From: Dr shreedhara K S, Professor in CSE, UBDT college of engineering, Davanagere

Automata Theory and Computability

Module 5

The model of Linear Bounded automata: Decidability: Definition of an algorithm, decidability, decidable languages, Undecidable languages, halting problem of TM, Post correspondence problem. Complexity: Growth rate of functions, the classes of P and NP, Quantum Computation: quantum computers, Church-Turing thesis.

- A word automata is a plural of word "automation", which means to automate or mechanize. Mechanization of a process means performing it on a machine without human intervention.
- The basic aim of Computer Science is to design Computing Machine (CM).
- To design Computing Machine for a problem it is necessary to ensure that the problem is solvable and computable.
- If it is not solvable in a reasonable amount of time, it is solvable in principle only
- As a student of Computer Science , we should know what is computable, and if it is computable, how it can be implemented on a machine.
- Aim of automata theory is to draw a boundary between what is computable and what is not, if computation is performed on a machine,
  Machine may be of two types
  1. problem specific dedicated machine
  2. Generic machine.

Church-Turing thesis-1936

- Any algorithmic procedure that can be carried out by a human or a computer, can also be carried out by a Turing machine.
- Now it is universally accepted by computer scientists that TM is a Mathematical model of an algorithm.
- TM has an algorithm and an algorithm has a TM. If there is an algorithm problem is decidable, TM solves that problem
- The statement of the thesis –

" Every function which would naturally be regarded as computable can be computed by a Turing machine"

Implies

- Any mechanical computation can be performed by a TM
- For every computable problem there is a TM
- If there is no TM that decides P there is no algorithm that can solve problem P.
- In our general life, we have several problems and some of these have solutions, but some have not, we simply say a problem is decidable if there is a solution otherwise undecidable.

example:

- Does Sun rises in the East? YES
- Will tomorrow be a rainy day ? (YES/NO ?)

Decidable and Undecidable Languages

- A problem is said to be decidable if its language is recursive OR it has solution.

Example:

Decidable :

-Does FSM accept regular language?

- is the power of NFA and DFA same

Undecidable:

- For a given CFG is L(G) ambiguous?

L is *Turing decidable* (or just decidable) if there exists a Turing machine M that accepts all strings in L and rejects all strings not in L. Note that by rejection means that the machine halts after a finite number of steps and announces that the input string is not acceptable.

- There are two types of TMs (based on halting):

1. **(*Recursive*)**
   TMs that *always* halt, no matter accepting or non-accepting ≡ **DECIDABLE** PROBLEMS
2. **(*Recursively enumerable*)**
   TMs that *are guaranteed to halt* only on acceptance.
   If non-accepting, it may or may not halt (i.e., could loop forever).
   - Undecidable problems are those that are not recursive

Recursive languages

A Language L over the alphabet∑ is called recursive if there is a TM M that accepts every word in L and rejects every word in L'

Accept (M)=L

Reject(M)=L'

loop(M)=ø

Example: b(a+b)*

M is a *Turing Machine* and L is a *recursive language* that M accepts,
if a string w ∈ L then M *halts in a final state* and
if w ∉ L then M *halts in a non-final state*

Recursively Enumerable Language:

A Language L over the alphabet∑ is called recursively enumerable if there is a TM M that accepts every word in L and either rejects or loops every word in L' the complement of L

Accept (M)=L

Reject(M) +Loop(M)= L'

Example: (a+b)*bb(a+b)*

M is a *Turing Machine* and L is a *recursively enumerable language* that M accepts,
if a string w ∈ L then M *halts in a final state* and
if w ∉ L then M *halts in a non-final state or loops forever*

