7.5 UNIVERSAL TURING MACHINE

In the examples given, we have designed a separate Turing machine for each language to be recognized. The concept of universal Turing machine (UTM) was to design a generic machine, which takes the code of an arbitrary Turing machine M and an arbitrary string w and checks if $w \in L(M)$. Thus, the input for the UTM is an ordered pair (M, w) and the output is *yes* if

Scanned by CamScanner

$w \in L(M)$ and *no* if $w \notin L(M)$. Now, after having defined the concept of UTM, the next thing is to design the code for the Turing machine.

The code for the Turing machine can be designed in many ways. The total set of transition functions δ provides the complete description of the Turing machine. Thus, the coding of the Turing machine involves the coding of all its transition functions. A transition function in a Turing machine is of the type: $\delta(q_i, x) \vdash (q_j, y, \mathrm{R})$, where the components have the usual meaning. A transition function has five elements: current state, current symbol, new state, new symbol, and the direction of motion. To code the transition function we have to encode all its components. There are many alternative choices for this coding, but we are choosing binary coding.

Let $M = (Q, \Sigma, \Gamma, \delta, q_0, \beta, F)$ be the Turing machine to be encoded where

$Q = \{q_0, q_1, q_2, q_f\}$,
$\Sigma = \{a, b\}$,
$\Gamma = \{a, b, x, y, \beta\}$,
$\delta = \{\delta(q_0, a) \vdash (q_1, b, \mathrm{R}), \delta(q_0, x) \vdash (q_1, y, \mathrm{L}), \delta(q_1, x) \vdash (q_2, y, \mathrm{R}),$
$\delta(q_2, y) \vdash (q_f, \beta, \mathrm{R}), \delta(q_f, x) \vdash (q_0, y, \mathrm{L})\}$, and
$F = \{q_f\}$.

Since we are using binary coding, the only symbols available for coding are 0 and 1. We use the 0s to encode the elements of transition function and 1 as the separator between the two elements of a transition function.

The states can be coded as follows:

$q_0 = 0$	$q_2 = 000$
$q_1 = 00$	$q_f = 0000$

The tape symbols can be coded as follows:

$a = 0$	$y = 0000$
$b = 00$	$\beta = 00000$
$x = 000$	

The direction of motion can be coded as follows:

L = 0
R = 00

Now, the transition function can be coded as follows:

$\delta(q_0, a) \vdash (q_1, b, \mathrm{R}) \Rightarrow 010100100100$

$\delta(q_1, x) \vdash (q_2, y, \mathrm{R}) \Rightarrow 001000100010000100$

$\delta(q_2, y) \vdash (q_f, \beta, \mathrm{R}) \Rightarrow 000100001000010000100$

$\delta(q_f, x) \vdash (q_0, y, \mathrm{L}) \Rightarrow 00001000101000010$

$\delta(q_f, x) \vdash (q_0, y, \mathrm{L}) \Rightarrow 00001000101000010$

In the coding of the Turing machine, the coding ends with 111. The transition functions are separated by 11. Thus, the total coding of the machine is 010100100100110010001000100001 00110001000010000100000100110000100010100000101100001000101000010111.

Scanned by CamScanner

Let $w = abbb$ be the string to be checked on the Turing machine M. The input to the UTM will be

11101010010010011001000100010000100110001000010000100000100110000100010100001011000010001010000101111*abbb*.

The UTM uses the binary code of the Turing machine M on the string *abbb* and will check if $abbb \in L(M)$. If true, then the UTM halts to say *yes* and if false it stops to say *no*. The UTM has to be a generic machine that should be capable of implementing the code of any arbitrary Turing machine. The UTM is a concept of a generic machine and there can be different ways to implement a UTM but the main thing is that its concept led to the modern day computer.

UTM and modern day computer The concept of UTM is similar to the behaviour of the modern day computer. The modern day computer can choose a program (a solution algorithm) for any desired problem and solve it on the required data. Thus a computer, today, is a generic machine that can choose the program related to any domain of real life such as mathematics, engineering, medicine, space research, games, dress designing, etc., and execute it to give the required solution.

7.6 ENHANCEMENTS IN TURING MACHINE

The model of the Turing machine presented in the previous sections is not only a possible one, but many enhancements have also been proposed to improve it. These enhancements are in terms of the number of tapes, number of tracks on the tape, permissible moves, attachment of memory, etc. The purpose of these enhancements is to make the Turing machine more efficient and easier to use. Let us discuss some such enhancements.

7.6.1 Multitrack Turing Machine

The model of the Turing machine discussed so far had only one track. A multitrack Turing machine consists of multiple tracks on a single tape. The additional tracks can be used for the purpose of the memory. Figure 7.11 shows a multitrack Turing machine.

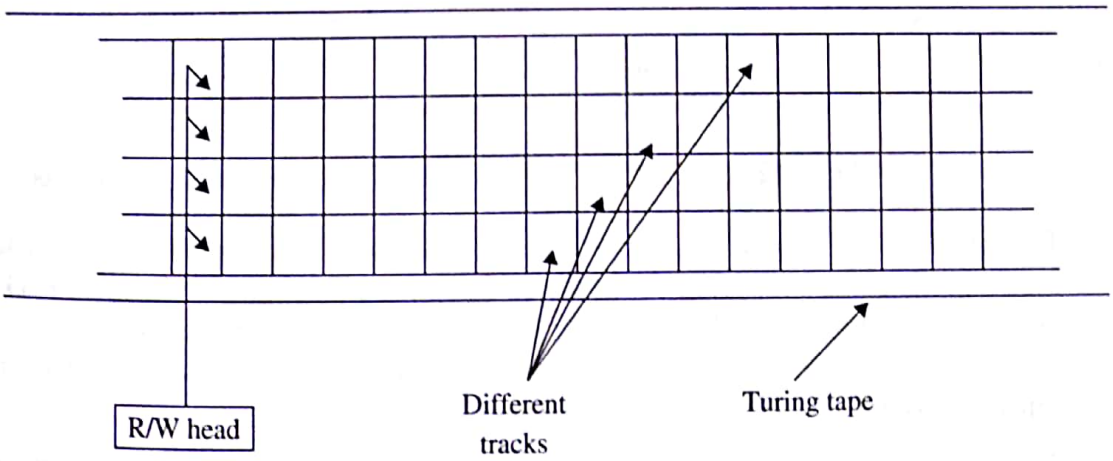

Fig. 7.11 Multitrack Turing machine

Scanned by CamScanner

In a multitrack Turing machine, all the tracks reside on the same Turing tape and move in same direction simultaneously, i.e., the direction of motion of the Turing tape. The r/w head is capable of reading and writing from all the cells simultaneously which lie in the same column. The bottom-most track is used as the data track where input data or strings are stored. Usually, the result is also stored on this track. The upper tracks are used as auxiliary memory to include the appropriate marks. The transition function of the multitrack (say n track) Turing machine looks as follows:

