$\delta(q_4, 1) = (q_4, 1, L)$$
$$\delta(q_4, X) = (q_2, X, R)$$

After applying the above transitions, the situation is shown below:



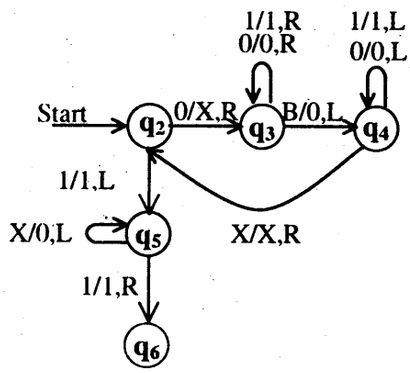
It is clear from this situation shown in above figure that, if there are only X's in second group (means no 0's left), there will be a transition from state q_2 on 1 which implies that n number of B's in the last group are replaced by *n* number of 0's. So, q_2 on 1 we move towards left and change the state to q_5 as shown in the situation below using the transition $\delta(q_2, 1) = (q_5, 1, L)$:



In state q_5 we should see that all X's are replaced by 0's. So, when we move towards left, we may encounter a delimiter 1. In that case, we simply move the head towards right by changing the state to q_6. The corresponding transitions are:

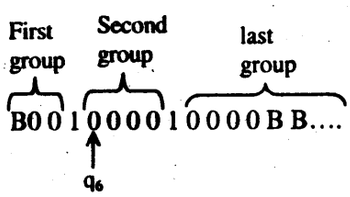
$$\delta(q_5, X) = (q_5, 0, L)$$
$$\delta(q_5, 1) = (q_6, 1, R)$$

Now, step 2 of the general procedure is completed when a leftmost 0 in first group is replaced by B and the corresponding transition diagram is shown below:



Using the above transition diagram, we can have the actual C function to implement copy function and we call this function as the "Copy function".

Step 3 (details) : Once step 2 is completed the current contents of tape will be of the form shown below:



It is clear from this figure that, when leftmost 0 in first group is replaced by a B, *n* 0's from the second group have been copied into last group. Now, we should replace the next leftmost 0 in first group to B and repeat the process again. So, q_6 on 0, change the state to q_7 and move the head towards left using the transition

$$\delta(q_6, 0) = (q_7, 0, L)$$

In state q_7, on 1 change the state to q_8 and move the head towards left using the transition

$$\delta(q_7, 1) = (q_8, 1, L)$$

In state q_8 on input zero move the head towards left using the transition

$$\delta(q_8, 0) = (q_9, 0, L)$$

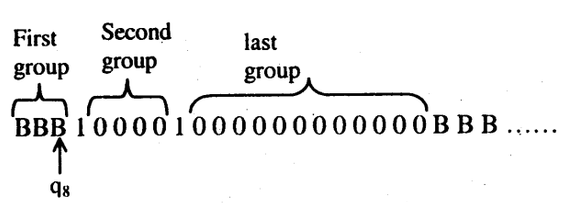
In state q_9 on any number of 0's move the head towards left using the transition

$$\delta(q_9, 0) = (q_9, 0, L)$$

But, if we encounter a B, change the state to q_0 and move the pointer towards right using the transition

$$\delta(q_9, B) = (q_0, B, R)$$

The purpose of the states q_7, q_8 and q_9 is to take control after copying a block of n 0's from the second group to the last group so as to obtain the leftmost zero in first group. But, in state q_8 on encountering B, it means that *n* 0's have been copied from the second group to the last group m number of times and the situation is shown below:



Now, let us replace the delimiter 1 which precede and follow the second group including the second group by B's. This is possible using the transitions shown below:

$\delta(q_8, B) = (q_{10}, B, R)$
$\delta(q_{10}, 1) = (q_{11}, B, R)$
$\delta(q_{11}, 0) = (q_{11}, B, R)$
$\delta(q_{11}, 1) = (q_{12}, B, R)$