Theorem: If L is recursive language, so is $\overline{L}$

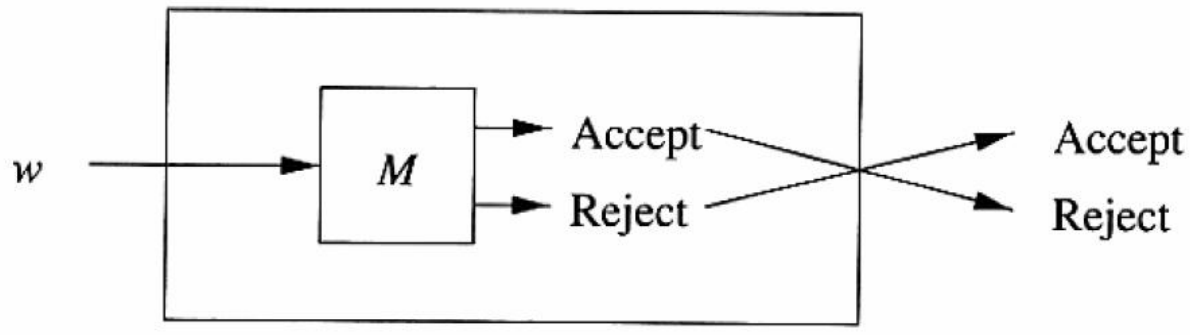
PROOF: Let $L = L(M)$ for some TM M that always halts. We construct a TM $\overline{M}$ such that $\overline{L} = L(\overline{M})$ by the construction suggested in Fig. 10. That is, $\overline{M}$ behaves just like M. However, M is modified as follows to create $\overline{M}$:

1. The accepting states of M are made nonaccepting states of $\overline{M}$ with no transitions; i.e., in these states $\overline{M}$ will halt without accepting.

2. $\overline{M}$ has a new accepting state r; there are no transitions from r.

3. For each combination of a nonaccepting state of M and a tape symbol of M such that M has no transition (i.e., M halts without accepting), add a transition to the accepting state r.

Since M is guaranteed to halt, we know that $\overline{M}$ is also guaranteed to halt. Moreover, $\overline{M}$ accepts exactly those strings that M does not accept. Thus $\overline{M}$ accepts $\overline{L}$. □



Recursively Enumerable Languages closed under complementation? (NO)

1. **Prove that Recursive Languagess are closed under Union**
   - Let M_u = TM for $L_1 \cup L_2$
   - M_u construction:
     1. Make 2-tapes and copy input w on both tapes
     2. Simulate M_1 on tape 1
     3. Simulate M_2 on tape 2
     4. If either M_1 or M_2 accepts, then M_u accepts
     5. Otherwise, M_u rejects.

2. **Prove that Recursive Languages are closed under Intersection**
   - Let M_n = TM for $L_1 \cap L_2$
   - M_n construction:
     1. Make 2-tapes and copy input w on both tapes
     2. Simulate M_1 on tape 1
     3. Simulate M_2 on tape 2
     4. If M_1 AND M_2 accepts, then M_n accepts
     5. Otherwise, M_n rejects.

3. Recursive languages are also closed under:
   a. Concatenation
   b. Kleene closure (star operator)
   c. Homomorphism, and inverse homomorphism
4. RE languages are closed under:
   a. Union, intersection, concatenation, Kleene closure
5. RE languages are *not* closed under:
   a. Complementation

1. **Decidable Languages about DFA : Prove that**

A_{DFA} is a decidable language.

$$A_{\text{DFA}} = \{\langle B, w\rangle \mid B \text{ is a DFA that accepts input string } w\}.$$

M = "On input $\langle B, w\rangle$, where B is a DFA and w is a string:

1. Simulate B on input w.
2. If the simulation ends in an accept state, *accept*. If it ends in a nonaccepting state, *reject*."

Proof: To prove we construct a TM that halts and also accept A_{DFA}
Define TM as

1. let B be a DFA and w input string (B,w) as input for TM M
2. Simulate B and input w in TM M
3. if the simulation ends in an accepting state of B then M accepts w. if it ends in non accepting state of B then M rejects w.