$$\delta([q_i, k_1, k_2, k_3, \ldots, k_{n-1}], [a_1, a_2, a_3, \ldots, a_n]) \vdash ([q_j, l_1, l_2, l_3, \ldots, l_{n-1}], [b_1, b_2, b_3, \ldots, b_n], \text{L/R})$$

Here the r/w head is currently in the state q_i, remembers the symbols $[k_1, k_2, k_3, \ldots, k_{n-1}]$ reads the symbol a_1 from the uppermost track, symbol a_2 from the next uppermost track, and so on. After reading it changes its state to q_j, remembers the symbols $[l_1, l_2, l_3, \ldots, l_{n-1}]$ and replaces the symbol a_1 by b_1; a_2 by b_2; ...; a_n by b_n; and moves towards the left/right.

To illustrate the working of the multitrack Turing machine, we take a two-track Turing machine and solve a few problems on it. A two-track Turing machine M is a seven-tuple structure as follows:

$$M = (Q, \Sigma, \Gamma, \delta, [q_0, \beta], [\beta, \beta], [q_f, \beta])$$

where the symbols Q, Σ, Γ, δ have the usual meaning. The Turing machine is initially in the state q_0 and remembers nothing as indicated by the symbol β in $[q_0, \beta]$. $[\beta, \beta]$ is the counterpart of the blank symbol β on a single-track machine and indicates that all the columns on the Turing tape are filled with a set of blank symbols before it is given the input. $[q_f, \beta]$ indicates that when the Turing machine halts in the final state q_f it remembers nothing.

Example 7.17 Design a two-track Turing machine to multiply two positive integers m and n.

Solution The initial condition of the Turing tape is shown in Fig. 7.12.

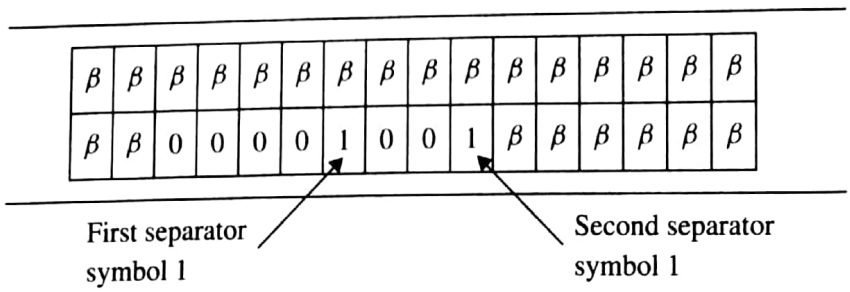

Fig. 7.12 Initial condition of a two-track Turing machine for multiplication

Two numbers $m = 4$ and $n = 2$ are represented by the corresponding number of 0s and are followed by 1. These numbers are lying on the lower track of the Turing tape. The upper track of the Turing tape contains all blank symbols initially. The Turing machine works as follows:

The basic process of multiplication involves copying the n number of 0s m times after the second separator symbol '1'. This is done as follows:

1. The r/w head reads the first 0 of number m and puts a check mark (√) above it. It now passes over the rest of the 0s of the number m and the first separator symbol '1'. It inserts an asterisk mark, *,

Scanned by CamScanner

above the first 0 of number n. It travels towards the right and crosses the second separator symbol '1' and inserts a 0 after it. The process is shown in Fig. 7.13.

β	β	√	β	β	β	β	*	β	β	β	β	β	β	β	β
β	β	0	0	0	0	1	0	0	1	0	β	β	β	β	β

Fig. 7.13 Copying of the first 0 of number n

2. The r/w head now comes back and inserts an asterisk mark, *, over the second 0 of number n. It travels towards the right and inserts another 0 after the second separator symbol '1'.
3. The process continues till all the 0s of number n are copied. The asterisk marks over the 0s of number n help us in keeping track of the number of 0s copied so far. This completes the first cycle of copying. The condition of the Turing tape after completion of the first cycle is shown in Fig. 7.14.

β	β	√	β	β	β	β	*	*	β	β	β	β	β	β	β
β	β	0	0	0	0	1	0	0	1	0	0	β	β	β	β

Fig. 7.14 Copying of the second 0 of number n and completion of the first cycle

4. To start the next cycle, the r/w head comes back and travels towards the left. In the path, it removes all the asterisk marks above the 0s of number n as shown in Fig. 7.15.

β	β	√	β	β	β	β	β	β	β	β	β	β	β	β	β
β	β	0	0	0	0	1	0	0	1	0	0	β	β	β	β

Fig. 7.15 After removal of * from above 0s of n

5. It reaches number m and inserts the check mark (√) over the next 0. Corresponding to this 0, it copies all the 0s of number n after the second separator symbol '1'. This completes the next cycle. The condition of the Turing tape after completion of the second cycle is shown in Fig. 7.16.

β	β	√	√	β	β	β	β	β	β	β	β	β	β	β	β
β	β	0	0	0	0	1	0	0	1	0	0	0	0	β	β

Fig. 7.16 After completion of the second cycle of copying

The process continues till all the 0s of number m are check-marked and the corresponding copies of 0s of number n are made after the second separator symbol 1. When this is done, $m \times n$ numbers of 0s have been copied after the second separator symbol '1' and the multiplication process is complete.

Scanned by CamScanner

Now, the Turing machine halts to show the result. The condition of the Turing tape after completion of the multiplication process is shown in Fig. 7.17.

