Module: 5

Variants of Turing Machines

The variants of TM are

(i) A TM with more than one tape [Multitape TM]

(ii) A TM where $\delta(q,a) = \{(p_1, Y_1, D_1), (p_2, Y_2, D_2) \ldots (p_r, Y_r, D_r)\}$ [Non-deterministic TM]

Multitape Turing Machine

A multitape turing machine is nothing but a standard TM with more number of tapes.

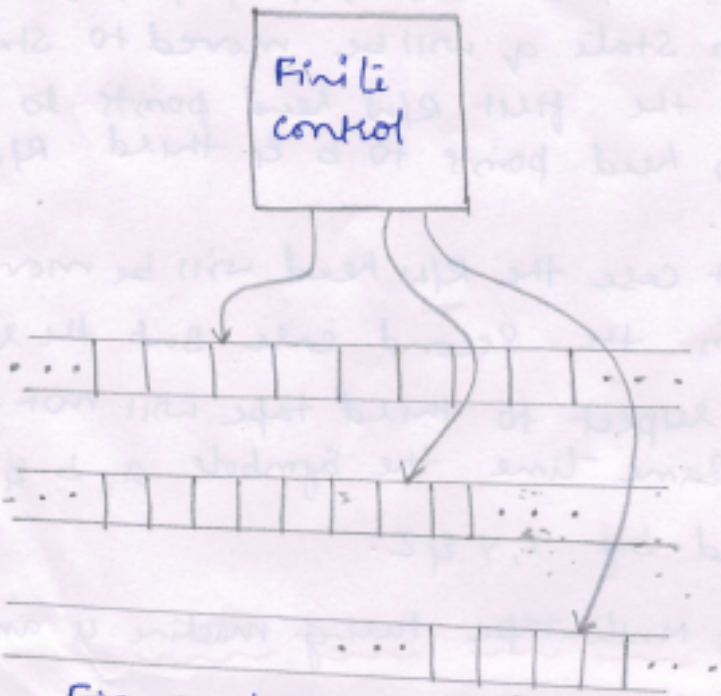


Fig: Multitape Turing machine.

The components of Multitape TM are

* finite control
* Multiple R/W heads.

* Each tape is divided into cells which can hold any symbol from the given alphabet. TO start with the TM should be in start state q_0.
* If the read/write head is pointing to tape 1 moves towards right, the R/W head pointing to tape 2 & tape 3 may move towards right or left depending on the transition.
* The move of the multi-tape TM depends on the current state & the scanned symbol by each of the tape heads.

For eg: If number of tapes in TM is 3

$$\delta(q, a, b, c) = (p, x, Y, Z, L, R, S)$$

* The TM in state q will be moved to state p only when the first R/W head points to a, the second R/W head points to b & third R/W head points to c.
* In the first case the R/W head will be moved to left & right in the second case. But the read/write head with respect to third tape will not be altered.
* At the same time, the symbols a, b & c will be replaced by x, y & z.

Definition of Multi-tape turing machine is an n-tape machine

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

where
* Q is set of finite states.
* Σ is set of input alphabets
* I is set of tape symbols
* δ is transition fnc from $Q \times I^n$ to $Q \times I^n \times \{L, R\}^n$
* q_0 is the start state
* B is a special symbol indicating blank symbol

Theorem: Every language accepted by a multitape TM is acceptable by some single-tape TM.

Proof: Every language L is accepted by a k-tape TM M. We simulate M with a single 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 track hold a head marker (a symbol, say X) to indicate the position of the respective tape head.

In the figure below the symbols A_2 and B_5 are the current symbols to be scanned & so the head marker X is above the two symbols.

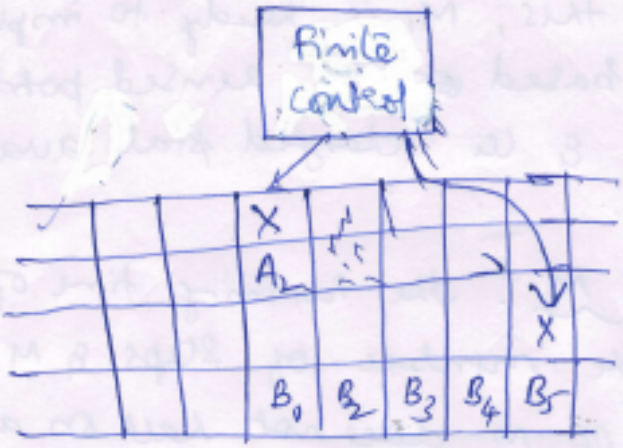


Fig: Simulation of multitape TM.

