LECTURE 6 – TURING MACHINES PART 2

Overview

- Turing Computability
- Multi-tape Turing Machine

Turing Computability

Let be the Turing machine $M = (Q , \Sigma , \Gamma , \delta , q_0 , F)$ and the input string $x \in \Sigma^*$ which results in a finite computation sequence $C_0, C_1, C_2,\ldots,C_m$ where each C_i is a configuration of M and C_0 is the initial configuration.

If the Turing machine M halts and succeeds in C_m, then the computation is a successful computation of length m.

Then, M has computed a result given the input x and the result is $f_M(x)$.

Since the machine M is deterministic then for the same input x=y, $f_M(x)=f_M(y)$.

For some $x \in \Sigma^*$, $f_M(x)$ is not defined because M halts and fails or does not halt.

This means that f_M is a partial function from Σ^* to Γ^* and f_M is the function computed by M.

Then, a function $f: \Sigma^* \rightarrow \Gamma^*$ is Turing computable if and only if there is a Turing machine such that $f=f_M$.

Example. The function $f:\{0,1\}^*\rightarrow\{0,1\}^*$ defined by f(x)=x+1 where x is a binary number is Turing computable. To demonstrate the above we construct a Turing machine that computes f(x) as illustrated in the Unit Addition example.

Example. The function $f:\aleph x\aleph\rightarrow\aleph$ defined by f(x,y)=x+y is Turing computable. To demonstrate the above we construct a Turing machine that computes f(x,y) as illustrated in Exercise 5.2.

A Turing machine can also be constructed to compute the function $f:\aleph x \aleph \rightarrow \aleph$ defined by $f(x,y)=x \cdot y$.

So, any polynomial function $f:\aleph \rightarrow \aleph$ defined by $f(x)=a_0+a_1x+a_2x^2+\ldots+a_nx^n$ can be computed by a Turing machine, i.e. is Turing computable.

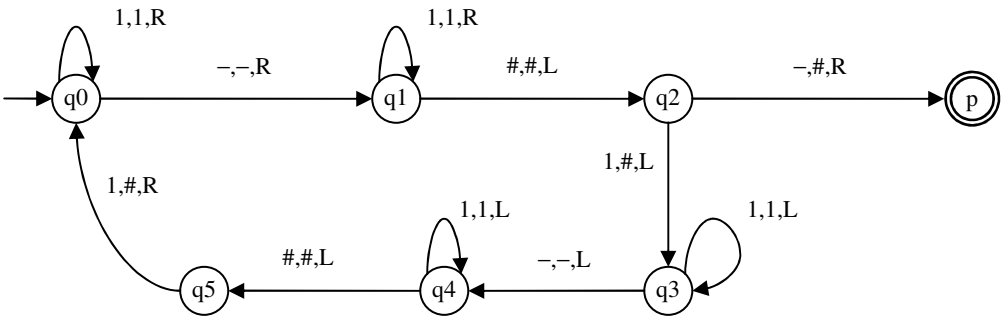
The Church's thesis (Church,1936) expresses that "every effectively computable function is Turing computable, i.e. is computable on a Turing machine."

Example. The function $f:\aleph x \aleph \rightarrow \aleph$ defined by $f(x,y)=x-y$ is Turing computable. The input is encoded as a unary string x–y, for example 11111–11 should produce the output 111.

To demonstrate that the above subtraction function is Turing computable we construct a Turing machine that computes f(x,y). The actions that should be carried out are:

1. Search the beginning of y and for each 1 in y go back to the start of x and delete one 1.

q_0 searches to the right for the – symbol
q_1 searches to the right for the least significant 1 in string y
q_2 deletes a 1 in y if any, otherwise the computation is finished
q_3 searches to the left of the – symbol
q_4 searches to the left for the most significant 1 in string x
q_5 replaces the most significant 1 in string x with a blanc