β	β	√	√	√	√	β	β	β	β	β	β	β	β	β	β	β	β
β	β	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0

Fig. 7.17 After completion of the multiplication process

The following is the design of a two-track Turing machine for multiplication:

1. The r/w head initially points to cells of the column containing the first 0 of the number m, in the state q_0, and does the following:
   It puts a check mark (√) in the upper track, keeps the symbol 0 as it is, and changes the state to q_1. The corresponding transition function is

   $\delta([q_0, \beta], [\beta, 0]) \vdash ([q_1, \beta], [\sqrt{}, 0], R)$.

2. In the state q_1, the r/w head continues to move towards the right till it keeps receiving 0's in the data track. The corresponding transition function is

   $\delta([q_1, \beta], [\beta, 0]) \vdash ([q_1, \beta], [\beta, 0], R)$.

3. As the r/w head reaches the first separator symbol '1' it changes its state to q_2 and moves towards the right. The corresponding transition function is

   $\delta([q_1, \beta], [\beta, 1]) \vdash ([q_2, \beta], [\beta, 1], R)$.

4. In the state q_2, it marks an * in the upper track of the first available 0 of number n, changes the state to q_3, and moves towards the right. The corresponding transition function is

   $\delta([q_2, \beta], [\beta, 0]) \vdash ([q_3, \beta], [*, 0], R)$.

5. In the state q_3, it continues to move towards the right till all the 0s of number n are passed over and the second separator symbol '1' is reached. The corresponding transition function is

   $\delta([q_3, \beta], [\beta, 0]) \vdash ([q_3, \beta], [\beta, 0], R)$.

6. When the second separator symbol '1' is reached, the r/w head changes its state to q_4, and moves towards the right. The corresponding transition function is

   $\delta([q_3, \beta], [\beta, 1]) \vdash ([q_4, \beta], [\beta, 1], R)$.

7. In the state q_4, it converts the blank symbol on the data track to 0, changes the state to q_5, and turns left. The corresponding transition function is

   $\delta([q_4, \beta], [\beta, \beta]) \vdash ([q_5, \beta], [\beta, 0], L)$.

   In addition, in state q_4, the r/w head may have to pass over the previously inserted 0s. The corresponding transition function is

   $\delta([q_4, \beta], [\beta, 0]) \vdash ([q_4, \beta], [\beta, 0], R)$

8. In the state q_5, it continues to move towards the left till the second separator symbol '1' is reached, changes its state to q_6, and moves towards the left. The corresponding transition function is

   $\delta([q_5, \beta], [\beta, 1]) \vdash ([q_6, \beta], [\beta, 1], L)$.

Scanned by CamScanner

In addition, in state q_5, the r/w head may have to pass over the previously inserted 0s. The corresponding transition function is

$\delta([q_5, \beta], [\beta, 0]) \vdash ([q_5, \beta], [\beta, 0], L)$.

9. In the state q_6, the r/w head continues to move towards the left keeping the tape cell contents unchanged until [*, 0] is reached. The corresponding transition function is

   $\delta([q_6, \beta], [\beta, 0]) \vdash ([q_6, \beta], [\beta, 0], L)$.

   On reaching [*, 0], it turns right and changes the state to q_2. The corresponding transition function is

   $\delta([q_6, \beta], [*, 0]) \vdash ([q_2, \beta], [*, 0], R)$.

10. In the state q_2, if it receives $[\beta, 0]$, then the action performed is as given in steps 4–9. If it receives $[\beta, 1]$, this is an indication of the completion of one cycle of copying of n. The r/w head turns left and changes its state to q_7. The corresponding transition function is

    $\delta([q_2, \beta], [\beta, 1]) \vdash ([q_7, \beta], [\beta, 1], L)$.

11. In the state q_7, the r/w head continues to move towards the left till it continues to receive [*, 0]. It keeps on cancelling the * by replacing it with the blank symbol β. The corresponding transition function is

    $\delta([q_7, \beta], [*, 0]) \vdash ([q_7, \beta], [\beta, 0], L)$.

    The process continues till $[\beta, 1]$ is reached.

12. When $[\beta, 1]$ is reached in the state q_7, the r/w head changes the state to q_8 and moves towards the left. The corresponding transition function is

    $\delta([q_7, \beta], [\beta, 1]) \vdash ([q_8, \beta], [\beta, 1], L)$.

13. In the state q_8, the r/w head continues to move towards the left till it continues to receive $[\beta, 0]$. The corresponding transition function is

    $\delta([q_8, \beta], [\beta, 0]) \vdash ([q_8, \beta], [\beta, 0], L)$.

    As soon as it receives a [√, 0], it turns right and changes the state to q_0. The corresponding transition function is

    $\delta([q_8, \beta], [\sqrt{}, 0]) \vdash ([q_0, \beta], [\sqrt{}, 0], R)$.

14. In the state q_0 if the r/w receives $[\beta, 1]$, it indicates the completion of the multiplication process. The Turing machine halts in the final state q_f. The corresponding transition function is

    $\delta([q_0, \beta], [\beta, 1]) \vdash ([q_f, \beta], [\beta, 1], R)$.

Example 7.18 Design a two-track Turing machine to check if a given string over $\{a, b, c\}$ is a palindrome.

Solution The initial condition of the Turing tape is shown in Fig. 7.18. The input string is in the lower data track and the upper track is empty, containing all blanks symbols.

β	β	β	β	β	β	β	β	β	β	β	β	β	β	β	β
β	β	a	b	b	c	b	b	c	b	b	a	β	β	β	β

Fig. 7.18 Initial condition of the Turing tape

Scanned by CamScanner

$\delta([q_2, a], [\sqrt{}, c]) \vdash ([q_f, \beta], [\sqrt{}, c], R)$
$\delta([q_2, b], [\sqrt{}, a]) \vdash ([q_f, \beta], [\sqrt{}, a], R)$
$\delta([q_2, b], [\sqrt{}, b]) \vdash ([q_f, \beta], [\sqrt{}, b], R)$
$\delta([q_2, b], [\sqrt{}, c]) \vdash ([q_f, \beta], [\sqrt{}, c], R)$
$\delta([q_2, c], [\sqrt{}, a]) \vdash ([q_f, \beta], [\sqrt{}, a], R)$
$\delta([q_2, c], [\sqrt{}, b]) \vdash ([q_f, \beta], [\sqrt{}, b], R)$
$\delta([q_2, c], [\sqrt{}, c]) \vdash ([q_f, \beta], [\sqrt{}, c], R)$

9. The Turing machine halts in the state q_f to accept the string.

7.6.2 Multitape Turing Machine

Another extension proposed to the Turing machine was the multitape Turing machine. In the multitrack Turing machine, all the tracks reside on the same tape and move in the same direction, that is, the direction of the tape. In the multitape Turing machine, each track resides on a different tape and can move in the desired direction independent of others. Another aspect that makes the multitape Turing machine more powerful is that it is also possible for a track to remain stationary if the movement is not required. Thus, for a track in a multitape Turing machine there are three options: to move towards the left (L), to move towards the right (R) or to remain stationary (S). Figure 7.20 shows the basic architecture of the multitape Turing machine.