2. Prove that A_{NFA} is a decidable language.

$$A_{NFA} = \{\langle B, w\rangle | \ B \text{ is an NFA that accepts input string } w\}.$$

N = "On input $\langle B, w\rangle$ where B is an NFA, and w is a string:

1. Convert NFA B to an equivalent DFA C, using the **last** procedure .
2. Run TM M on input $\langle C, w\rangle$.
3. If M accepts, *accept*; otherwise, *reject*."

3. Prove that A_{REX} is a decidable language.

$$A_{REX} = \{\langle R, w\rangle | \ R \text{ is a regular expression that generates string } w\}.$$

P = "On input $\langle R, w\rangle$ where R is a regular expression and w is a string:

1. Convert regular expression R to an equivalent DFA A
2. Run TM N on input $\langle A, w\rangle$.
3. If N accepts, *accept*; if N rejects, *reject*."

Halting and Acceptance Problems:

$$A_{TM} = \{\langle M, w\rangle | \ M \text{ is a TM and } M \text{ accepts } w\}.$$

Acceptance Problem:
Does a Turing machine accept an input string?

1. A_{TM} is recursively enumerable.

Simulate M on w. if M enters an accepting state, We prove by contradiction. We assume **A_{TM} is decidable by a TM H that eventually halts on all input, then**

$$H(\langle M, w\rangle) = \begin{cases} accept & \text{if } M \text{ accepts } w \\ reject & \text{if } M \text{ does not accept } w. \end{cases}$$

$D =$ "On input $\langle M \rangle$, where M is a TM:

1. Run H on input $\langle M, \langle M \rangle\rangle$.
2. Output the opposite of what H outputs; that is, if H accepts, *reject* and if H rejects, *accept*."

$$D(\langle M \rangle) = \begin{cases} accept & \text{if } M \text{ does not accept } \langle M \rangle \\ reject & \text{if } M \text{ accepts } \langle M \rangle. \end{cases}$$

$$D(\langle D \rangle) = \begin{cases} accept & \text{if } D \text{ does not accept } \langle D \rangle \\ reject & \text{if } D \text{ accepts } \langle D \rangle. \end{cases}$$

We construct new TM D with H as a subroutine. D calls H to determine what M does when it receive the input <M>. Based on the received information on (M,<M>), D rejects M if M accepts <M> and accepts M if M rejects <M>.

D described as follows:
1. <M> is an input to D
2. D calls H to run on (M,<M>)
3. D rejects <M> if H accepts (M, <M>) and accepts <M>if H rejects (M, <M>)

This means D accepts <D> if D does not accept <D>, which is a contradiction. Hence **A_{TM}** is Undecidable.

The Post Correspondence Problem

PCP is a combinatorial problem formulated by Emil Post in 1946. This problem has many applications in the field theory of formal languages. A correspondence system P is a finite set of ordered pairs of non empty strings over some alphabet. Let $A = w_1, w_2, \ldots, w_n$ $B = v_1, v_2, \ldots, v_n$

There is a **P**ost **C**orrespondence Solution if there is a sequence $i, j, \ldots, k$ such that:

$$w_i w_j \cdots w_k = v_i v_j \cdots v_k$$

Index	w_j	V_i
1	100	001
2	11	111
3	111	11

Let W = $w_2w_1w_3=v_2v_1v_3$ = 11100111 we have got a solution. But we may not get solution always for various other combinations and strings of different length. Hence PCP is undecidable.

