UNIT-VII

TURING MACHINES

Introduction

Until now we have been interested primarily in simple classes of languages and the ways that they can be used for relatively constrainted problems, such as analyzing protocols, searching text, or parsing programs.

We begin with an informal argument, using an assumed knowledge of C programming, to show that there are specific problems we cannot solve using a computer.

These problems are called "undecidable". We then introduce a respected formalism for computers called the Turing machine. While a Turing machine looks nothing like a PC, and would be grossly inefficient should some startup company decide to manufacture and sell them, the Turing machine long has been recognized as an accurate model for what any physical computing device is capable of doing.

TURING MACHINE:

- The purpose of the theory of undecidable problems is not only to establish the existence of such problems – an intellectually exciting idea in its own right – but to provide guidance to programmers about what they might or might not be able to accomplish through programming.
- The theory has great impact when we discuss the problems that are decidable, may require large amount of time to solve them. These problems are called "intractable problems" tend to present greater difficulty to the programmer and system designer than do the undecidable problems.
- We need tools that will allow us to prove everyday questions undecidable or intractable.
- *As a result we need to rebuild our theory of undecidability, based not on programs in C or another language, but based on a very simple model of a computer called the* ***Turing machine****.*
- *This device is essentially a finite automaton that has a single tape of infinite length on which it may read and write data.*
- *One advantage of the Turing machine over programs as representation of what can be computed is that the Turing machine is sufficiently simple that we can represent its configuration precisely, using simple notation much like the ID's of a PDA.*

NOTATION or MODEL OF THE TURING MACHINE

- We may visualize a Turing machine as in figure-1 given as follows. The machine consists of a *finite control* which can be in any of a finite set of states.

- There is a *tape* divided into squares or *cells*; each cell can hold any one of a finite number of symbols.

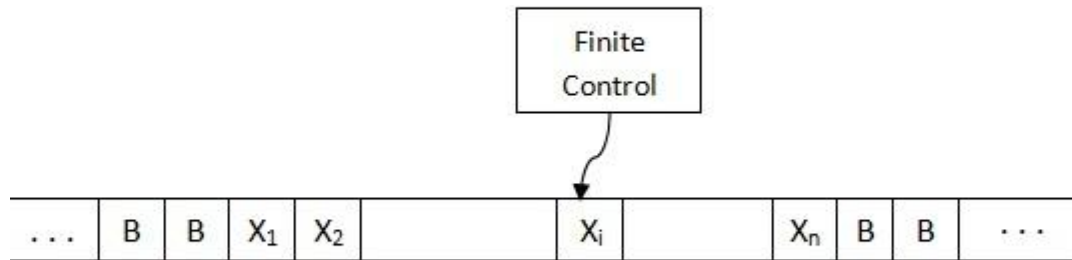

Figure 1: A Turing Machine

- Initially the *input*, which is a finite length string of symbols chosen from the *input alphabet*, is placed on the tape.
- All other tape cells, extending infinitely to the left and right, initially hold a special symbol called the *blank*.
- The blank is a *tape symbol*, but not an input symbol, and there may be other tape symbols besides the input symbols and the blank as well.
- There is a *tape head* that is always positioned at one of the tape cells. The Turing machine is said to be *scanning* that cell.
- Initially the tape head is at the left most cells that hold the input.
- A *move* of the Turing machine is a function of the state of the finite control and the tape symbol scanned.
- In one move, the Turing machine will:
    1. *Change state*. The next state optionally may be the same as the current state.
    2. *Write a tape symbol in the cell scanned*. This tape symbol replaces whatever symbol was in that cell. Optionally the symbol written may be the same as the symbol currently there.
    3. *Move the tape head left or right.* In our formalism we require a move, and do not allow the head to remain stationary. This restriction does not constrain what a truing machine can compute, since any sequence of moves with a stationary head could be condensed, along with the next tape head move, into a single state change, a new tape symbol, and a move left or right.