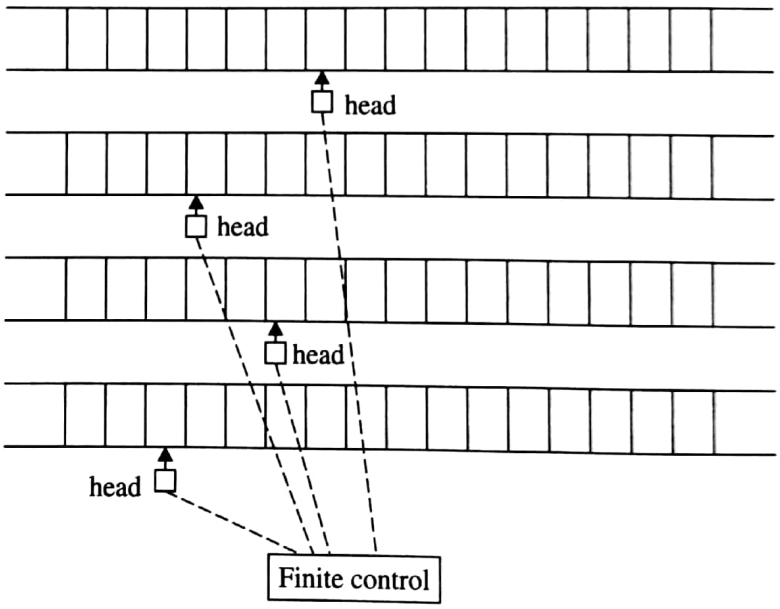

Fig. 7.20 Architecture of a multitape Turing machine

In a multitape Turing machine, there is a finite control that is connected to all the heads, with each head pointing to a track on each tape. The finite control has a state as is the case with the r/w head of a single tape Turing machine. Initially,

1. the finite control is in the initial state, say q_0;

Scanned by CamScanner

2. the r/w head in the lowermost tape points to the cell containing leftmost symbol of the input string/data; and
3. all the cells in the upper tapes contain the blank symbols β. Since all these tapes are of infinite length, we say that the heads on the tapes are pointing to some arbitrary cell in the beginning.

When a transition occurs

1. the finite control may change its state;
2. the r/w head reads the symbol from the current cell and writes a symbol in it. This symbol may be a new symbol or the same as the previous one; and
3. each head can either move towards the left (L) or the right (R) or stay stationary (S).

An example transition function, on an n tape Turing machine is as follows:

$$\delta([q_i, k_1, k_2, k_3, \ldots, k_{n-1}], [a_1, a_2, a_3, \ldots, a_n]) \vdash ([q_j, l_1, l_2, l_3, \ldots, l_{n-1}], [b_1, b_2, b_3, \ldots, b_n], [D_1, D_2, D_3, \ldots, D_n])$$

Here, the finite control is currently in the state q_i, it remembers the symbols $[k_1, k_2, k_3, \ldots, k_{n-1}]$, reads the symbol a_1 from the uppermost tape, symbol a_2 from the next uppermost tape, and so on. After reading, it changes its state to q_j, remembers the symbols $[l_1, l_2, l_3, \ldots, l_{n-1}]$, and replaces the symbol a_1 by b_1; a_2 by b_2; ...; a_n by b_n. The r/w head is on the ith tape as per the direction D_i given in the transition function.

The usefulness of the multitape Turing machine can be illustrated with the help of an example language $L = \{a^n b^n | n \geq 1\}$. On a normal Turing machine, the r/w head will be making many to and fro rounds on the Turing tape to match each pair of symbols a and b. On a multitape Turing machine, there is no need for such rounds of the head. The matching task can be done in $k/2$ moves for a string of length k. This can be done by making a copy of the string on another tape, say tape 2. The head of tape 1(where the string originally lies) can be fixed on the first 'a' of the input string and the head of tape 2 can be positioned on the first 'b' of the input string. Now, the heads advance on both tapes simultaneously towards the right and the string is accepted if there are equal number of a's and b's in the input string. This will happen if the head on tape 1 encounters the first 'b' of the input string and the head on tape 2 encounters the first blank after the string, simultaneously. Thus, each head needs to traverse only half of the string.

Simulating multitape Turing machine on single tape

It is possible to simulate an n-tape Turing machine on a single tape. In such a case, the single-tape Turing machine will contain $2n$ number of tracks. Each tape is simulated by a pair of tracks. In each pair of tracks, the cell in the lower track keeps the contents of the corresponding cell on the tape and in the upper track all cells contain blank symbols except one, which contains the symbol H to indicate the position of the r/w head. The r/w head of the corresponding tape points to a cell just below it. Figure 7.21 shows the simulation of a 3-tape Turing machine on the single-tape Turing machine.