So, the Turing machine to accept the given language is given by

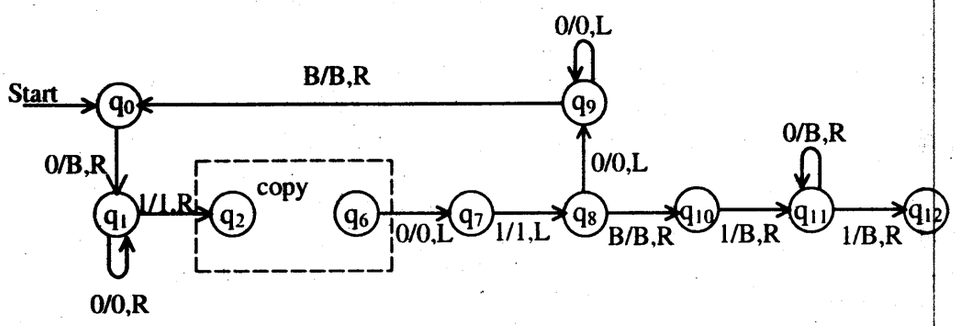
$$M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$$

where

$Q = \{q_0, q_1, q_2, q_3, q_4, q_5, q_6, q_7, q_8, q_9, q_{10}, q_{11}, q_{12}\}$
$\Sigma = \{0, 1\}$
$\Gamma = \{0, 1, X, B\}$
$q_0 \in Q$ is the start state of machine.
$B \in \Gamma$ is the blank symbol.
$F = \phi$
δ is shown below.

$\delta(q_0, 0) = (q_1, B, R)$
$\delta(q_1, 0) = (q_1, 0, R)$
$\delta(q_1, 1) = (q_2, 1, R)$
$\delta(q_2, 0) = (q_3, X, R)$
$\delta(q_3, 0) = (q_3, 0, R)$
$\delta(q_3, 1) = (q_3, 1, R)$
$\delta(q_3, B) = (q_4, 0, L)$
$\delta(q_4, 0) = (q_4, 0, L)$
$\delta(q_4, 1) = (q_4, 1, L)$
$\delta(q_4, X) = (q_2, X, R)$
$\delta(q_5, X) = (q_5, 0, L)$
$\delta(q_5, 1) = (q_6, 1, R)$
$\delta(q_6, 0) = (q_7, 0, L)$
$\delta(q_7, 1) = (q_8, 1, L)$
$\delta(q_8, 0) = (q_9, 0, L)$
$\delta(q_9, 0) = (q_9, 0, L)$
$\delta(q_9, B) = (q_0, B, R)$
$\delta(q_8, B) = (q_{10}, B, R)$
$\delta(q_{10}, 1) = (q_{11}, B, R)$
$\delta(q_{11}, 0) = (q_{11}, B, R)$
$\delta(q_{11}, 1) = (q_{12}, B, R)$

The corresponding transition diagram is shown below:



Exercises:

1. Design a TM that computes the function
   $f(x, y) = x + y$ if $x \geq y$
   $f(x,y) = xx$ if $x < y$

2. Obtain a TM to multiply two unary numbers separated by the delimiter 1.

Now, we know

- Subroutines
- Multiple tracks (Multi-track)
- Solutions for some typical examples

12

Restricted Turing Machines

What we will know after reading this chapter?

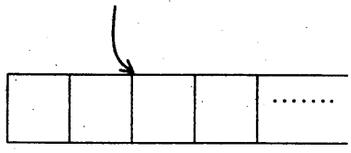
- Turing machine with semi-infinite tape
- Multi-stack Machines
- Counter Machines
- Off-line Turing machine
- Linear bounded automata

So far in the previous sections, we have discussed various concepts of Standard Turing Machines and other variations of TM. Now, we shall concentrate on the other variations of TM by imposing certain restriction on TM. 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:

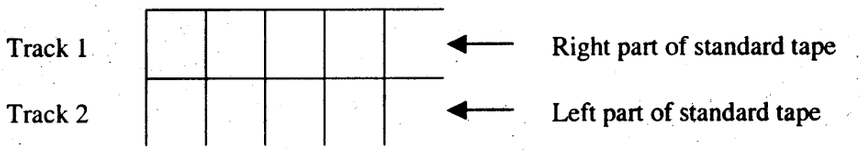
- Turing machine with semi-infinite tape
- Multi-stack Machines
- Counter Machines
- Off-line Turing machine
- Linear bounded automata