    ✓ The formal notation we shall use for a Turing Machine (TM) is similar to that used for finite automata or PDA's. We describe a TM by the 7 – tuple like

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

    ✓ Whose components have the following meanings:

    Q: the finite set of states of the finite control

    Σ: the finite set of input symbols

Γ: the complete set of tape symbols; Σ is always a subset of Γ.

δ: The transition function. **δ = Q / F x Γ → Q x Γ x {L, R}**

The arguments of δ(q, X) are a state q and a tape symbol X. The value of δ (q, X), if it is defined, is a triple (p, Y, D), where:

- p is the next state, in Q.
- Y is the symbol, in Γ, written in the cell being scanned, replacing whatever symbol was there.
- D is a direction, either L or R, standing for “left” or “right”, respectively, and telling us the direction in which the head moves.

q_0: the start state, a member of Q, in which the finite control is found initially.

B: the blank symbol. This symbol is in Γ but not in Σ; i.e. it is not an input symbol. The blank appears initially in all but the finite number of initial cells that hold input symbols.

F: the set of final or accepting states, a subset of Q.

INSTANTANEOUS DESCRIPTION FOR TURING MACHINE

- Since a TM in principle has an infinitely long tape, we might imagine that it is impossible to describe the configurations of a TM briefly.
- However after any finite number of moves, the TM can have visited only a finite number of cells, even though the number of cells visited can eventually grow beyond any finite limit.
- *Thus in every ID, there is an infinite prefix and an infinite suffix of that cells that have never been visited.*
- *These cells must all hold either blanks or one of the finite numbers of input symbols.* We thus show in an ID only the cells between the leftmost and rightmost nonblanks.
- *Under a special condition, when the head is scanning one of the leading or trailing blanks, a finite number of blanks to the left or right of the nonblank portion of the tape must also be included in the ID.*
- *In addition to representing the tape, we must represent the finite control and the tape-head position. To do so we embed the state in the tape, and place it immediately to the left of the cell scanned.*
- However it is easy to change the names of the states so they have nothing in common with the tape symbols, since the operation of the TM does not depend on what the states are called.
- Thus we shall use the string $X_1X_2 \ldots X_{i-1}qX_iX_{i+1} \ldots X_n$ to represent the ID in which
  1. q is the state of the Turing machine

2. The tape head is scanning the ith symbol from the left.
3. $X_1X_2 \ldots X_n$ is the portion of the tape between the leftmost and the rightmost nonblank. As an exception if the head is to the left of the leftmost nonblank or the right of the rightmost nonblank, then some prefix or suffix of $X_1X_2 \ldots X_n$ will be blank, and i will be 1 or n, respectively.

- We describe the Turing machine M = (Q, Σ, Γ, δ, q_0, B, F) by the $\vdash_M$ notation that was used for PDA's. When the Turing machine M is understood, we shall use just ⊢ to reflect moves.
- Suppose $\delta(q, X_i) = (p, Y, L)$; i.e. the next move is leftward. Then

$$X_1 X_2 \ldots X_{i-1}\, q\, X_i X_{i+1} \ldots X_n \vdash_M X_1 X_2 \ldots X_{i-2}\, p\, X_{i-1} Y X_{i+1} \ldots X_n$$

- Notice how this move reflects the change to state p and the fact that the tape head is now positioned at cell i-1. There are two important exceptions:
    1. If i = 1, then M moves the blank to the left of X_1. In that case,

    $$X_1X_2 \ldots X_n \vdash_M pBYX_2 \ldots X_n$$

    2. If i = n and Y = B then the symbol B written over X_n joins the infinite sequence of trailing blanks and does not appear in the next ID. Thus,

    $$X_1X_2 \ldots X_{n-1}\, q\, X_n \vdash_M X_1X_2 \ldots X_{n-2}\, p\, X_{n-1}$$

- Now suppose $\delta(q, X_i) = (p, Y, R)$; i.e. the next move is rightward. Then

$$X_1 X_2 \ldots X_{i-1}\, q\, X_i X_{i+1} \ldots X_n \vdash_M X_1 X_2 \ldots X_{i-1} Y\, p\, X_{i+1} \ldots X_n$$