Scanned by CamScanner

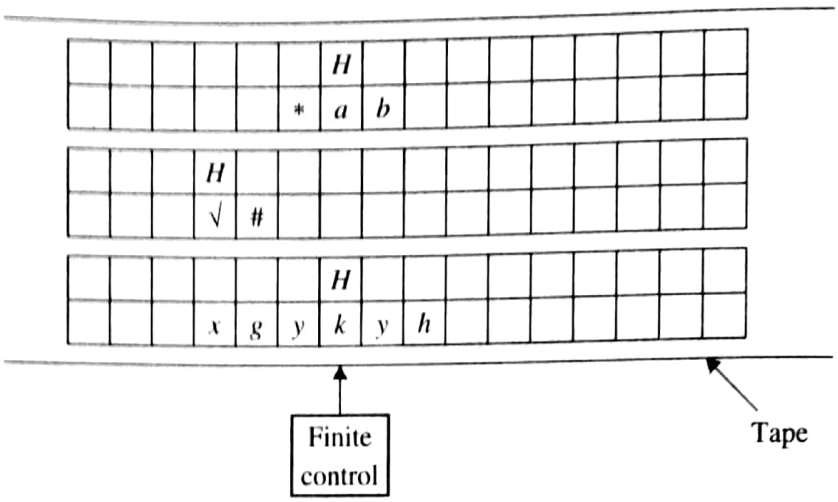

Fig. 7.21 Simulation of three-tape Turing machine on a single-tape Turing machine

7.7 TURING MACHINE AS ENUMERATOR

As described earlier, a Turing machine is a language recognizer and a language is a set of valid strings. The task that we have been performing so far has been to take an arbitrary string w and to check its belongingness to an arbitrary language L on a Turing machine specifically designed for L. Now, the question arises as to whether it is possible to create the set L using a Turing machine. The answer to this question is *yes* to an extent. The reason for this is that the language set is normally not finite and it is impossible to create a set of infinite length. If it is desirable that the set contains the strings of length up to k, only then the set becomes finite and can therefore be created. The Turing machine doing this task is known as an enumerator Turing machine.

An enumerator Turing machine as shown in Fig. 7.22 contains three tapes.

1. The first tape contains all the strings up to the length k belonging to Σ^* in the canonical order. The canonical order of strings starts with the string of length 0, i.e., ϵ. This is followed by all the strings of length 1, followed by all the strings of length 2 and so on. For example, the canonical order of strings over $\Sigma = \{a, b, c\}$ is as follows: ϵ, *a*, *b*, *c*, *aa*, *ab*, *ac*, *ba*, *bb*, *bc*, *ca*, *cb*, *cc*, *aaa*, *aab*, *aac*, *aba*, *abb*, *abc*, *aca*, *acb*, *acc*, *baa*, *bab*, *bac*, *bba*, *bbb*, *bbc*, *bca*, *bcb*, *bcc*, *caa*, *cab*, *cac*, *cba*, *cbb*, *cbc*, *cca*, *ccb*, *ccc*, *aaaa*, *aaab*, ...
2. The second tape contains the code for the Turing machine.
3. The third tape is used for the purpose of enumeration.

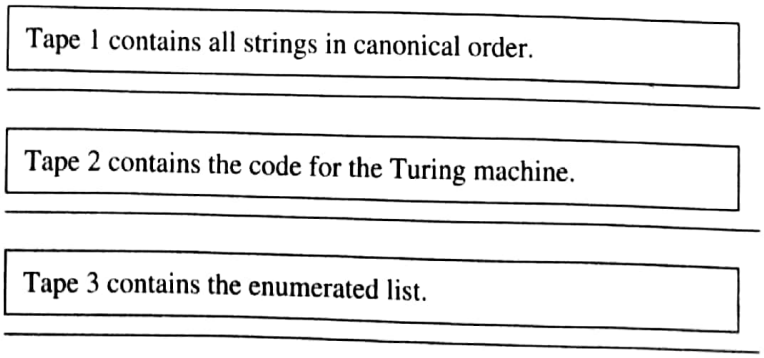

Fig. 7.22 Turing machine as an enumerator

Scanned by CamScanner

The Turing machine works as follows:

It reads the first string from tape 1 and tests it on the Turing machine coded on tape 2. If the string is accepted by the Turing machine then it is written on tape 3. The Turing machine now reads the next string from tape 1 and so on. The process continues till all the strings on tape 1 are exhausted. Now, tape 3 contains the strings (up to a given length) belonging to the language under consideration.

7.8 NONDETERMINISTIC AND DETERMINISTIC TURING MACHINE

A nondeterministic Turing machine (NDTM) is one in which corresponding to a given state and an input symbol there can be multiple options. The transition function of an NDTM looks as follows:

$$\delta(q_i, k) \vdash \{(q_1, a_1, D_1), (q_2, a_2, D_2), (q_3, a_3, D_3), \ldots, (q_n, a_n, D_n)\}$$

where D_i is the direction of motion. As can be seen, given the current state as q_i and the input symbol k, the Turing machine has n number of options. The value of n can be different for different combinations of current state and input symbols. If $n = 1$, then the behaviour of the Turing machine is deterministic corresponding to that transition function, though the overall behaviour of the machine is nondeterministic. Like nondeterministic finite automaton and PDA, a given input string w is accepted by the NDTM if there is some path, out of the many possible, from the initial state to the final state where the Turing machine halts. It is possible to make an equivalent deterministic Turing machine (DTM) corresponding to a given NDTM as follows:

Let M_n be an NDTM and M_d be the corresponding deterministic version. Then M_d can be simulated using a multitape machine as shown in Fig. 7.23. The machine in Fig. 7.23 contains a finite control that knows all the transition functions. The machine contains two tapes—tape 1 which contains the current ID, IDs explored so far, and the ID to be explored yet and the second tape (tape 2) which is an auxiliary tape.

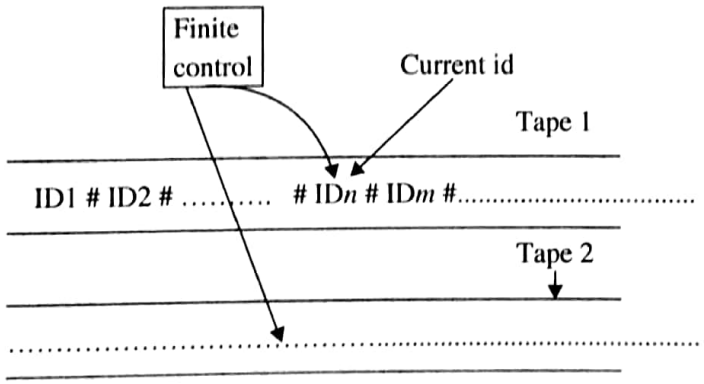

Fig. 7.23 DTM corresponding to an NDTM

The machine works as follows:

The finite control writes the initial ID on tape 1 and does the following:

1. If the current state is a final one and there is move on the RHS of the transition function then the Turing machine halts to indicate the acceptance of the string.
2. If the current state is not a final state, then it takes the RHS of the transition function and copies it on tape 2.

