Variants of TURING MACHINE

Module 5

Variants of TM

1. Multitape TM: TM with more than one tape

2. Non deterministic TM: TM where

$\delta(q, a) = \{(p_1, y_1, D_1), (p_2, y_2, D_2), \cdots\cdots (p_r, y_r, D_r)\}$

1. Multitape TM

- A multitape TM has :
- a finite set Q of states.
- an initial state q_0.
- a subset F of Q called the set of final states.
- a set P of tape symbols.
- a new symbol b, not in P called the blank symbol.

assume that $\Sigma \subseteq \Gamma$ and $b \notin \Sigma$.)

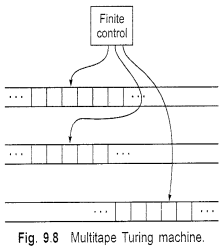


Fig. 9.8 Multitape Turing machine.

- There are *k tapes, each divided into cells. The first tape holds the input* string *w. Initially, all the other tapes hold the blank symbol.*
- Initially the head of the first tape (input tape) is at the left end of the input w. All the other heads can be placed at any cell initially.
- δ is a partial function from Q x Γ^k into Q x Γ^k x $\{L, R, S\}^k$.
- A move depends on the current state and *k tape symbols under k tape heads.*

- In a typical move:
- (i) M enters a new state.
- (ii) On each tape, a new symbol is written in the cell under the head.
- (iii) Each tape head moves to the left or right or remains stationary. The heads move independently: some move to the left, some to the right and the remaining heads do not move.
- The initial ID has the initial state q_0, the input string w in the first tape (input tape), empty strings of b's in the remaining k - 1 tapes. An accepting ID has a final state, some strings in each of the k tapes.

- **Theorem 9.1: Every language accepted by a multitape TM is acceptable by some single-tape TM (that is, the standard TM).**
- Proof: Suppose a language L is accepted by a k-tape TM M. We simulate M with a single-tape TM with 2k tracks.
- The second. fourth, ..., (2k)th tracks hold the contents of the k-tapes. The first. third, ... , (2k - 1)th tracks hold a head marker (a symbol say X) to indicate the position of the respective tape head.
- We give an 'implementation description' of the simulation of M with a single tape TM M1. We give it for the case k =2. The construction can be extended to the general case.
- Figure 9.9 can be used to visualize the simulation. The symbols A2 and B5 are the current symbols to be scanned and so the head marker X is above the two symbols.

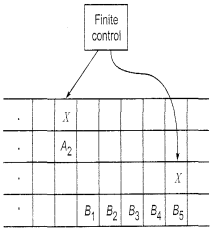


Fig. 9.9 Simulation of multitape TM.

- M1 revisits each of the headmarkers:
- It changes the tape symbol in the corresponding track of M1 based on the information regarding the move of *M corresponding to the state* (of *M) and the tape symbol in the corresponding tape M.*
- (ii) It moves the headmarkers to the left or right.
- (iii) M1 changes the state of *M in its control.*
- This is the simulation of a single move of *M.*

- **Definition.** Let *M* be a TM and *w* an input string.
- The **running time** of *M* on input w is the number of steps that M takes before halting.
- If *M* does not halt on an input string w, then the running time of *M* on *w* is infinite.

- *Note:* Some TMs may not halt on all inputs of length *n.* But we are interested in computing the running time, only when the TM halts.

- **Definition:** The time complexity of TM *M* is the function *T(n), n* being the input size, where *T(n)* is defined as the maximum of the running time of M over all inputs w of size *n.*

Theorem 9.2: If M1 is the single-tape TM simulating multitape TM M, then the time taken by M1 to simulate n moves of M is $O(n^2)$.

Proof

- Let M be a k-tape TM.
- After n moves of M, the head markers of M1 will be separated by 2n cells or less.
- (At the worst. one tape movement can be to the left by 11 cells and another can be to the right by II cells. In this case the tape headmarkers are separated by 211 cells. In the other cases, the 'gap' between them is less).
- To simulate a move of M, the TM M1 must visit all the *k* headmarkers.