The **M**odified **P**ost **C**orrespondence Problem

$A = w_1, w_2, \ldots, w_n$ $B = v_1, v_2, \ldots, v_n$

$$1, i, j, \ldots, k$$

$$w_1 w_i w_j \cdots w_k = v_1 v_i v_j \cdots v_k$$

If the index start with 1 and then any other sequence then it is called MPCP

Algorithm: An algorithm is "a finite set of precise instructions for performing a computation or for solving a problem"

- A program is one type of algorithm
  - All programs are algorithms
  - Not all algorithms are programs!
- The steps to compute roots of quadratic equation is an algorithm
- The steps to compute the cosine of 90° is an algorithm

Algorithms generally share a set of properties:

- Input: what the algorithm takes in as input
- Output: what the algorithm produces as output
- Definiteness: the steps are defined precisely
- Correctness: should produce the correct output
- Finiteness: the steps required should be finite
- Effectiveness: each step must be able to be performed in a finite amount of time

- Generality: the algorithm *should* be applicable to all problems of a similar form

Comparing Algorithms (While comparing two algorithm we use time and space complexities)

- Time complexity
  - The amount of time that an algorithm needs to run to completion
- Space complexity
  - The amount of memory an algorithm needs to run
- To analyze running time of the algorithm we use following cases
  - Best case
  - Worst case

    Average case

Asymptotic analysis

- The big-Oh notation is used widely to characterize running times and space bounds
- The big-Oh notation allows us to ignore constant factors and lower order terms and focus on the main components of a function which affect its growth
- Given functions $f(n)$ and $g(n)$, we say that $f(n)$ is $O(g(n))$ if there are positive constants c and n_0 such that

$$f(n) \leq cg(n) \text{ for } n \geq n_0$$

- Example: $2n + 10$ is $O(n)$
  - $2n + 10 \leq cn$
  - $(c - 2)\, n \geq 10$
  - $n \geq 10/(c - 2)$

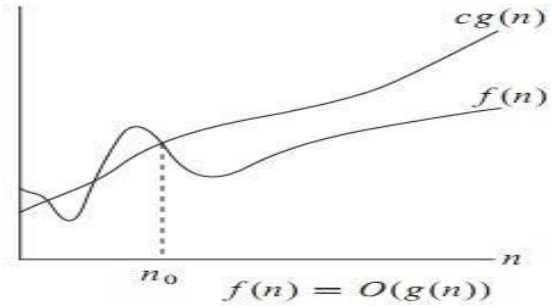
  It is true for $c = 3$ and $n_0 = 10$

- $7n-2$ is $O(n)$

  need $c > 0$ and $n_0 \geq 1$ such that $7n-2 \leq c \bullet n$ for $n \geq n_0$

  this is true for $c = 7$ and $n_0 = 1$

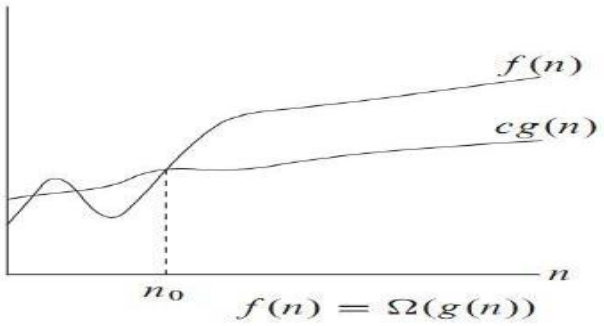
f(n)=O(g(n)) *iff* there exist positive constants c and n0 such that f(n) ≤ cg(n) for all n ≥ n0
O-notation to give an upper bound on a function



Big oh provides an asymptotic upper bound on a function.

Omega provides an asymptotic lower bound on a function.

$$\Omega(g(n)) = \{f(n) : \text{there exist positive constants } c \text{ and } n_0 \text{ such that } 0 \le cg(n) \le f(n) \text{ for all } n \ge n_0\}.$$



