Theory of Formal Languages (Turing Machine and Computability)

Lecture 3: Feb. 23, 2021

Prof. K.R. Chowdhary *: Professor of CS*

3.1 Computability

We have represented a deterministic Turing Machine as $M = (Q, \Sigma, \Gamma, \delta, q_0, H)$ and input as $w \in \Gamma^*$. This input results in finite computation (configurations) sequence $C_0, C_1, C_2, \ldots, C_m$, where each C_i is a configuration of M and C_0 is initial configuration. If this Turing machine M halts and succeeds in C_m, then computation is a successful with computation length m. Then, M has computed a results, given the input w, and result is $f_M(w)$. Since M is deterministic, for same inputs $w = x$, $f_M(w) = f_M(x)$.

If, for some $w \in \Sigma^*$, $f_M(w)$ is not defined because M halts and fails or does not halt, then f_M is a *partial function* from Σ^* to Γ^* and f_M is a function computed by M. We say that a function $f : \Sigma^* \to \Gamma^*$ is *Turing Computable*, if and only if there is a Turing machine M, such that $f = f_M$.

Example 3.1 *Turing Machine computes the functions.*

The function $\mathbb{N}_0 \times \mathbb{N}_0 \to \mathbb{N}_0$ defined by $f(x, y) = x + y$ is Turing computable. A Turing machine can be constructed to compute the function $f : \mathbb{N}_0 \times \mathbb{N}_0 \to \mathbb{N}_0$ defined by $f(x, y) = x * y$.

In the same line, any polynomial function $f : \mathbb{N}_0 \to \mathbb{N}_0$ defined by $f(x) = a_0 + a_1x + a_2x^2 + \cdots + a_nx^n$ can be computed by a Turing machine, and the function is Turing computable.

Definition 3.2 Church-Turing Thesis. *A function can be computed by an algorithm iff it can be computed by a Turing machine.*

In addition to language acceptance we had discussed so far, a TM may be viewed as computer of functions from integers to integers, that is,

$$f : \mathbb{N}_0 \to \mathbb{N}_0. \tag{3.1}$$

The approach use representation of integers in union for $i \geq 0$, where integer i is represented by a string 1^{i+1}. If a function has k arguments $i_1, i_2, \ldots, i_k$, these arguments are initially placed on the tape separated by 0's, in the form:

$$1^{i_1}01^{i_2}0\ldots01^{i_{k-1}}01^{i_k}. \tag{3.2}$$

If a TM halts in state q', with ID (Instantaneous description) $q'1^{m+1}$, then it shows that the value m has been computed by function f as

$$f(i_1, i_2, ..., i_k) = m. \tag{3.3}$$

It may be noted that TM may compute a function of one argument, or a different function of two or more arguments. For a function of k arguments, all the k-tuples need not to have values. A function with all the arguments present is a *total recursive function*, where as a function with some of the tuples having values, is called *partial recursive function*. All the arithmetic functions, like $[\log n]$, n^1, 2^n etc., are total recursive functions.

The total recursive functions correspond to Recursive language, which always cause the TM to halt. Whereas, the partial recursive functions are like Recursively Enumerable languages, correspond to TMs, which may not halt on every input.

Example 3.3 *Given two positive integers x and y, construct a Turning Machine to computes the sum $x + y$.*

We first choose a convenient way to represent x and y on the tape. We use a unary notation so that weight of each number is represented by a sequence of 1's, i.e., $\{1\}^+$. Therefore, $w(x), w(y) \in \{1\}^+$, $|w(x)| = x$, and $|w(y)| = y$. m It is assumed that $w(x)$ and $w(y)$ are placed on the tape in sequence with a '0' separating both. Initially the TM is in state q_0, and read-write head is positioned to first character of $w(x)$. When computation is over, the result $w(x + y)$ is placed on the tape with read-write head positioned to first character of the result $w(x + y)$ and a blank (B) is placed after result. Therefore,

$$q_0w(x)0w(y) \vdash^* q'w(x + y)B, \tag{3.4}$$

where q' is a accepting or halting state. We note that the addition operation is simple. What is required is to move the separator 0 to end, and replace it with B, so that $w(x)$ and $w(y)$ are concatenated. This results to $w(x + y)$. It is important to note that in the process of concatenation, one extra '1' gets appended, which needs to be removed, so that:

$$1^{m+1}.1^{n+1} \text{ results to } 1^{m+n+1} \text{ and not } 1^{m+n+2}.$$

.

Let the TM be,

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

where,

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

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

$\Gamma = \{0, 1, B\}$, with left most symbol as B, and

The corresponding TM is shown in Fig. 3.1 and δ is given in transition table 3.1.

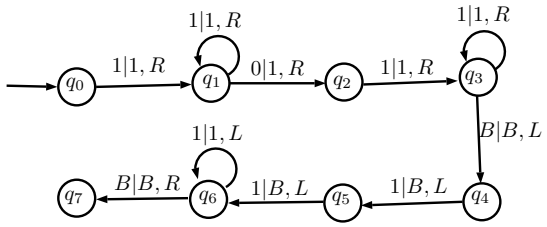

Figure 3.1: TM to add two integers in unary format

Table 3.1: Transition table for unary addition

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

In the process of moving 0 from middle of $w(x)$ and $w(y)$ to right, 1 gets added extra temporarily. This is deleted later on.