- If M starts with the leftmost headmarker,
- M1 will go through all the headmarkers by moving right by at most 2n cells.
- To simulate the change in each tape. M1 has to move left by at most 2n cells; to simulate changes in *k* tapes, it requires at most two moves in the reverse direction for each tape.
- Thus the total number of moves by M1 for simulating one move of M is atmost $4n + 2k$. (211 moves to light for locating all headmarkers, $2n + 2k$ moves to the left for simulating the change in the content of *k* tapes.)
- So the number of moves of M1 for simulating *n* moves of M is $11(4n + 2k)$. As the constant *k* is independent of 11, the time taken by M1 is $O(n^2)$.

NONDETERMINISTIC TURING MACHINES

Definition 9.5 A nondeterministic Turing machine is a 7-tuple $(Q, \Sigma, \Gamma, \delta, q_0, b, F)$ where

1. Q is a finite nonempty set of states
2. Γ is a finite nonempty set of tape symbols
3. $b \in \Gamma$ is called the blank symbol
4. Σ is a nonempty subset of Γ, called the set of input symbols. We assume that $b \notin \Sigma$.
5. q_0 is the initial state
6. $F \subseteq Q$ is the set of final states
7. δ is a partial function from $Q \times \Gamma$ into the power set of $Q \times \Gamma \times \{L, R\}$.

- Theorem 9.3: If M is a nondeterministic TM, there is a deterministic TM M1 such that T(M) = T(M1)
- Proof:
- We construct M1 as a multitape TM.
- Each symbol in the input string leads to a change in ID.
- M1 should be able to reach all IDs and stop when an ID containing a final state is reached.
- So the first tape is used to store IDs of M as a sequence and also the state of M.

- These IDs are separated by the symbol * (induded as a tape symbol).
- The current ID is known by marking an x along with the ID-separator * (The symbol * marked with x is a new tape symbol.)

- All IDs to the left of the current one have been explored already and so can be ignored subsequently.
- Note that the current ID is decided by the current input symbol of w.

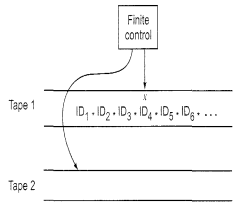


Fig. 9.10 The deterministic TM simulating *M*.

1. M1 examines the state and the scanned symbol of the current ID.
- Using the knowledge of moves of *M stored in the finite control of M1, it* checks whether the state in the current ID is an accepting state of *M*.
- In this case M1 accepts and stops simulating *M*.

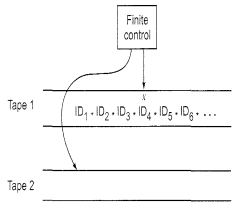


Fig. 9.10 The deterministic TM simulating *M*.

2. If the state q say in the current ID xqay, is not an accepting state of M and δ(q, a) has k triples,
   - M1 copies the ID xqay in the second tape and makes k copies of this ID at the end of the sequence of IDs in tape 2.
   - M1 modifies these k IDs in tape 2 according to the k choices given by δ(q, a).
4. M1 returns to the marked current ID, erases the mark x and marks the next ID-separator * with x (to the * which is to the left of the next ID to be processed). Then M1 goes back to step 1.

Linear bounded Automata

- This model is important because

(a) the set of context-sensitive languages is accepted by the model.

(b) the infinite storage is restricted in size but not in accessibility to the storage in comparison with the Turing machine model. It is called the linear bounded automaton (LBA) because a linear function is used to restrict (to bound) the length of the tape.

- It is a nondetelministic Turing machine

- Formal definition:

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

- ¢ is left end marker and $ is right end marker.
- There are two tapes: input tape and working tape.
- ID is denoted by (q, w, k), where $q \in Q$, $w \in \Gamma$ and k is some integer between 1 and n.

The language accepted by LBA is defined as the set

$$\{w \in (\Sigma - \{¢, \$\})^* | (q_0, ¢w\$, 1) \overset{*}{\vdash} (q, \alpha, i)$$

for some $q \in F$ and for some integer i between 1 and n}.

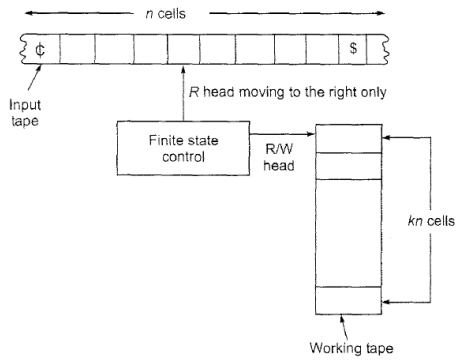


Fig. 9.11 Model of linear bounded automaton.