Scanned by CamScanner

3. Now, the contents of tape 2 are transferred on the end of tape 1.
4. The finite control advances towards the right to explore the next ID on tape 1 and repeats the steps 1–3.
5. The process continues till the machine reaches a final state to accept the string or there is no more ID on tape 1 to explore. If there is no more ID on tape 1 to explore, then the Turing machine halts to indicate non-acceptance.

It can be seen that the DTM M_d explores the IDs of NDTM M_n in a breadth-first search manner.

7.9 ALTERNATIVE REPRESENTATION OF TURING MACHINES

The model of the Turing machine that we have used so far contains a tape that is infinite on both sides. The same model of the Turing machine can be represented in other ways as well. In this section, we discuss two alternative representations for the normal Turing machine—single tape and single track. The first representation folds the Turing tape in the form of a two-track one-way infinite tape with the first symbol of the input string as the folding point. The second representation uses two stacks, with one holding the string on the LHS of the current position of the head and the other holding the string on the RHS of the head.

7.9.1 Turing Machine with One-way Infinite Tape

It is also possible to fold the Turing tape and convert it into a tape that has a beginning on one side and extends to infinity on the other side. This type of mechanism is easier to implement and the infinite length on one side can be implemented by using a tape of sufficient length. The place of folding is the place in the Turing tape from where the input string begins, that is, the place the r/w head points to initially. Figure 7.24(a) shows the Turing tape of a normal Turing machine and Fig. 7.24(b) shows its folded version. In the folded version, the lower track shows the part of the Turing tape to the left of the input string. The upper track shows the part of the Turing tape beginning with the input string and after the input string. The folded version of the Turing machine contains a one-way infinite tape and is referred to as the *semi-infinite tape version*.

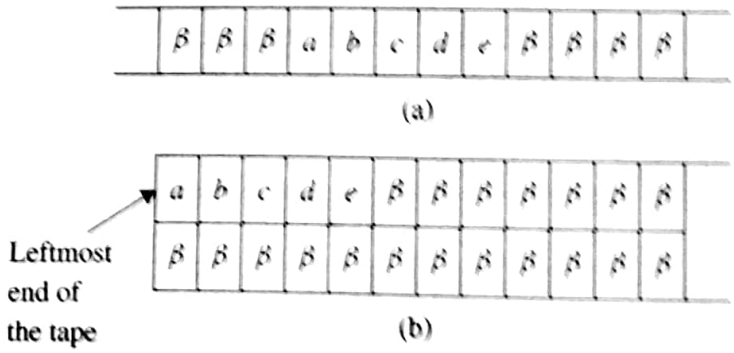

Fig. 7.24 Types of Turing tape (a) Tape of a normal Turing machine—two-way infinite tape (b) Folded tape version of a Turing machine—one-way infinite tape

In the semi-infinite tape version of the Turing machine, it is necessary that the r/w head does not move to the left of the leftmost end of the tape. This can be achieved by inserting a marker. Let us insert a # mark in the leftmost cell of the lower track and insert a restriction on the

Scanned by CamScanner

machine whenever a # mark is encountered in the lower track; left movement must be converted to climbing to the upper track. The condition of the Turing machine after inserting the marker symbol appears as shown in Fig. 7.25.

a	b	c	d	e	β	β	β	β	β	β	β
#	β	β	β	β	β	β	β	β	β	β	β

Fig. 7.25 Turing machine with the folded tape and left-end marker

Simulation of two-way infinite tape version on semi-infinite tape version

It is possible to simulate every possible move of a normal Turing machine on a machine with semi-infinite tape. Let M_1 be a normal Turing machine and M_2 be its equivalent semi-infinite tape version. Given M_1, M_2 can be created as follows:

In M_1, at a particular instant, the r/w head can either be on the left side of the first symbol of the input string, on the first symbol of input string, or on the right side of first symbol of input string. The corresponding positions in M_2 are as follows:

Upper track: if the r/w head is on the first symbol of the input string or on its right side
Lower track: if the r/w head is on the left side of the first symbol of the input string

Thus, the state parameter in M_2 contains two parts $[q_i, \text{U/L}]$ where q_i is the state of the finite control and the second parameter indicates the track being accessed. If it is U, then the upper track is being accessed; if it is L, then the lower track is being accessed. The equivalence between the machines M_1 and M_2 is created as follows:

1. The initial transitions in M_2 involve writing the # mark in the lower track and then pointing the first symbol in the upper track. The corresponding transitions are as follows:

   $\delta([p_0, \text{L}], [a, \beta]) \vdash ([p_1, \text{U}], [a, \#], \text{R})$
   $\delta([p_1, \text{U}], [b, \beta]) \vdash ([s_0, \text{U}], [b, \beta], \text{L})$

   Here a is some input symbol belonging to Σ and is the first symbol in the input string. The first transition writes a # mark in the leftmost cell of the lower track and the r/w head moves towards the right. The second transition function brings the r/w head to the beginning of the input string and the state is now s_0, which corresponds to the initial state q_0 of M_1.

2. The transition function

   $\delta(q_i, x) \vdash (q_j, y, \text{L/R})$

   in the Turing machine M_1 will have two possible equivalent functions in M_2 depending upon whether x is on the upper track or the lower track.

   (a) If x is on the upper track then the corresponding equivalent transition function is

   $\delta([s_i, \text{U}], [x, z]) \vdash ([s_j, \text{U}], [y, z], \text{L/R})$

   where z is some tape symbol in M_1.

Scanned by CamScanner

(b) If x is on the lower track then the corresponding equivalent transition function is

$$\delta([s_i, \text{L}], [z, x]) \vdash ([s_j, \text{L}], [z, y], \text{R/L})$$

The direction of motion is R/L, which indicates that the finite control will move in the opposite directions when x is on the lower track.