Let us consider $x = 2$ and $y = 3$, hence $w(2) = 111$, and $w(3) = 1111$. All the moves, in sequence, of the TM are given as follows:

$$
\begin{aligned}
Bq_0 11101111 &\vdash B1q_1 1101111B \vdash B11q_1 101111B \vdash B111q_1 01111B \\
&\vdash B1111q_2 1111B \vdash B11111q_3 111B \vdash B111111q_3 11B \\
&\vdash B1111111q_3 1B \vdash B11111111q_3 B \vdash B1111111q_4 1B \\
&\vdash B111111q_5 1BB \vdash B11111q_6 1BBB \vdash B1111q_6 11BBB \\
&\vdash B111q_6 111BBB \vdash B11q_6 1111BBB \vdash B1q_6 11111BBB \\
&\vdash Bq_6 111111BBB \vdash q_6 B111111BBB \vdash Bq_7 111111BBB
\end{aligned}
$$

This shows that, the sum of the numbers 2 and 3 is computed as 5 ($wt(5) = 11111 = 1^6$).

Note that, we have considered x and y as finite numbers (> 0). When zero is one of the arguments, we represents the integers as: $|x| = 1^{x+1}$, and $|y| = 1^{y+1}$, and for decimal zero, the representation is unary 1. For $|x| + |y|$ the final result is 1^{x+y+1}.

Further, the number of the configurations through which the TM goes are 19, and number of transitions or moves are 18. Thus, total 18 *elementary instructions* have been executed to compute the function $f(x, y) = f(2, 3)$. Note that, as the size of numbers x and y increases the the length of three loops along the states q_1, q_3, and q_6 will increase, linearly.

Example 3.4 *Find the expression for number of steps (i.e., elementary instructions) carried out by the Turing machine to sum two positive integers m and n.*

For this we refer to Fig. 3.1. The counts of steps (moves) with respect to different states, including loops, as follows:

q_0 to q_1: 1

q_1 to q_1: $m - 1$

q_1 to q_2: 1

q_2 to q_3: 1

q_3 to q_3: $n - 1$

q_3 to q_4: 1

q_4 to q_5: 1

q_5 to q_6: 1

q_6 to q_6: $m + n + 1$

q_6 to q_7: 1

Sum total of the above steps is $2(m + n + 3)$, where $m + n$ is total length of the input arguments, is same as length of the input w. This gives complexity of $O(n)$, for input $|w| = n$.

Example 3.5 *Show that function $f : \mathbb{N}_0 \times \mathbb{N}_0 \to \mathbb{N}_0$ defined by $f(x, y) = x - y$ is Turing Computable.*

To demonstrate that the above subtraction function is Turing computable, we construct a Turing machine that computes $f(x, y)$. We represent the integers x and y as unary number. The approach shall be as follows: search the beginning of y and for each 1 in y go back to the start of x and delete a symbol 1. For subtraction to be possible, x should be grater that y.

Example 3.6 *Construct a TM that increments a given binary number by unity.*

Let the machine be $M = (Q, \Sigma, \Gamma, \delta, s, H)$, where $Q = \{q_0, q_1, q_2, q_3\}$, $s = q_0$, and $q_3 \in H$. We store the binary number to be incremented, in reverse order on tape, i.e., its LSB (least significant bit) at start of the tape. So, if number to be increased is 13 (i.e., 1101), it is stored on tape as 1011, with initial configuration as Bq_01011B. Various transitions for this machine are given below.

1. While the RW head is moved to right, every 1 is replaced by 0, and when 0 is encountered, it is replaced by 1 and head moves back to begin without changing any bit value on tape.

2. If all bits are 1 then they are replaced by 0, and at end B is replaced by 1, and head scans back to begin of the string.

3. If there is only one B on tape, it is replaced by 1, and tape head returns back to first bit.

The Fig. 3.2 shows the transition diagram, and table 3.2 is transition table for this Turing machine.

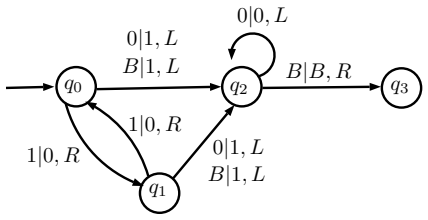

Figure 3.2: Turing Machine to increment a binary number

Table 3.2: Transition table for Fig. 3.2

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

Note that, if $w = 101110$, it becomes 101111, and for $w = 110111$ the result is 111000, and for $w = 111111$, result is 1000000.

Example 3.7 *Turing machine as function computer of concatenation.*

We compute the binary function of concatenation of strings over $\{a, b\}^*$. The initial configuration is q_0BuBvB. One or both or none of u and v may be null. The final configuration is q_fBuvB. The TM to concatenate two strings is shown in Fig. 3.3.

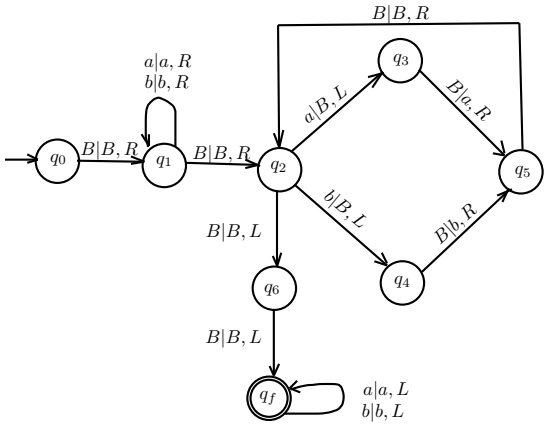

Figure 3.3: Turing Machine to concatenate two strings