Decidability(CH-10)

Module 5

Algorithm

- algorithm is a procedure (finite sequence of instructions which can be mechanically carried out) that terminates after a finite number of steps for any input.
- Euclidean algorithm, for computing the greatest common divisor of two natural numbers. In 1900, the mathematician David Hilbert, in his famous address at the International congress of mathematicians in Paris, averred that every definite mathematical problem must be susceptible for an exact settlement either in the form of an exact answer or by the proof of the impossibility of its solution. He identified 23 mathematical problems as a challenge for future mathematicians. But only 10 have been solved so far.

- The Church-Turing thesis states that any algorithmic procedure that can be carried out by a human or a computer, can also be carried out by a Turing machine. Thus the Turing machine arose as an ideal theoretical model for an algorithm.
- The Turing machine provided a machinery to mathematicians for attacking the Hilberts' tenth problem.
- The problem can be restated as follows: does there exist a TM that can accept a polynomial over n variables if it has an integral root and reject the polynomial if it does not have one.

- In 1970, Yuri Matijasevic. after studying the work of Martin Davis, Hilary Putnam and Julia Robinson showed that no such algorithm (Turing machine) exists for testing whether a polynomial over *n variables has integral roots.*
- *Now* it is universally accepted by computer scientists that Turing machine is a mathematical model of an algorithm.

- A procedure for solving a problem is a finite sequence of instructions which can be mechanically carried out given any input.
- An algorithm is a procedure that terminates after a finite number of steps for any input.
- A set X is recursive if we have an algorithm to determine whether a given element belongs to X or not.
- A recursively enumerable set is a set X for which we have a procedure to determine whether a given element belongs to X or not.
- It is clear that a recursive set is recursively enumerable.

Decidability

- Now these terms are also defined using Turing machines. When a Turing machine reaches a final state, it halts.
- We can also say that a Turing machine *M halts when M reaches a state q and a current symbol 'a' to be scanned so* that *δ(q, a) is undefined.*
- *There are TMs that never halt on some inputs in any* one of these ways, So we make a distinction between the languages accepted by a TM that halts on all input strings and a TM that never halts on some input strings.

Definitions

Definition 10.1 A language $L \subseteq \Sigma^*$ is recursively enumerable if there exists a TM M, such that $L = T(M)$.

Definition 10.2 A language $L \subseteq \Sigma^*$ is recursive if there exists some TM M that satisfies the following two conditions.

(i) If $w \in L$ then M accepts w (that is, reaches an accepting state on processing w) and halts.
(ii) If $w \notin L$ then M eventually halts, without reaching an accepting state.

Definition 10.3 A problem with two answers (Yes/No) is decidable if the corresponding language is recursive. In this case, the language L is also called *decidable*.

Definition 10.4 A problem/language is undecidable if it is not decidable.

Decidable languages

- Decidability of regular and context free languages.

Definition 10.5

$A_{\text{DFA}} = \{(B, w) \mid B \text{ accepts the input string } w\}$

Theorem 10.1 A_{DFA} is decidable.

We define a TM M as follows:

1. Let B be a DFA and w an input string. (B, w) is an input for the Turing machine M.
2. Simulate B and input w in the TM M.
3. If the simulation ends in an accepting state of B, then M accepts w. If it ends in a nonaccepting state of B, then M rejects w.

Definition 10.6

$A_{\text{CFG}} = \{(G, w) \mid \text{the context-free grammar } G \text{ accepts the input string } w\}$

Theorem 10.2 A_{CFG} is decidable.

- Convert the grammar into CNF. Then any string w of length k requires 2k-1 steps.
- Then design TM that halts:

1. Let G be a CFG in Chomsky normal form and w an input string. (G, w) is an input for M.
2. If $k = 0$, list all the single-step derivations. If $k \neq 0$, list all the derivations with $2k - 1$ steps.
3. If any of the derivations in step 2 generates the given string w, M accepts (G, w). Otherwise M rejects.