12.1 Turing machine with semi-infinite tape

In the Standard Turing Machine, there was no boundary specified for the left side as well as right side of the tape. The read/write head can be moved infinitely towards left as well as towards right i.e., the tape was unbounded in both the directions. Now, if we restrict the read/write head to move only in one direction say towards right (i.e., bounded on the left and unbounded on the right), then the Turing machine is called TM with semi-infinite tape and it can be visualized as shown in figure below:



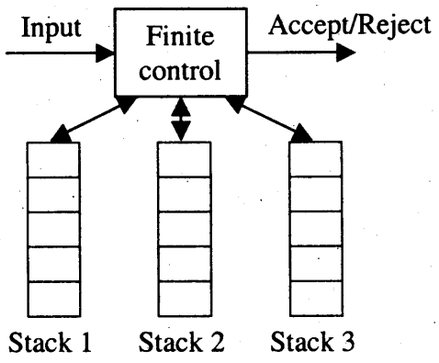
It is clear from the above figure that there are no cells to the left hand side of the initial head position and this restriction does not affect the power of the machine. The TM M can simulate the semi-infinite TM M_S using two tracks for a tape as shown below:



The upper part represents track1 and in these cells let us store the information which lies to the right of some reference point. Initially, the reference point could be the position of the read-write head. The lower part i.e., track 2 contains the left part of the reference point in reverse order. The machine M which is simulating M_S will use the information of track 1 as long as the read-write head of the M_S is towards right of the reference point. When M_S moves towards left of the reference point, the machine M uses cells of track2 from right to left. We can partition the states of M into Q_1 and Q_2 so that the states of Q_1 are used when working with respect to track 1 and the states of Q_2 are used when working with respect to track 2. Some special markers such as #s can be placed at the left hand side of track 1 and track 2 to enable the programmer to switch between the tracks. Following in this direction, we can show that M and M_S are equivalent.

12.2 Multi-stack machines

All the languages which are accepted by PDA are accepted by TM. At the same time, any language which is not accepted by PDA is also accepted by TM. TM is a generalized machine which can accept all languages. The multi-stack machine is generalized model of PDA. A machine with three stacks is shown below:



The pictorial representation of 3-stack machine will have 3 stacks. In general, a k-stack machine is a deterministic PDA with k stacks. Similar to the PDA, it accepts input chosen from the alphabets Σ and has a finite control. All the k stacks will have the alphabets chosen from stack alphabet Γ. The move of the multi-stack depends on:

1. The current state of the finite control
2. The input symbol chosen from Σ
3. The symbol on top of each stack

Once the transition is applied for the current state, input symbol and the current top of each stack, the multistack machine may:

1. Change the state (or it may remain the same state)
2. It may replace the top of the stack

Thus, the general transition for a k-stack machine may take the form:

$\delta(q, a, X_1, X_2, \ldots..X_k) = (p, \alpha_1, \alpha_1, \alpha_1, \ldots. \alpha_k)$ where X_i is on top of the stack for $1 \leq i \leq k$. It is clear from this transition that each symbol on top of the stack can be replaced by different symbols. It can be shown that any language accepted by Turing machine, is also accepted by two-stack machine.

12.3 Counter machines

The counter machine is a restricted multi-stack machine which can be interpreted in several ways: The counter machine is similar to that of multi-stack machine with some modifications. Each stack in the machine is replaced by a counter. Each counter holds a non-negative number. The move of a machine depends on the current state, current input symbol and if one of the value of the counter is zero. The machine can do the following activities in one move:

a. It can change the state
b. It can add or subtract 1 from the counters. But, if the counter is zero, subtracting of 1 from that counter is not possible

Some observations of the languages accepted by counter machines are shown below:

1. The languages accepted by counter machines are recursively enumerable i.e., we can obtain an equivalent TM to accept those languages which are accepted by counter machines.
2. A language accepted by only one counter machine is context free language.