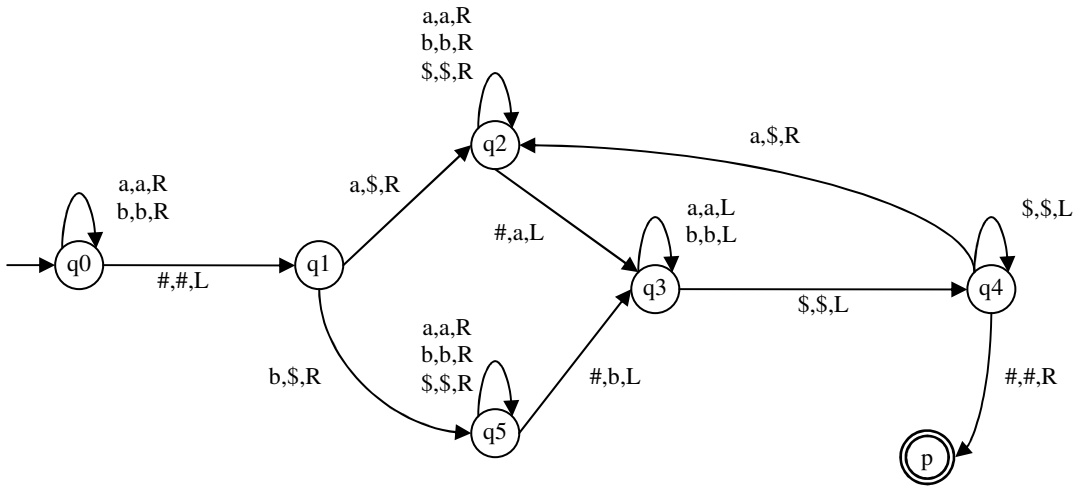


What will happen if x < y?

Example. The function INV(w) where w ∈ {a,b}* is Turing computable. This function denotes the reverse of string w. For example INV(aaba)=abaa.

To demonstrate that the above function is Turing computable we construct a Turing machine that computes INV(w). The actions to carried out are:

1. Search the symbol x_n at the end of the string w
2. Copy every symbol x_i to the left of x_n to the end of the string at the right of x_n and replace x_i by a special symbol $



Let $L \subset \Sigma^*$ be a language and let M be a Turing machine such that, for any string $x \in \Sigma^*$, if $x \in L$ then M(x)="yes" and if $x \notin L$ then M(x)="no".

Then we say that M decides L. If L is decided by some Turing machine M, then L is called a recursive language.

We say that M simply accepts L whenever, for any string $x \in \Sigma^*$, if $x \in L$ then M(x)="yes" but if $x \notin L$ then M(x)="undecided".

If L is accepted by some Turing machine M, then the language L is called recursively enumerable.

Example. The language L made of all strings of well-matched parenthesis is recursively enumerable because it can be decided by a Turing machine. To demonstrate the above we construct a Turing machine that decides L as illustrated in the Parenthesis Checker example.

Exercise 6.1 Give a recursive definition for the language L containing all strings of well-matched parenthesis.

Suppose that f is a function from Σ^* to Γ^* and let M be a Turing machine with alphabet Σ. We say that M computes f if for any string $x \in \Sigma^*$, M(x)=f(x). If such M exists, then f is called a recursive function.

Turing machines are equivalent in computation power to arbitrarily general (recursive) computer programs.

Turing machines can be thought of as algorithms for solving string-related problems. But even calculations can be expressed with strings, e.g. "110+11".

To solve problems whose instances are mathematical objects (graphs, networks, numbers, etc.) with a Turing machine, we must decide how to represent an instance of the problem with a string. Then, the algorithm for a decision problem is a Turing machine that decides the corresponding language.

Any finite mathematical object can be represented by a finite string over an appropriate alphabet.

Exercise 6.2 Construct a Turing machine that recognises palindromes over the alphabet {0,1}. For example $f_M(1)$=yes, $f_M(10101)$=yes, $f_M(110011)$=yes, $f_M(1101)$=no. The actions that should be carried out are:

1. Search the first symbol
2. Remembers the first symbol and goes to the end of string
3. Deletes the last symbol if it is the same as the one remembered
4. If there is only one symbol then the string is accepted

Multi-tape Turing Machine

A multi-tape Turing machine consists of n>1 infinite tapes and a head over each tape.

All heads are connected to the control and the control can be in any of a finite number of states.

The program specifies the next move which depends of the state of the control and the current symbol in each tape head.

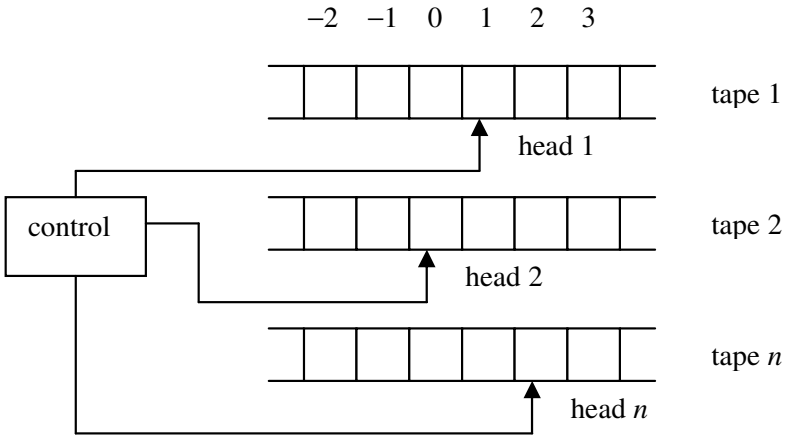
The machine can change state, write a new symbol in each tape and move each head independently to the right or to the left.