Definition 10.7 $A_{\text{CSG}} = \{(G, w) \mid \text{the context-sensitive grammar } G \text{ accepts the input string } w\}$.

Theroem 10.3 A_{CSG} is decidable.

- Construct TM as follows:

1. Let G be a context-sensitive grammar and w an input string of length n. Then (G, w) is an input for TM.
2. Construct $W_0 = \{S\}$. $W_{i+1} = W_i \cup \{\beta \in (V_N \cup \Sigma)^* \mid \text{there exists } \alpha_i \in W_i \text{ such that } \alpha \Rightarrow \beta \text{ and } |\beta| \leq n\}$. Continue until $W_k = W_{k+1}$ for some k. (This is possible by Theorem 4.3.)
3. If $w \in W_k$, $w \in L(G)$ and M accepts (G, w); otherwise M rejects (G, w). ∎

Undecidable languages

Theorem 10.4 There exists a language over Σ that is not recursively enumerable.

Proof A language L is recursively enumerable if there exists a TM M such that $L = T(M)$. As Σ is finite, Σ^* is countable (that is, there exists a one-to-one correspondence between Σ^* and N).

As a Turing machine M is a 7-tuple $(Q, \Sigma, \Gamma, \delta, q_0, b, F)$ and each member of the 7-tuple is a finite set, M can be encoded as a string. So the set I of all TMs is countable.

Let $\mathcal{L}$ be the set of all languages over Σ. Then a member of $\mathcal{L}$ is a subset of Σ^* (Note that Σ^* is infinite even though Σ is finite). We show that $\mathcal{L}$ is uncountable (that is, an infinite set not in one-to correspondence with N).

Definition 10.8 $A_{\text{TM}} = \{(M, w) \mid \text{The TM } M \text{ accepts } w\}$.

Theorem 10.5 A_{TM} is undecidable.

Proof We can prove that A_{TM} is recursively enumerable. Construct a TM U as follows:

(M, w) is an input to U. Simulate M on w. If M enters an accepting state, U accepts (M, w). Hence A_{TM} is recursively enumerable. We prove that A_{TM} is undecidable by contradiction. We assume that A_{TM} is decidable by a TM H that eventually halts on all inputs. Then

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

We construct a new TM D with H as subroutine. D calls H to determine what M does when it receives the input $\langle M \rangle$, the encoded description of M as a string. Based on the received information on $(M, \langle M \rangle)$, D rejects M if M accepts $\langle M \rangle$ and accepts M if M rejects $\langle M \rangle$. D is described as follows:

1. $\langle M \rangle$ is an input to D, where $\langle M \rangle$ is the encoded string representing M.
2. D calls H to run on $(M, \langle M \rangle)$
3. D rejects $\langle M \rangle$ if H accepts $(M, \langle M \rangle)$ and accepts $\langle M \rangle$ if H rejects $(M, \langle M \rangle)$.

Now step 3 can be described as follows:

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

Let us look at the action of D on the input $\langle D \rangle$. According to the construction of D,

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

This means D accepts $\langle D \rangle$ if D does not accept $\langle D \rangle$, which is a contradiction. Hence ATM is undecidable. ∎

The Turing machine U used in the proof of Theorem 10.5 is called the *universal Turing machine*. U is called universal since it is simulating any other Turing machine.

Halting problem of TM

- Reduction technique-used to prove undecidability of halting problem of TM.

We say that problem A is reducible to problem B if a solution to problem B can be used to solve problem A.

For example, if A is the problem of finding some root of $x^4 - 3x^2 + 2 = 0$ and B is the problem of finding some root of $x^2 - 2 = 0$, then A is reducible to B. As $x^2 - 2$ is a factor of $x^4 - 3x^2 + 2$, a root of $x^2 - 2 = 0$ is also a root of $x^4 - 3x^2 + 2 = 0$.

Note: If A is reducible to B and B is decidable then A is decidable. If A is reducible to B and A is undecidable, then B is undecidable.

Theorem 10.6 $HALT_{TM} = \{(M, w) \mid$ The Turing machine M halts on input $w\}$ is undecidable.