12.4 Off-line Turing machine

In the Standard Turing machine we have assumed that the tape has both the input and output. To start with contents of the tape are assumed to be input symbols and once the TM reaches the final state, the contents of the tape are considered as the output. Now, if we have a separate input buffer as we had in case of finite automaton, we get Off-line Turing machine. But, only difference is that the move of TM is now depends on the state the machine currently in, the current symbol read from the input buffer and the symbol which is currently pointed by read/write head. The formal definition of Off-line Turing machine is shown below:

Definition: The Turing Machine M = (Q, Σ, Γ, δ, q0, 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 x Σ x Γ to Q x Γ x {L,R}
q_0 is the start state
B is a special symbol indicating blank character
$F \subseteq Q$ is set of final states.

12.5 Linear bounded Automata (LBA)

In the previous sections, we have seen that the power of Turing Machine can not be extended beyond the power of Standard Turing Machine by complicating the TM with multiple tapes or by using multi-dimensional tapes. Instead, the power of TM can be restricted by restricting the tape usage. An example of this is the Push Down Automaton which can be considered as a non-deterministic Turing machine. In PDA, it can be assumed that the tape is used like a stack. We can also assume finite portion of the tape as input that leads to finite automaton. With slight alteration in usage of the tape, let us restrict the workspace on the tape using two delimiters '[' and ']'. The given string has to be enclosed between these two limiters. So, longer the string, longer the workspace. This leads to another class of machine called Linear Bounded Automaton (LBA). So, linear bounded automaton is a TM which is bounded based on the length of the input string. Thus, using TM and by restricting the usage of tape, we can obtain LBA, we construct PDA and finite automaton. It is clear from all these types of machines that TM is superset of all these machines. The formal definition of LBA is provided below:

Definition: The Linear Bounded Automaton is a TM

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

where

Q is set of finite states
Σ is set of input alphabets which also has two special symbols '[' and ']'
Γ is set of tape symbols
δ is transition function from Q x Γ to Q x $2^{Q X \Gamma X \{L,R\}}$ with two more transitions of the form $\delta(q_i,[) = (q_j,[,R)$ and $\delta(q_i,]) = (q_j,],L)$ forcing the read/write head to be within the boundaries '[' and ']'
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 that the read/write head cannot go out of the boundaries specified as '[' and ']'. Now, the string can be accepted by LBA only if there is a sequence of moves such that

$$q_0[w] \vdash^* [xq_fz]$$

for some $q_f \in F$ and $x, z \in \Gamma^*$.

Undecidability: A machine may accept a language or it may not accept(reject) a language. So, the output of the machine may be *accept* or *reject.* This problem is called *membership problem.* Formally, the membership problem can be stated as "Given a machine M and a string x, does M accept x?" The output will be *yes/no.* Given a language, the machine may have to identify whether the language is *finite* or *infinite.* All such problems with two answers *yes/no, accept/reject, finite/infinite* are called ***decision problems***. For majority of the decision problems, we can design decision algorithms.

According to Church-Turing thesis, an appropriate way to formulate the idea of a decision algorithm precisely is to use a Turing machine. For some decision problems we can write the algorithms for any given specific instance (which indicates that the problem can be solved using TM), but it is not possible to write a general decision algorithm that works for any given instance, which indicates that a Turing machine does not exist to solve a decision problem. Thus, there are problems that are solvable by TM and that are not solvable by TM. With respect to this, we can divide problems into two groups:

1. The problems for which the solution exists in the form of algorithms i.e., If there is an algorithm to solve a problem, there exists a Turing machine that halts whether the input is accepted or rejected.
2. The problems that run forever i.e., a Turing machine will not halt on inputs that they do not accept

Now, we shall see "What are solvable/decidable problems and what are not solvable (unsolvable/undecidable) problems?"