Initially the contents of tapes 1 & 2 of M are stored in the second & fourth tracks of M_1. The head markers of the first & third tracks are at the cells containing the first symbol.

To simulate a move of M, the 2k-track TM M_1 has to visit the two headmarkers & store the scanned symbols in its control. Keeping track of the headmarkers visited & those to be

storing it in the finite control of M_1. The finite control of M_1 has also the information about the states of M & its moves. After visiting both head markers, M_1 knows the tape symbols being scanned by the two heads of M.

Now M_1 revisits each of the head markers:

(i) It changes the tape symbol in the corresponding track of M_1 based on the information regarding the move of M corresponding to the state (of M) & the tape symbol in the corresponding tape M.

(ii) It moves the head markers to the left or right.

(iii) M_1 changes the state of M in its control.

This is the simulation of a single move of M. At the end of this, M_1 is ready to implement its next move based on the revised position of its headmarkers & the changed state available in its control.

Running time of M: The running time of M on input w, is the number of steps M takes before halting. If M does not halt on an input string w, then the running time of M on w is infinite.

Time complexity of M: Time complexity of M is the fnc $T(n)$, 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 : If M_1 is the single-tape TM simulating multitape TM M, then the time taken by M_1 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 M_1 be separated by 2n cells or less. To simulate a move of M, the TM M_1 must visit all the k headmarkers. If M starts with the leftmost head marker, M_1 will go through all the head markers by moving right by at most 2n cells. TO simulate the change in each tape, M_1 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.

The total number of moves by M_1 for simulating one move of M is atmost $4n+2k$. So the number of moves of M_1 for simulating n moves of M is $n(4n+2k)$. As the constant k is independent of n, the time taken by M_1 is $O(n^2)$.

NONDETERMINISTIC TURING MACHINE

Definition: The nondeterministic Turing machine is a 7-tuple $M=(Q, \Sigma, I, \delta, q_0, B, F)$ where

- Q is set of finite states
- Σ is set of input alphabets
- I is set of tape symbols
- δ is transition fnc which is a subset of $Q \times I \times \{L, R\}$
- q_0 is the start state.
- B is a special symbol indicating blank character.
- $F \subseteq Q$ is set of final states.

It is clear from the definition of δ that for each state q & tape symbol X, $\delta(q_0, X)$ is a set of triples:

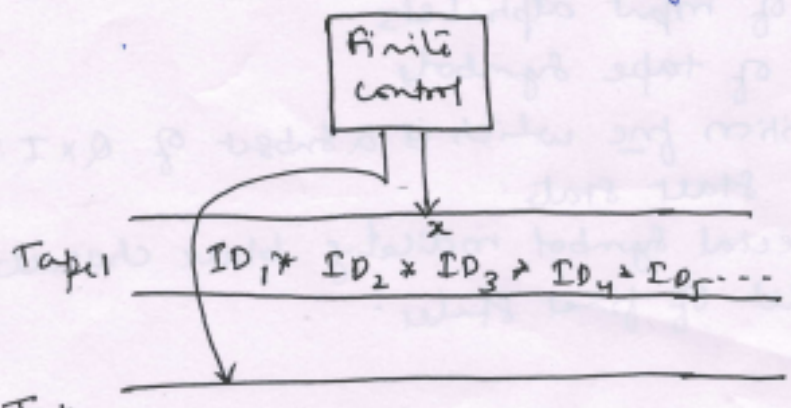
$$\{(q_1, X_1, D), (q_2, X_2, D), (q_3, X_3, D), \ldots, (q_i, X_i, D)\}$$

where

* i is a finite integer.
* D is the direction with 'L' indicating left or 'R' indicating right.
* The machine can choose any of the triples in the next move.

Theorem: If M is a nondeterministic TM, there is a deterministic TM M_1 such that $T(M) = T(M_1)$

Proof: we can construct M_1 as a multitape TM. Each symbol in the input string leads to a change of ID. M_1 should be able to reach all IDs & stop when an ID containing a final state is reached. So the 1st tape is used to store IDs of M as a seq^n & also the state of M. These IDs are separated by the symbol The current ID is known by marking an x along with the ID-separator *. All IDs to the left of the current one have been explored already & so can be ignored subsequently.



To process the current ID, M_1 performs the following steps.

1) M_1 examines the state and the scanned symbol of the current ID. Using the knowledge of moves of M stored in the finite control of M_1. M_1 checks whether the state in the current ID is an accepting state of M. In this case M_1 accepts & stops simulating M.

2) If the state q say in the current ID xqay is not an accepting state of M_1 and $\delta(q, a)$ has k tuples, M_1 copies the ID xqay in the second tape & makes k copies of this ID at the end of the sequence of IDs in tape 2.