- Here the move reflects the fact that the head has moved to cell i+1. Again there are two important exceptions:
    1. If i = n, then i+1st cell holds a blank, and that cell was not part of the previous ID. Thus we instead have

    $$X_1 X_2 \ldots X_{n-1}\, q\, X_n \vdash_M X_1 X_2 \ldots X_{n-1} Y\, p\, B$$

    2. If i = 1 and Y =B , then the symbol B written over X_1 joins the infinite sequence of leading blanks and does not appear in the next ID. Thus,

    $$q\, X_1 X_2 \ldots X_n \vdash_M p\, X_2 \ldots X_n$$

Example- 7.1:

- Let us design a TM and see how it behaves on a typical input. The TM we construct will accept the language consisting of all palindromes of 0's and 1's.
- Initially it is given a finite sequence of 0's and 1's on its tape, preceded and followed by infinity of blanks.
- Starting at the left end it checks the first symbol of the input if it is a 0 and changes it to X. Similarly if it is a 1, it changes it to Y and moves the right until it sees a blank.
- Now it moves left and checks whether the symbol read matches the one most recently changed. If so it is also changed correspondingly and the machine move back left until it finds the leftmost 0 or 1.

- This process is continued by moving left and right alternately until all 0's and 1's have been matched.
- If the input is not a palindrome, the TM will not have a next move and hence will halt without accepting. However if it is a palindrome it accepts.
- The formal specification of TM M is
  $M = (\{q_0, q_1, q_2, q_3, q_4, q_5, q_6\}, \{0, 1\}, \{0, 1, X, Y, B\}, \delta, q_0, B, \{q_6\})$
- Where δ is given by the table as follows:

State	0	1	X	Y	B
q_0	(q_1, X, R)	(q_2, Y, R)	(q_6, X, R)	(q_6, Y, R)	(q_6, B, R)
q_1	$(q_1, 0, R)$	$(q_1, 1, R)$	(q_3, X, L)	(q_3, Y, L)	(q_3, B, L)
q_2	$(q_2, 0, R)$	$(q_2, 1, R)$	(q_4, X, L)	(q_4, Y, L)	(q_4, B, L)
q_3	(q_5, X, L)	-	(q_6, X, R)	(q_6, Y, R)	-
q_4	-	(q_5, Y, L)	(q_6, X, R)	(q_6, Y, R)	-
q_5	$(q_5, 0, L)$	$(q_5, 1, L)$	(q_0, X, R)	(q_0, Y, R)	-

Figure 2: A Turing machine to accept the set of all palindromes over {0, 1}*

- As M performs its computation, the portion of the tape, where M's tape head has visited, will always be a sequence of symbols described by the regular expression(X + Y)*(0 + 1)*(X + Y)*.
- That is there will be some 0's that have been changed to X's and 1's that have been changed to Y's followed by 0's that have not yet been changed to X's and 1's that have not yet been changed to Y's followed again with a string of X's and Y's.
- State q_0 is the initial state, and M also enters the state q_0 every time it returns to the leftmost remaining 0 or 1. Accordingly we have two cases:
  1. If M is in state q_0 and it scans a 0, the rule in the upper-left corner of figure-2 tells it to go to state q_1, change the 0 to an X, and move right over all 0's and 1's, that it finds on the tape remaining in the same state until it sees an X, Y or B. Now M moves one step to the left and changes to state q_3. It verifies that the symbol read is 0 and changes the 0 to an X and goes to state q_5.
  2. If M is in state q_0 and it scans a 1 it goes to state q_2 changes the 1 to a Y and moves right over all 0's and 1's, that it finds on the tape remaining in the same state until it sees an X, Y or B. Now M moves one step to the left and changes to state q_4. It then verifies that the symbol read is a 1 and changing it to a Y goes to state q_5.
- In state q_5 M moves left over 0's and 1's until it finds an X or Y. Now it changes state to q_0 once again and moves right. Once again there are two cases:
  1. If M now sees 0's or 1's, it repeats the matching cycle we have just described.