3. The transition function

$$\delta(q_i, x) \vdash (q_j, y, \text{R})$$

in the Turing machine M_1 will have the following equivalent transition function in M_2 if x is the leftmost symbol on the upper track.

$$\delta([s_i, \text{U}], [x, \#]) \vdash ([s_j, \text{U}], [y, \#], \text{R})$$

4. The transition function

$$\delta(q_i, x) \vdash (q_j, y, \text{L})$$

in the Turing machine M_1 will have the following equivalent transition function in M_2 if x is the leftmost symbol on the upper track.

$$\delta([s_i, \text{U}], [x, \#]) \vdash ([s_j, \text{L}], [y, \#], \text{R})$$

5. The transition function

$$\delta(q_i, x) \vdash (q_j, y, \text{R})$$

in the Turing machine M_1 will have the following equivalent transition function in M_2 if x is the second leftmost symbol (after #) on the lower track.

$$\delta([s_i, \text{L}], [a, x]) \vdash ([s_j, \text{U}], [a, y], \text{L})$$

6. The transition function

$$\delta(q_i, x) \vdash (q_j, y, \text{L})$$

in the Turing machine M_1 will have the following equivalent transition function in M_2 if x is the second leftmost symbol (after #) on the lower track.

$$\delta([s_i, \text{L}], [a, x]) \vdash ([s_j, \text{L}], [a, y], \text{R})$$

The accepting states F_2 in M_2 are the states $F_1 \times \{\text{U, L}\}$ where F_1 is the acceptor state for M_1. For example, if $F_1 = \{s_1, s_2\}$, then $F_2 = \{[s_1,\text{L}], [s_1,\text{U}], [s_2, \text{L}], [s_2, \text{U}]\}$.

7.9.2 Two-stack Machine

Another way to implement a Turing machine is through a two-stack machine. Let M be an arbitrary Turing machine and S be its corresponding two-stack equivalent. The machine S contains two stacks, one holding the portion of the input string to the left of the r/w head and the other holding the portion of the string to the right of it, including the symbol to which it is pointing. The stacks (stack 1 and stack 2) hold the symbols Z_1 and Z_2 initially. Figure 7.26 shows the instantaneous state of a Turing machine.

The corresponding equivalent state of a two-state machine is shown in Fig. 7.27.

Scanned by CamScanner

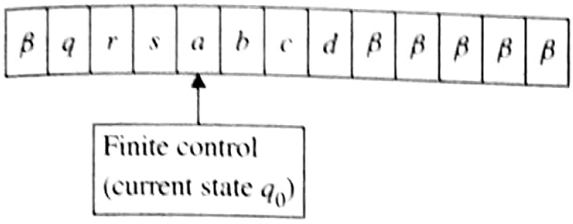

Fig. 7.26 Instantaneous state of a Turing machine

Fig. 7.27 Two-stack equivalent of a Turing machine

Stack 1 holds the part of the input string to the left of the finite control and stack 2 holds the part of the input string to the right of the finite control. The symbol currently under the finite control is on the top of the stack 2.

The process begins as follows:

1. The markers Z_1 and Z_2 are pushed into stack 1 and stack 2 respectively. This is equivalent to creating the blank tape of the Turing machine.
2. The input string is pushed into stack 1 symbol by symbol. Now, each symbol is popped from stack 1 and pushed into stack 2. This makes stack 1 empty except for the marker symbol Z1. Stack 2 now contains the input string with the leftmost symbol of the input string on the top of the stack.
3. The situation in *S*, after step 2, is the same as that of a Turing machine with the r/w head pointing to the first symbol of the input string.

Simulation

The two-stack machine *S* simulates the transition functions of the Turing machine *M* as follows:

Let *M* be in state q_i and x be the symbol currently under finite control. Given the transition function

$$\delta(q_i, x) \vdash (q_j, y, \mathrm{R})$$

S implements this transition function as follows:

1. *S* changes its state from s_i to s_j.
2. It pops the symbol x from stack 2.
3. It pushes the symbol y in stack 1.

Given the transition function

$$\delta(q_i, x) \vdash (q_j, y, \mathrm{L})$$

S implements this transition function as follows:

1. S changes its state from s_i to s_j.
2. It pops the symbol x from stack 2.
3. It pushes the symbol y in stack 2.
4. It pops the topmost symbol from stack 1 and pushes it in stack 2.

This replaces x by y and brings the symbol to the left of x on the top of stack 2.

Scanned by CamScanner

Given the transition function

$\delta(q_i, x) \vdash (q_j, \beta, \mathrm{R})$

S implements this transition function as follows:

1. S changes its state from s_i to s_j.
2. It pops the symbol x from stack 2.

Given the transition function

$\delta(q_i, x) \vdash (q_j, \beta, \mathrm{L})$

S implements this transition function as follows:

1. S changes its state from s_i to s_j.
2. It pops the symbol x from stack 2.
3. It pops the topmost symbol from stack 1 and pushes it in stack 2.

Special conditions arise when the symbols under consideration are Z_1 and Z_2.

A transition function $\delta(q_i, \beta) \vdash (q_j, \beta, \mathrm{L})$ has two versions depending upon whether the blank symbol is on the left of the input string or on the right of the input string. If it is on the left of the input string, then the marker Z_1 is encountered by the blank symbol and if it is on the right of the input string the marker Z_2 is encountered.

If Z_1 is encountered, then the corresponding transition is implemented as follows:

1. S changes its state from s_i to s_j.
2. It maintains the marker Z_1 in the stack 1.

If Z_2 is encountered, then the corresponding transition is implemented as follows:

1. S changes its state from s_i to s_j.
2. It pops a symbol from stack 1 and pushes it into stack 2.

Similarly, a transition function $\delta(q_i, \beta) \vdash (q_j, \beta, \mathrm{R})$ has two versions.

If Z_1 is encountered, then the corresponding transition is implemented as follows:

1. S changes its state from s_i to s_j.
2. It pops a symbol from stack 2 and pushes it into stack 1.

If Z_2 is encountered, then the corresponding transition is implemented as follows:

1. S changes its state from s_i to s_j.
2. It maintains the marker Z_2 in the stack 2.

The machine S accepts the string if it is in the final state corresponding to M and comes to halt because of non-availability of transition function.

7.10 TIME AND SPACE COMPLEXITY OF TURING MACHINE

The time and space complexity of a Turing machine is only described in the case of recursive language. A recursive language is one in which the Turing machine essentially halts for every given string $w \in \Sigma^*$ to show the acceptance or rejection. The topic of recursive language will be taken up in detail in Chapter 8; the only thing that needs to be understood at this point is that time and space complexity is defined only for recursive language.