Proof We assume that $HALT_{TM}$ is decidable, and get a contradiction. Let M_1 be the TM such that $T(M_1) = HALT_{TM}$ and let M_1 halt eventually on all (M, w). We construct a TM M_2 as follows:

1. For M_2, (M, w) is an input.
2. The TM M_1 acts on (M, w).
3. If M_1 rejects (M, w) then M_2 rejects (M, w).
4. If M_1 accepts (M, w), simulate the TM M on the input string w until M halts.
5. If M has accepted w, M_2 accepts (M, w); otherwise M_2 rejects (M, w).

When M_1 accepts (M, w) (in step 4), the Turing machine M halts on w. In this case either an accepting state q or a state q' such that $\delta(q', a)$ is undefined till some symbol a in w is reached. In the first case (the first alternative of step 5) M_2 accepts (M, w). In the second case (the second alternative of step 5) M_2 rejects (M, w).

It follows from the definition of M_2 that M_2 halts eventually.

Also, $T(M_2) = \{(M, w) \mid$ The Turing machine accepts $w\}$
$= A_{TM}$

This is a contradiction since A_{TM} is undecidable. ∎

The halting problem

- The Halting Problem (HP):
  Given a TM P and input w, does P halt on w?
- HP is undecidable
- That is, there is no program (turing machine) that solves HP
- If there was such a program
  - Its input will have two portions, P and w
  - It outputs either a YES or a NO depending on whether P halts on input w

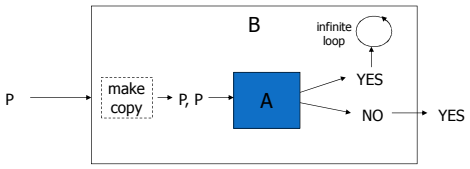
HP is undecidable

- Proof by contradiction: suppose HP is decidable
  - Plan: arrive at a contradiction
- If HP is decidable, then there exists a program(TM) A that solves HP:

HP is undecidable

- Create a program B based on A as follows:
  - B takes in a program P
  - In B, P is duplicated so that there are now two portions on the input tape
  - Feed this new input into A
  - When it is about to print NO, print YES instead
  - When it is about to print YES, send the program to an infinite loop

HP is undecidable

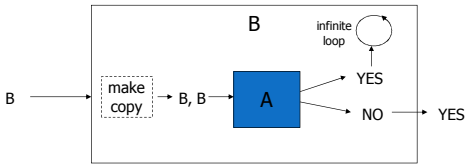


Program B takes a program P as input,
prints a YES if P **does not halt** on input P,
but goes into an infinite loop if P **halts** on input P

HP is undecidable

- Consider feeding program B to itself
- Consequence (two possibilities)
  - It prints a YES
    - B halts on input B
      if B does not halt on input B → a contradiction
  - It goes to an infinite loop
    - B does not halt on input B
      if B halts on input B → a contradiction
- Therefore the supposition cannot hold, and HP is undecidable

Feeding program B to itself



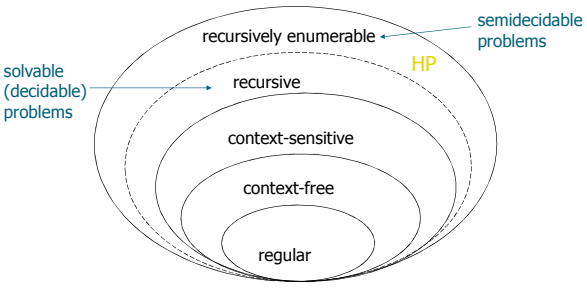
B halts on input B (prints a YES, see outer box) if
B does not halt on input B (A should yield a NO, see inner box)

B does not halt on input B (infinite loop, see outer box) if
B halts on input B (A should yield a YES, see inner box)

Notes and Conclusions

- There are problems such as HP that cannot be solved
- Actually, HP is **semidecidable**, that is if all we need is print YES when P on w halts, but not worry about printing NO if otherwise, a TM machine exists for the halting problem
  - Just simulate P on w, print YES (or go to a final state) when the simulation stops
  - This means that HP is not recursive but it is recursively enumerable