- The big-Oh notation gives an upper bound on the growth rate of a function
- The statement "***f***(***n***) is ***O***(***g***(***n***))" means that the growth rate of ***f***(***n***) is no more than the growth rate of ***g***(***n***)
- We can use the big-Oh notation to rank functions according to their growth rate

$$f(n) = a_0 + a_1 n + a_2 n^2 + \ldots + a_d n^d$$

- If is ***f***(***n***) a polynomial of degree ***d***, then ***f***(***n***) is ***O***(n^d), i.e.,
    1. Drop lower-order terms
    2. Drop constant factors
- Use the smallest possible class of functions
    1. Say "2***n*** is ***O***(***n***)" instead of "2***n*** is ***O***(n^2)"
- Use the simplest expression of the class
    Say "3***n*** + 5 is ***O***(***n***)" instead of "3***n*** + 5 is ***O***(3***n***)"
- Following are the terms usually used in algorithm analysis:
    1. Constant ≈ ***1***
    2. Logarithmic ≈ log ***n***
    3. Linear ≈ ***n***
    4. N-Log-N ≈ ***n*** log ***n***
    5. Quadratic ≈ n^2
    6. Cubic ≈ n^3
    7. Exponential ≈ 2^n

Class P Problems:

P stands for deterministic polynomial time. A deterministic machine at each time executes an instruction. Depending on instruction, it then goes to next state which is unique. Hence time complexity of DTM is the maximum number of moves made by M is processing any input string of length n, taken over all input of length n.

- The class P consists of those problems that are solvable in polynomial time.
- More specifically, they are problems that can be solved in time $O(n^k)$ for some constant k, where n is the size of the input to the problem
- The key is that n is the **size of input**

Def: A language L is said to be in class P if there exists a DTM M such that M is of time complexity P(n) for some polynomial P and M accepts L.

Class NP Problems

Def: A language L is in class NP if there is a nondeterministic TM such that M is of time complexity P(n) for some polynomial P and M accepts L.

- **NP is not the same as non-polynomial complexity/running time. NP does not stand for not polynomial.**
- **NP = Non-Deterministic polynomial time**
- NP means verifiable in polynomial time
- Verifiable?
    - If we are somehow given a 'certificate' of a solution we can verify the legitimacy in polynomial time

- Problem is in NP iff it is decidable by some non deterministic Turing machine in polynomial time.
- It is provable that a Non Deterministic Turing Machine is equivalent to a Deterministic Turing Machine
- Remember NFA to DFA conversion?
    - Given an NFA with n states how many states does the equivalent DFA have?
    - Worst case …. 2^n
    - The deterministic version of a polynomial time
- non deterministic Turing machine will run in exponential time (worst case)
- Since it takes polynomial time to run the program, just run the program and get a solution
- But is NP a subset of P? It is not yet clear whether P = NP or not

Quantum Computers

- **Computers are physical objects, and computations are physical processes. What computers can or cannot compute is determined by the law of physics alone, and not by pure mathematics. Computation with coherent atomic-scale dynamics.**
  The behavior of a quantum computer is governed by the laws of quantum mechanics.

- In 1982 Richard Feynmann, a Nobel laurite in physics suggested to build computer based on quantum mechanics.
- Quantum mechanics arose in the early 1920s, when classical physics could not explain everything.
- QM will provide tools to fill up the gulf between the small and the relatively complex systems in physics.
- Bit (0 or 1) is the fundamental concept of classical computation and information. Classical computer built from electronic circuits containing wires and gates.
- Quantum bit and quantum circuits which are analogous to bits and circuits. Two possible states of a qubit (Dirac)are $|0\rangle$ $|1\rangle$
- Quantum bit is qubit described mathematically (where (alpha) is complex number)

$$\alpha_0|0\rangle + \alpha_1|1\rangle$$

- Qubit can be in infinite number of state other than dirac |0> or |1>

- The operations are induced by the apparatus *linearly*, that is, if

$$|0\rangle \to \frac{i}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle \qquad |1\rangle \to \frac{1}{\sqrt{2}}|0\rangle + \frac{i}{\sqrt{2}}|1\rangle$$