2. If M sees an X or a Y, then it has changed all the 0's to X's and 1's to Y's; the input was of a palindrome of even length, and hence M should accept. Thus M enters state q_6 and halts

- In case M is in state q_3 and reads an X or q_4 and reads a Y instead of 0 and 1, it concludes that the input was a palindrome of odd length. In this case it changes to state q_6 and halts.
- On the other hand if M encounters a 1 in state q_3 or 0 in state q_4, then the input is not a palindrome and so M halts without accepting.
- Here is an example of an accepting computation by M. Its input is 1001. Initially M is in state q_0, scanning the first 1 i.e. M's initial ID is $q_0$1001.the entire sequence of moves of M is :
  $q_0 1001 \vdash Yq_2 001 \vdash Y0q_2 01 \vdash Y00q_2 1 \vdash Y001q_2 \vdash Y00q_4 1 \vdash Y00q_5 Y \vdash Y0q_5 0Y \vdash Yq_5 00Y \vdash q_5 Y00Y \vdash Yq_0 00Y \vdash YXq_1 0Y \vdash YX0q_1 Y \vdash YXq_3 0Y \vdash YXq_5 XY \vdash Yq_5 XXY \vdash YXq_0 XY \vdash Yq_6 XXY$

DESIGN OF TURING MACHINE

TRANSITION DIAGRAM OF TURING MACHINE

- We can represent the transitions of a Turing machine pictorially much as we did for the PDA.
- A transition diagram consists of a set of nodes corresponding to the states of the TM. An arc from state q to state p is labeled by one or more items of the form X/Y D, where X and Y are tape symbols and D is a direction either L or R.
- That is whenever δ(q, X) = (p, Y, D), we find the label X/Y D on the arc form q to p. However in our diagrams the direction D is represented pictorially by ← for "left" and → for "right".
- As for other kinds of transition diagram we represent diagrams, we represent the start state by the word "Start" and an arrow entering that state.
- Thus the only information about the TM one cannot read directly from the diagram is the symbol used for the blank. We shall assume that symbol is B unless we state otherwise.

  Example-7.2:
- The following Figure – 3 shows the transition diagram for the Turing machine of example-7.1 whose transition function was given in figure-2.

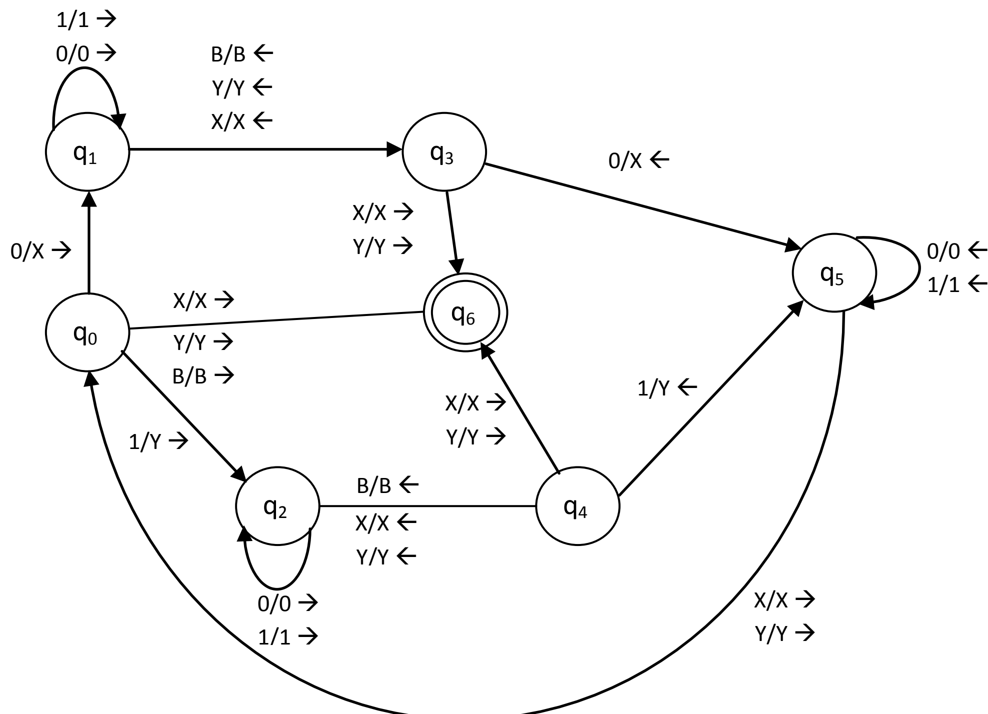

Figure 3: Transition diagram for the Turing Machine which accepts the set of all palindromes over {0, 1}