Typically, tape 1 is used for input and output and the other tapes are workspace.

In the initial configuration, the input is in locations 1,2,…,|x| of tape 1 and there are only blanc symbols in the other tapes.

The machine halts if enters a final state and the output is in tape 1.

Like in the single-tape Turing machine, auxiliary symbols are permitted.



Then, in a multi-tape Turing machine a move depends on the current input symbol in each tape and the current state in the control.

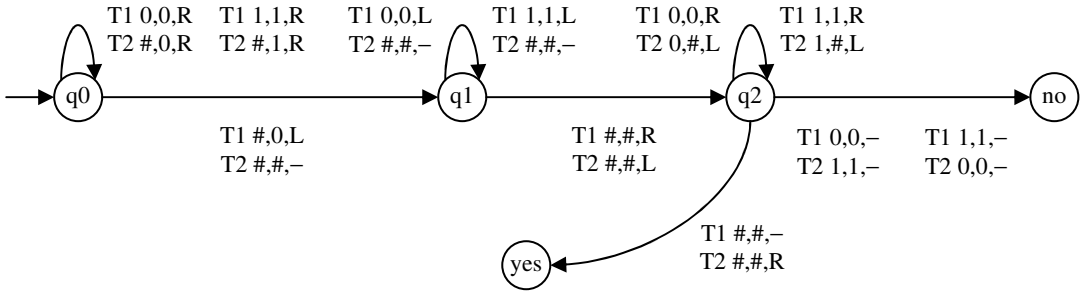
A step in a n-tape Turing machine program is an action which is specified as:

$$\delta(q,x_1,x_2,\ldots,x_n) = (p,y_1,D_1,y_2,D_2,\ldots, y_n,D_n)$$

The above means that if the machine is in state q and the head in tape 1 points to x_1, the head in tape 2 points to x_2, the head in tape 3 points to x_3 and so on, then the next state will be p, the head 1 writes y_1 and moves in the direction D_1, the head 2 writes y_2 and moves in the direction D_2, the head 3 writes y_3 and moves in the direction D_3 and so on.

Example. A 2-tape Turing machine computes palindromes more efficiently than the 1-tape Turing machine. The actions that should be carried out are:

1. Copies the input from tape 1 to tape 2
2. Positions head 1 at the beginning of the string in tape 1 and head 2 at the end of the string in tape 2
3. Moves the two heads in opposite directions checking that the two symbols are identical and erasing the copy in tape 2 at the same time



Processing of string "010":

$$\#q_0010\# \rightarrow \#0q_010\# \rightarrow \#01q_00\# \rightarrow \#010q_0\# \rightarrow \#01q_10\# \rightarrow$$
$$\#q_0\#\#\# \qquad \#0q_0\#\#\# \qquad \#01q_0\#\# \qquad \#010q_0\# \qquad \#010q_1\#$$

$$\#0q_110\# \rightarrow \#q_1010\# \rightarrow q_1\#010\# \rightarrow \#q_2010\# \rightarrow \#0q_210\# \rightarrow$$
$$\#010q_1\# \qquad \#010q_1\# \qquad \#010q_1\# \qquad \#01q_20\# \qquad \#0q_21\#\#$$

$$\#01q_20\# \rightarrow \#010q_2\# \rightarrow \#010q_2\# \rightarrow \text{halts and suceeds}$$
$$\#q_20\#\#\# \qquad q_2\#\#\#\#\# \qquad \#q\#\#\#\#$$

For a n-tape Turing machine M and input x, the time required by M to compute input x is simply the number of steps to halting.

If M(x) = undecided, then the time required by M on x is thought to be ∞.

We say that machine M operates within time f(n) if for any input string x the time required by M on x is at most f(|x|).

The function f(n) is a time bound.

Suppose $L \subset \Sigma^*$ is decided by a multi-tape Turing machine operating in time f(n) we say that $L \in$ TIME(f(n)), that is, TIME(f(n)) is a set of languages and contains those languages that can be decided by a Turing machine with multiple tapes operating within the time bound f(n).

TIME(f(n)) is called a complexity class and is a set of languages and all these languages can be decided within some specified bound on some aspect of their performance (e.g. time, space).

Exercise 6.3 Compare the time (number of steps) taken by the 1-tape and 2-tape Turing machines which recognise palindromes.

Reading: (Papadimitriou, chapter 2), (Boolos, chapter 3), (Lewis&Papadimitriou, chapter 4)