Then

$$\alpha_0|0\rangle+\alpha_1|1\rangle \to \alpha_0\left(\frac{i}{\sqrt{2}}|0\rangle+\frac{1}{\sqrt{2}}|1\rangle\right)+\alpha_1\left(\frac{1}{\sqrt{2}}|0\rangle+\frac{i}{\sqrt{2}}|1\rangle\right) \quad =\left(\alpha_0\frac{i}{\sqrt{2}}+\alpha_1\frac{1}{\sqrt{2}}\right)|0\rangle+\left(\alpha_0\frac{1}{\sqrt{2}}+\alpha_1\frac{i}{\sqrt{2}}\right)|1\rangle$$

Any linear operation that takes states $\alpha_0|0\rangle+\alpha_1|1\rangle$ satisfying and maps them to be UNITARY

i.e. $|\alpha_0|^2+|\alpha_1|^2=1$

Linear Algebra: $|0\rangle$ Corresponds to $\begin{pmatrix}1\\0\end{pmatrix}$

$|1\rangle$ Corresponds to $\begin{pmatrix}0\\1\end{pmatrix}$

$\alpha_0|0\rangle+\alpha_1|1\rangle$ Corresponds to $\alpha_0\begin{pmatrix}1\\0\end{pmatrix}+\alpha_1\begin{pmatrix}0\\1\end{pmatrix}=\begin{pmatrix}\alpha_0\\\alpha_1\end{pmatrix}$

If we concatenate two qubits

$(\alpha_0|0\rangle+\alpha_1|1\rangle)$ $(\beta_0|0\rangle+\beta_1|1\rangle)$

we have a 2-qubit system with 4 basis states

$|0\rangle|0\rangle=|00\rangle$ $|0\rangle|1\rangle=|01\rangle$ $|1\rangle|0\rangle=|10\rangle$ $|1\rangle|1\rangle=|11\rangle$

And it is describes the state as $\alpha_0\beta_0|00\rangle+\alpha_0\beta_1|01\rangle+\alpha_1\beta_0|10\rangle+\alpha_1\beta_1|11\rangle$

- Quantum computer is a system built from quantum circuits, containing wires and elementary quantum gates, to carry out manipulation of quantum information.

Variants of Turing Machines

Various types of TM are

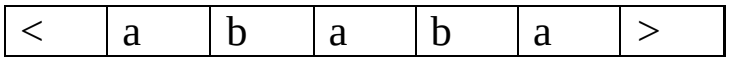
1. **With Multiple tapes**
2. **With one tape but multiple heads**
3. **With two dimensional tapes**
4. **Non deterministic TM**

1. **Multiple tapes: It consists of finite control with k tape heads and k tapes each tape is infinite in both directions. On a single move depending on the state of the finite control and symbol scanned by each of the tape head the machine can change state Or print new symbol on each of cell scanned etc..**

2. **With One tape but Multiple heads: a K head TM has fixed k number of heads and move of TM depends on the state and the symbol scanned by each head. (head can move left, right or stationary).**
3. **Multidimensional TM: It has finite control but the tape consists of a K-dimensional array of cells infinite in all 2 k directions. Depending on the state and symbol scanned , the device changes the state, prints a new symbol, and moves its tape head in one of the 2 k directions, either positively of negatively along one of the k axes.**
4. **Non deterministic TM: In the TM for a given state and tape symbol scanned by the tape head, the machine has a finite number of choices for the next move. Each choice consists of new state, a tape symbol to print and direction of head motion.**

Linear Bounded Automata

LBA is a restricted form of a Non deterministic Turing machine. It is a multitrack turing machine which has only one tape and this tape is exactly same length as that of input. It accepts the string in the similar manner as that of TM. For LBA halting means accepting. In LBA computation is restricted to an area bounded by length of the input. This is very much similar to programming environment where size of the variable is bounded by its data type. Lba is 7-tuple on Deterministic TM with



Left EM **Right EM**

$$M= (Q,\sum,\Gamma, Delta, q_{accept}, q_{reject}, q0)$$

- Two extra symbols < and > are used left end marker and right end marker.
- Input lies between these markers