COMPUTABLE FUNCTIONS

- Computable functions are the basic objects of study in computability theory.
- Computable functions are the formalized analogue of the intuitive notion of algorithm.
- They are used to discuss computability without referring to any concrete model of computation such as Turing machines or register machines.
- Any definition, however, must make reference to some specific model of computation but all valid definitions yield the same class of functions.
- Particular models of computability that give rise to the set of computable functions are the Turing-computable functions and the μ-recursive functions.
- Before the precise definition of computable function, mathematicians often used the informal term effectively calculable. This term has since come to be identified with the computable functions.
- Note that the effective computability of these functions does not imply that they can be efficiently computed (i.e. computed within a reasonable amount of time).
- In fact, for some effectively calculable functions it can be shown that any algorithm that computes them will be very inefficient in the sense that the running time of the algorithm increases exponentially with the length of the input.
- According to the Church–Turing thesis, computable functions are exactly the functions that can be calculated using a mechanical calculation device given unlimited amounts of time and storage space.

- Equivalently, this thesis states that any function which has an algorithm is computable. Note that an algorithm in this sense is understood to be a sequence of steps a person with unlimited time and an infinite supply of pen and paper could follow.
- The basic characteristic of a computable function is that there must be a finite procedure (an algorithm) telling how to compute the function.

COMPUTABLE SETS AND RELATION

- A set **A** of natural numbers is called computable if there is a computable, total function *f* such that for any natural number n, *f*(n) = 1 if n is in A and *f*(n) = 0 if n is not in A.
- A set of natural numbers is called computably enumerable if there is a computable function *f* such that for each number n, *f*(n) is defined if and only if n is in the set.
- Thus a set is computably enumerable if and only if it is the domain of some computable function.
- The word enumerable is used because the following are equivalent for a nonempty subset B of the natural numbers:
  1. B is the domain of a computable function.
  2. B is the range of a total computable function. If B is infinite then the function can be assumed to be injective (or one to one).
- If a set B is the range of a function *f* then the function can be viewed as an enumeration of B, because the list *f*(0), *f*(1), ... will include every element of B.
- In computability theory in computer science, it is common to consider formal languages. An alphabet is an arbitrary set.
- A word on an alphabet is a finite sequence of symbols from the alphabet; the same symbol may be used more than once.
- For example, binary strings are exactly the words on the alphabet {0, 1}.
- A language is a subset of the collection of all words on a fixed alphabet.
- For example, the collection of all binary strings that contain exactly 3 ones is a language over the binary alphabet.
- A key property of a formal language is the level of difficulty required to decide whether a given word is in the language. Some coding system must be developed to allow a computable function to take an arbitrary word in the language as input; this is usually considered routine.

RECURSIVELY ENUMERABLE LANGUAGES

- A language is computably enumerable if there is a computable function *f* such that *f*(w) is defined if and only if the word w is in the language. The term enumerable has the same etymology as in computably enumerable sets of natural numbers.

  Examples
- The following functions are computable:

1. Each function with a finite domain; e.g., any finite sequence of natural numbers.
2. Each constant function $f : N^k \rightarrow N$, $f(n_1,...n_k) := n$.
3. Addition $f : \mathbf{N}^2 \rightarrow \mathbf{N}$, $f(n_1,n_2) := n_1 + n_2$
4. The function which gives the list of prime factors of a number.
5. The greatest common divisor of two numbers is a computable function.