HP and the Chomsky hierarchy



Post correspondence problem

- PCP- introduced by Emil Post in 1946

The Post Correspondence Problem (PCP) was first introduced by Emil Post in 1946. Later, the problem was found to have many applications in the theory of formal languages. The problem over an alphabet Σ belongs to a class of yes/no problems and is stated as follows: Consider the two lists $x = (x_1 \ldots x_n)$, $y = (y_1 \ldots y_n)$ of nonempty strings over an alphabet $\Sigma = \{0, 1\}$. The PCP is to determine whether or not there exist $i_1, \ldots, i_m$ where $1 \le i_j \le n$, such that

$$x_{i_1} \ldots x_{i_m} = y_{i_1} \ldots y_{i_m}$$

Note: The indices i_j's need not be distinct and m may be greater than n. Also, if there exists a solution to PCP, there exist infinitely many solutions.

EXAMPLE 10.1

Does the PCP with two lists $x = (b, bab^3, ba)$ and $y = (b^3, ba, a)$ have a solution?

Solution

We have to determine whether or not there exists a sequence of substrings of x such that the string formed by this sequence and the string formed by the sequence of corresponding substrings of y are identical. The required sequence is given by $i_1 = 2$, $i_2 = 1$, $i_3 = 1$, $i_4 = 3$, i.e. (2, 1, 1,3), and $m = 4$. The corresponding strings are

bab^3	b	b	ba	=	ba	b^3	b^3	a
x_2	x_1	x_1	x_3		y_2	y_1	y_1	y_3

Thus the PCP has a solution.

EXAMPLE 10.2

Prove that PCP with two lists $x = (01, 1, 1)$, $y = (01^2, 10, 1^1)$ has no solution.

Solution

For each substring $x_i \in x$ and $y_i \in y$, we have $|x_i| < |y_i|$ for all i. Hence the string generated by a sequence of substrings of x is shorter than the string generated by the sequence of corresponding substrings of y. Therefore, the PCP has no solution.

Note: If the first substring used in PCP is always x_1 and y_1, then the PCP is known as the *Modified Post Correspondence Problem.*

Theorem 10.7 The PCP over Σ for $|\Sigma| \geq 2$ is unsolvable.

It is possible to reduce the PCP to many classes of two outputs (yes/no) problems in formal language theory. The following results can be proved by the reduction technique applied to PCP.

1. If L_1 and L_2 are any two context-free languages (type 2) over an alphabet Σ and $|\Sigma| \geq 2$, there is no algorithm to determine whether or not
   (a) $L_1 \cap L_2 = \emptyset$,
   (b) $L_1 \cap L_2$ is a context-free language,
   (c) $L_1 \subseteq L_2$, and
   (d) $L_1 = L_2$.
2. If G is a context-sensitive grammar (type 1), there is no algorithm to determine whether or not
   (a) $L(G) = \emptyset$,
   (b) $L(G)$ is infinite, and
   (c) $x_0 \in L(G)$ for a fixed string x_0.
3. If G is a type 0 grammar, there is no algorithm to determine whether or not any string $x \in \Sigma^*$ is in $L(G)$.

Complexity(CH-12)

Module 5

Complexity – chapter 12

- P stands for polynomial / class of problems that can be solved by a deterministic algorithm in polynomial time.
- NP stands for nondeterminisitc polynomial/ class of problems that can be solved by a nondeterministic algorithm in polynomial time.

Growth rate of functions:

- comparison between the running time of two algorithms.
- Growth rate of functions defined on the set of natural numbers(N)

Definition 12.1 Let $f, g : N \to R^+$ (R^+ being the set of all positive real numbers). We say that $f(n) = O(g(n))$ if there exist positive integers C and N_0 such that

$$f(n) \leq Cg(n) \quad \text{for all } n \geq N_0.$$

In this case we say f is of the order of g (or f is 'big oh' of g)

EXAMPLE 12.1

Let $f(n) = 4n^3 + 5n^2 + 7n + 3$. Prove that $f(n) = O(n^3)$.

Solution

In order to prove that $f(n) = O(n^3)$, take $C = 5$ and $N_0 = 10$. Then

$$f(n) = 4n^3 + 5n^2 + 7n + 3 \leq 5n^3 \quad \text{for } n \geq 10$$

When $n = 10$, $5n^2 + 7n + 3 = 573 < 10^3$. For $n > 10$, $5n^2 + 7n + 3 < n^3$. Then, $f(n) = O(n^3)$.