12.6 A Language that is not Recursively Enumerable

Definition: The decision problems that have decision algorithms the out of which *yes/no* are called solvable problems. According to Church-Turing thesis, an appropriate way to formulate precisely the idea of a decision algorithm is to use a Turing machine. The language L accepted by a Turing machine M is *recursively enumerable language* if and only if L = L(M). Any instance of a problem for which the Turing machine halts whether the input is accepted or rejected is called ***solvable*** or ***decidable problem***. There are so many problems that are solvable. But, there are some problems that are *not solvable.* Now, we shall see what are called *unsolvable/undecidable problems.*

Definition: The problems that run forever on a Turing machine are not *solvable.* In other words, there are some problem input instances for which Turing machines will not halt on inputs that they do not accept. Those problems are called ***unsolvable*** or ***undecidable problems***. In general, if there is no general algorithm capable of solving every instance of the problem, then the decision problem is *unsolvable.* More precisely, if there is no Turing machine recognizing the language of all strings for various instances of the problems input for which the answer is *yes* or *no*, then the decision problem is *unsolvable.*

Let us assume M is a Turing machine with input alphabets {0, 1}, w is a string of 0's and 1's and M accepts w. If this problem with inputs restricted to binary alphabets {0, 1} is undecidable, then the general problem is undecidable and can not be solved with a Turing machine with any alphabet.

Note: The term *unsolvable* and *undecidable* are used interchangeably.

Note: Even though we are able to answer the question in many specific instances, a problem may be undecidable. It means that there is no single algorithm guaranteed to provide an answer for every case.

Note: If a language L is not accepted by a Turing machine, then the language is not recursively enumerable. One important problem which is not recursively enumerable that is unsolvable/undecidable decision problem is "**Halting problem**".

12.7 Halting Problem

The "***Halting Problem***" can informally be stated as "Given a Turing machine M and an input string *w* with the initial configuration q_0, after some (or all) computations do the machine M halts?" In other words we have to identify whether (M, w) where M is the Turing machine, halts or does not halt when *w* is applied as the input. The domain of this problem is to be taken as the set of all Turing machines and all *w* i.e., Given the description of an arbitrary Turing machine M and the input string *w*, we are looking for a single Turing machine that will predict whether or not the computation of M applied to *w* will halt.

When we state decidability or undecidability results, we must always know what the domain is, because this may affect the conclusion. The problems may be decidable on some domain but not on another.

It is not possible to find the answer for Halting problem by simulating the action of M on *w* by a universal Turing machine, because there is no limit on the length of the computation. If M enters into an infinite loop, then no matter how long we wait, we can never be sure that M is in fact in a loop. The machine may be in a loop because of a very long computation. What is required is an algorithm that can determine the correct answer for any M and *w* by performing some analysis on the machine's description and the input.

Formally, the Halting Problem is stated as "Given an arbitrary Turing machine $M = (Q, \Sigma, \Gamma, \delta, q_0, B, A)$ and the input $w \in \Sigma^*$, does M halt on input *w*?"

Post's Correspondence Problem

The Post correspondence problem can be stated as follows. Given two sequences of n strings on some alphabet Σ say

$$A = w_1, w_2, \ldots, w_n$$

and

$$B = v_1, v_2, \ldots, v_n$$

we say that there exists a Post correspondence solution for pair (A,B) if there is a nonempty sequence of integers i,j,.....,k, such that

$$w_i w_j \ldots w_k = v_i v_j, \ldots v_k.$$

The Post correspondence problem is to devise an algorithm that will tell us, for any (A,B) whether or not there exists a PC-solution.

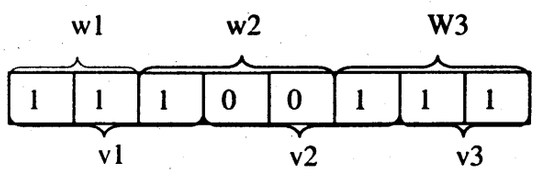
For example, Let $\Sigma = \{0.1\}$. Let A is w1, w2, w3 as shown below:

$w_1 = 11, w_2 = 100, w_3 = 111$

Let B is v1, v2, v3 as shown below:

$v_1 = 111, v_2 = 001, v_3 = 11$

For this case, there exists a PC-solution as shown below:



If we take

$w_1 = 00, w_2 = 001, w_3 = 1000$

$v_1 = 0, v_2 = 11, v_3 = 011$

there cannot be any PC-solution simply because any string composed of elements of A will be longer than the corresponding string from B.

Definition of Multi-tape TM: 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 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 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.

Exercises:

Write short notes on the following variations of TM

- 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

Summary

Now!! We know

- Turing machine with semi-infinite tape
- Multi-stack Machines
- Counter Machines
- Off-line Turing machine
- Linear bounded automata