CHURCH HYPOTHESIS

- In 1900 mathematician David Hilbert delivered a lecture, where he identified for every twenty three mathematical problems and posed them as a challenge for the coming century. The tenth problem on his list concerned algorithms.
- According to Hilbert's tenth problem, it was to devise an algorithm that tests whether a polynomial has an integral root.
- He did not use the term algorithm but rather a process according to which it can be determined by a finite number of operations.
- Hilbert explicitly asked that an algorithm be devised. Thus he apparently assumed that such an algorithm must exist – someone need only find it.
- As we now know, no algorithm exists for this task; it is algorithmically unsolvable.
- The definition came in 1936 papers of Alonzo Church and Alan Turing. Church used a notational system called the λ-calculus to define algorithms. Turing did it with his machines.
- These two definitions were shown to be equivalent. This connection between the informal notion of algorithm and the precise definition has come to be called the Church-Turing thesis.
- In computability theory the Church-Turing thesis is a combined hypothesis about the nature of functions whose values are effectively calculable or in more modern terms, functions whose values are algorithmically computable.
- In simple terms the church-Turing thesis states that a function is algorithmically computable if and only if it is computable by a Turing machine.
- The Church–Turing thesis is a statement that characterizes the nature of computation and cannot be formally proven.

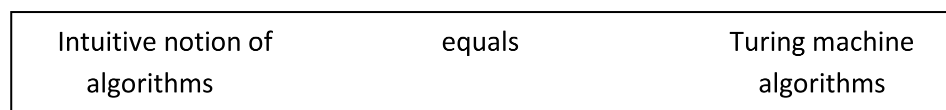

Figure 4: The Church-Turing thesis

COUNTER MACHINES

- A counter machine may be thought of in one of the two ways:
  1. The counter machine has the same structure as the multistack machine but in place of each stack is a counter. Counters hold any nonnegative integer, but we

can only distinguish between zero and nonzero counters. That is the move of the counter machine depends on its state, input symbol, and which if any of the counters are zero. In one move the counter machine can:

a. Change state.
b. Add or subtract 1 from any of its counters, independently. However a counter is not allowed to become negative, so it cannot subtract 1 from a counter that is currently 0.

2. A counter machine may also be regarded as a restricted multistack machine. The restrictions are as follows:
   a. There are only two stack symbols, which we shall refer to Z_0 (the bottom of the stack marker) and X.
   b. Z_0 is initially on each stack.
   c. We may replace Z_0 only by a string of the form X^iZ_0 for some $i \geq 0$.
   d. We may replace X only by X^i for some $i \geq 0$. That is Z_0 appears only on the bottom of each stack, and all other stack symbols, if any are X.

THE POWER OF COUNTER MACHINES

- There are a few observations about the languages accepted by counter machines that are obvious but worth stating:
  1. Every language accepted by a counter machine is recursively enumerable. The reason is that a counter machine is a special case of a stack machine, and a stack machine is a special case of a Multitape Turing machine which accepts only recursively enumerable languages.
  2. Every languages accepted by a one-counter machine is a CFL. In fact the languages of one-counter machines are accepted by deterministic PDA's, although the proof is surprisingly complex.
- The difficulty in the proof stems from the fact that the multistack and counter machines have an end marker $ at the end of their input.

TYPES OF TURING MACHINES

- There are different types of Turing machines
  1. Two way infinite tape
  2. Multitape Turing machines
  3. Non deterministic Turing machines
  4. Two-dimensional Turing machines
  5. Multihead Turing machines

TWO WAY INFINITE TAPE TURING MACHINE

- It is a Turing machine with its input tape infinite in both the directions, the other components being the same as that of the basic model.

MULTIHEAD TURING MACHINE

- It is a single tape Turing machine having k heads reading symbols on the same tape. In one step all the heads sense the scanned symbols and move or write independently.
- A Multihead Turing machine can be simulated by single head Turing machine.

TWODIMENSIONAL TURING MACHINES

- The Turing machine can have two dimensional tapes.
- When the head is scanning the symbol it can move either left or right or up or down. The smallest rectangle containing the non blank portion is m x n, then it has m rows and n columns.
- A one dimensional tape Turing machine which tries to simulate this 2 dimensional Turing machine will have 2 tapes.