Theorem 12.1 If $p(n) = a_k n^k + a_{k-1}n^{k-1} + \cdots + a_1 n + a_0$ is a polynomial of degree k over Z and $a_k > 0$, then $p(n) = O(n^k)$.

Proof $p(n) = a_k n^k + a_{k-1}n^{k-1} + \cdots + a_1 n + a_0$. As a_k is an integer and positive, $a_k \geq 1$.

As $a_{k-1}, a_{k-2}, \ldots, a_1, a_0$ and k are fixed integers, choose N_0 such that for all $n \geq N_0$ each of the numbers

$$\frac{|a_{k-1}|}{n}, \frac{|a_{k-2}|}{n^2}, \ldots, \frac{|a_1|}{n^{k-1}}, \frac{|a_0|}{n^k} \text{ is less than } \frac{1}{k} \qquad (*)$$

Hence,

$$\left|\frac{a_{k-1}}{n} + \frac{a_{k-2}}{n^2} + \cdots + \frac{a_0}{n^k}\right| < 1$$

As $a_k \geq 1$, $\frac{p(n)}{n^k} = a_k + \frac{a_{k-1}}{n} + \cdots + \frac{a_1}{n^{k-1}} + \frac{a_0}{n^k} > 0$ for all $n \geq N_0$

Also,

$$\frac{p(n)}{n^k} = a_k + \left(\frac{a_{k-1}}{n} + \cdots + \frac{a_1}{n^{k-1}} + \frac{a_0}{n^k}\right)$$
$$\leq a_k + 1 \quad \text{by } (*)$$

So,

$$p(n) \leq Cn^k, \quad \text{where} \quad C = a_k + 1$$

Hence,

$$p(n) = O(n^k). \quad \blacksquare$$

Corollary The order of a polynomial is determined by its degree.

Definition 12.2 An exponential function is a function $q : N \to N$ defined by

$$q(n) = a^n \quad \text{for some fixed } a > 1.$$

When n increases, each of n, n^2, 2^n increases. But a comparison of these functions for specific values of n will indicate the vast difference between the growth rate of these functions.

TABLE 12.1 Growth Rate of Polynomial and Exponential Functions

n	$f(n) = n^2$	$g(n) = n^2 + 3n + 9$	$q(n) = 2^n$
1	1	13	2
5	25	49	32
10	100	139	1024
50	2500	2659	$(1.13)10^{15}$
100	10000	10309	$(1.27)10^{30}$
1000	1000000	1003009	$(1.07)10^{301}$

Classes P and NP

Definition 12.5 A Turing machine M is said to be of time complexity $T(n)$ if the following holds: Given an input w of length n, M halts after making at most $T(n)$ moves.

Note: In this case, M eventually halts. Recall that the standard TM is called a deterministic TM.

Definition 12.6 A language L is in class **P** if there exists some polynomial $T(n)$ such that $L = T(M)$ for some deterministic TM M of time complexity $T(n)$.

EXAMPLE 12.3

Find the running time for the Euclidean algorithm for evaluating gcd(a, b) where a and b are positive integers expressed in binary representation.

Solution

The Euclidean algorithm has the following steps:

1. The input is (a, b)
2. Repeat until $b = 0$
3. Assign $a \leftarrow a \bmod b$
4. Exchange a and b
5. Output a.

Step 3 replaces a by $a \bmod b$. If $a/2 \geq b$, then $a \bmod b < b \leq a/2$. If $a/2 < b$, then $a < 2b$. Write $a = b + r$ for some $r < b$. Then $a \bmod b = r < b < a/2$. Hence $a \bmod b \leq a/2$. So a is reduced by at least half in size on the application of step 3. Hence one iteration of step 3 and step 4 reduces a and b by at least half in size. So the maximum number of times the steps 3 and 4 are executed is $\min\{\lceil\log_2 a\rceil, \lceil\log_2 b\rceil\}$. If n denotes the maximum of the number of digits of a and b, that is $\max\{\lceil\log_2 a\rceil, \lceil\log_2 b\rceil\}$ then the number of iterations of steps 3 and 4 is $O(n)$. We have to perform step 2 at most $\min\{\lceil\log_2 a\rceil, \lceil\log_2 b\rceil\}$ times or n times. Hence $T(n) = nO(n) = O(n^2)$.