Scanned by CamScanner

When a Turing machine M takes up a string w and a recursive language L, then it gives the output in the form of *yes* or *no* to indicate whether the string $w \in L$ or not. In such a case, it is possible to exactly measure the number of moves made by the r/w head of the Turing machine and the maximum space used on the Turing tape during the processing of the string. Although the number of moves made by the Turing machine to process a string may depend on that specific instance of the string, it is possible to make an estimate of the maximum possible number of moves that the Turing machine will make to accept or reject a string of length n. It is because the notion of a Turing machine is that of an algorithm in case of recursive languages.

In case of a recursively enumerable language (discussed in Chapter 8) where the Turing machine may not halt if $w \notin L$, it is possible to estimate the number of moves that a Turing machine will make to accept a string of length n but it is not possible to estimate the number of moves required to reject a string of length n as the machine may not halt in this case.

Thus, in case of a Turing machine it is possible to find out the time and space complexity if and only if it has been designed for a recursive language. Now, we are in a position to define the time and space complexity of a Turing machine.

For a Turing machine M, the time complexity τ for a natural number n, written as $\tau_M(n)$, is the maximum number of moves that M shall make to accept/reject a string of length n in the worst case scenario.

For a Turing machine M, the space complexity s for a natural number n, written as $s_M(n)$, is the maximum number of cells that M shall make use of, on Turing tape, to accept/reject a string of length n in the worst case scenario.

To understand things better, let us see some example languages.

Let us take the Turing machine M to accept the language $L = \{ww^R \mid w \in (a, b)^+\}$ over $\Sigma = \{a, b\}$. The machine works as follows:

It checks the first symbol of the input string, remembers it by change of state, converts it into blank, and travels the whole of the input string and reaches the blank symbol just following the input string. It turns back and matches the last symbol of string with the first one. If the match occurs, then this last symbol is also converted into blank. The r/w travels back to the second symbol of the input string and then to find its corresponding match. The worst case is one in which the input string is valid and the Turing machine has to match all the corresponding pairs. In addition, with every successful match the number of moves required to match the next symbol decreases. The process stops when all the symbols are successfully matched and no move is required. Now, the Turing machine halts.

Let us find the number of moves required by the Turing machine to match a valid string of length n. The number of moves required to match the first pair of symbols are $2n + 1$, which includes both forward and backward movements. The number of moves required to match the next pair of symbols is $2n - 3$. When all the pairs are successfully matched, the Turing machine makes just one move and finds a blank symbol, which brings it to halt.

The total number of moves made by a Turing machine for processing a string of length n is computed as follows:

$$
\begin{aligned}
&(2n + 1) + (2n - 3) + (2n - 7) + \cdots + 1 \\
&\quad = ((2n + 1) + 1)/2 \times (n/2 + 1) \\
&\quad = (n + 1) \times (n/2 + 1) \\
&\quad = n^2/2 + 3n/2 + 1
\end{aligned}
$$

Scanned by CamScanner

Hence, the time complexity of a Turing machine to recognize the language $L = \{ww^R \mid w \in (a, b)^+\}$ is $O(n^2)$. To process the string of length n the Turing tape requires a minimum of $n + 2$ cells, n for the string and 2 cells for the blanks on either side of the string. Hence, the space complexity is of the order $O(n)$.

Consider a Turing machine to recognize the language $L = \{ww \mid w \in (a, b)^+\}$. The basic mechanism of string processing involves the following steps:

1. Marking the middle of the string
2. Matching the two halves on either side of the string

The worst case arises when the input string w belongs to the language L and the matching process occurs completely after marking the middle. To mark the middle of the string the r/w head converts the leftmost symbol of the string to p if it is a and to q if it is b. Then it travels to pass over the whole of the string and returns from the blank symbol just following the input string. Now, it converts the rightmost symbol to x if it is a and to y if it is b. This completes the conversion of one pair of symbols. The r/w head comes back for the conversion of the second pair of symbols. Thus, the steps required to convert the first pair of symbols is $2n + 1$ for the string of length n. For the second pair of symbols (second from both sides) the number of moves made by the Turing machine is $2n - 3$. The process is completed when all the symbols are converted either to p/q or x/y as follows:

$\beta abbbabbb\beta$
$\beta pqqqxyyy\beta$

The total number of moves required for this conversion process is as follows:

$$\begin{aligned}
&(2n + 1) + (2n - 3) + (2n - 7) + \cdots + 1 \\
&\quad = ((2n + 1) + 1)/2 \times (n/2 + 1) \\
&\quad = (n + 1) \times (n/2 + 1) \\
&\quad = n^2/2 + 3n/2 + 1
\end{aligned}$$

When the conversion process is complete, the r/w head points to the middle character of the string. Now, the r/w head comes back to the first symbol of the input string, for which it returns from the blank symbol just preceding the leftmost symbol of the string. The number of moves required for the process is $n/2 + 1$.

Now, the matching process starts. For matching each pair of symbols p/x or q/y, the r/w head travels $n/2 + 1$ number of steps in the forward direction and then comes back by $n/2$ number of steps to start the next matching. Thus, the total number of moves required for matching one pair of symbols is $n/2 + 1 + n/2 = n + 1$.

For $n/2$ pairs of symbols, the required number of moves is $n/2 \times (n + 1) = n^2/2 + n/2$.

Thus, the total number of moves required to process a string of length n is as follows:

$$n^2/2 + 3n/2 + 1 + n/2 + 1 + n^2/2 + n/2 = n^2 + 5n/2 + 2$$

Hence, the time complexity of the Turing machine to recognize the language $L = \{ww \mid w \in (a, b)^+\}$ is $O(n^2)$. To process the string of length n the Turing tape requires a minimum of $n + 2$ cells, n for the string and 2 cells for the blanks on either side of the string. Hence, the space complexity is of the order $O(n)$.

Scanned by CamScanner

This discussion shows that the time complexity in the case of both Turing machines is of the order of n^2. This is because the machine makes repeated back and forth movements to match the symbols. Each back and forth movement of the Turing machine is associated with the conversion/matching of one symbol. If one converts/matches n symbols at a time then the time complexity of the machine shall be reduced, although at the cost of design complexity. Time complexity can also be reduced if the multitrack or multitape versions are used effectively so that the back and forth movements of the machine are reduced.

Scanned by CamScanner