Note: The Euclidean algorithm is a polynomial algorithm.

Definition 12.7 A language L is in class **NP** if there is a nondeterministic TM M and a polynomial time complexity $T(n)$ such that $L = T(M)$ and M executes at most $T(n)$ moves for every input w of length n.

Quantum computation

- In 1982, Richard Feynmann, scientist of physics suggested that scientists should start thinking of building computers based on the principles of quantum mechanics.
- The subject of physics studies elementary objects and simple systems and the study becomes more interesting when things are larger and more complicated.
- Quantum computation and information based on the principles of Quantum Mechanics will provide tools to fill up the gulf between the small and the relatively complex systems in physics.

Quantum computers

- We know that a bit (a 0 or a 1) is the fundamental concept of classical computation and information. Also a classical computer is built from an electronic circuit containing wires and logical gates.
- Let us study quantum bits and quantum circuits which are analogous to bits and (classical) circuits.
- A quantum bit, or simply qubit can be described mathematically as

$|\psi> = \alpha|0> + \beta|0>$

Two possible states of qbit are $|0>$ and $|1>$ (notation|.> is called due to Dirac) and infinite number of states.

- α and β are complex numbers such that $|\alpha|^2 + |\beta|^2=1$
- The 0 and 1are called the computational basis states and $|\psi>$ is called a superposition.
- $|\psi> = \alpha|0> + \beta|0>$ is called as quantum state.

Multiple qubits can be defined in a similar way. For example, a two-qubit system has four computational basis states, $|00\rangle$, $|01\rangle$, $|10\rangle$ and $|11\rangle$ and quantum states $|\psi\rangle = \alpha_{00}|00\rangle + \alpha_{01}|01\rangle + \alpha_{10}|10\rangle + \alpha_{11}|11\rangle$ with $|\alpha_{00}|^2 + |\alpha_{01}|^2 + |\alpha_{10}|^2 + |\alpha_{11}|^2 = 1$.

Now we define the qubit gates. The classical NOT gate interchanges 0 and 1. In the case of the qubit the NOT gate, $\alpha|0\rangle + \beta|1\rangle$, is changed to $\alpha|1\rangle + \beta|0\rangle$.

The action of the qubit NOT gate is linear on two-dimensional complex vector spaces. So the qubit NOT gate can be described by

$$\begin{bmatrix}\alpha\\ \beta\end{bmatrix} \rightarrow \begin{bmatrix}0 & 1\\ 1 & 0\end{bmatrix}\begin{bmatrix}\alpha\\ \beta\end{bmatrix} = \begin{bmatrix}\beta\\ \alpha\end{bmatrix}$$

The matrix $\begin{bmatrix}0 & 1\\ 1 & 0\end{bmatrix}$ is a unitary matrix. (A matrix A is unitary if A adj$A = I$.)

We have seen earlier that {NOR} is functionally complete (refer to Exercises of Chapter 1). The qubit gate corresponding to NOR is the controlled-NOT or CNOT gate. It can be described by

$$|A, B\rangle \rightarrow |A, B \oplus A\rangle$$

where $\oplus$ denotes addition modulo 2. The action on computational basis is $|00\rangle \rightarrow |00\rangle$, $|01\rangle \rightarrow |01\rangle$, $|10\rangle \rightarrow |11\rangle$, $|11\rangle \rightarrow |10\rangle$. It can be described by the following 4×4 unitary matrix:

$$\cup_{CN} = \begin{bmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 0 & 1\\ 0 & 0 & 1 & 0\end{bmatrix}$$

- Quantum computer can be defined as "a system built from quantum circuits, containing wires and elementary quantum gates, to carry put manipulation of quantum information"

Church Turing Thesis

- States that "any algorithm that can be performed on any computing machine can be performed on a Turing machine as well"
- Strong Church-Turing Thesis:
- "*Any algorithmic process can be simulated efficiently using a nondetenninistic Turing machine".*
- In 1985, David Deutsch tried to build computing devices using quantum mechanics.
- "*Computers are physical objects, and computations are physical processes. What computers can or cannnot compute is determined by the law of physics alone, and not by pure mathematics "* -David Deutsch