3) M_1 modifies these k IDs in tape 2 according to the k choices given by $\delta(q, a)$

4) M_1 returns to the marked current ID, erases the mark x and marks the next ID - Separator # with x. Then M_1 goes back to step 1.

M_1 stops when an accepting state of M is reached in step 1. Now M_1 accepts an input string w only when it is able to find that M has entered an accepting state, after a finite no. of moves. This is clear from the simulated sequence of moves of M_1.

We need to prove that M_1 will reach an accepting ID if M enters an accepting ID after n moves. Note each move of M is simulated by several moves of M_1.

Let m be the maximum number of choices that M has for various (q, a)'s. So for each initial ID of M, there are at most m IDs that M can reach after one move, at most m^2 IDs that M can reach after two moves & so on. So corresponding to n moves of M, there are at most $1 + m + m^2 + \cdots + m^n$ moves of M_1.

Hence the number of IDs to be explored by M_1 is at most $n m^n$.

We assume that M_1 explores these IDs. These IDs have a tree structure having the initial ID as its root. We can apply breadth-first search of the nodes of the tree. If M reaches an accepting ID after n moves, then M_1 has to search atmost n^n IDs before reaching an accepting ID. So if M accepts w, then M_1 also accepts w. Hence $T(M) = T(M_1)$.

The Model of Linear Bounded Automaton

As a linear fnc is used to restrict (to bound) the length of the tape it is called linear bounded automaton (LBA)

A LBA is a nondeterministic TM which has a single tape whose length is not infinite but bounded by a linear function of the length of the input string. The LBA models can be described formally by the following set format:

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

where

Q is set of finite states.
Σ is set of input alphabets which also has two special symbols ¢ and $.
Γ is set of tape symbols.
δ is subset of $Q \times \Gamma$ to $Q \times \Gamma \times \{L, R\}$ with two more transitions of the form:

- $\delta(q_i, ¢) = (q_j, ¢, R)$
- $\delta(q_i, \$) = (q_j, \$, L)$

Forcing the read/write head to be within the boundaries '¢' and '$'.

q_0 is start state
B is a special symbol indicating blank character

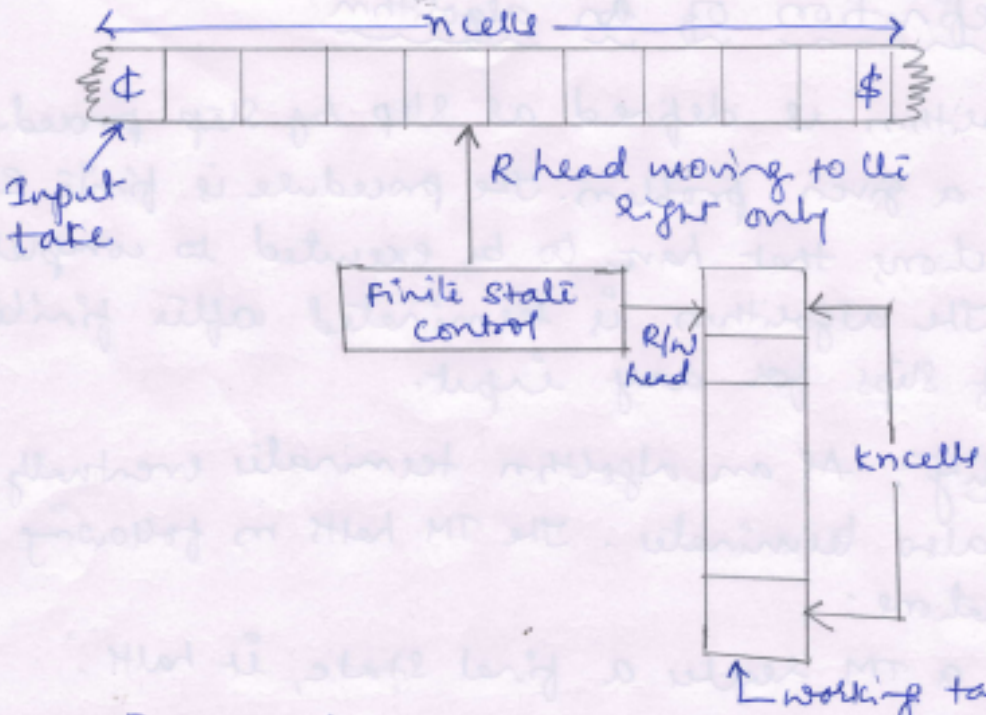


Fig: Model of linear bounded automaton.

In the case of LBA, an ID is denoted by (q, w, k) where $q \in Q$, $w \in I$ & k is some integer between 1 & n. The transition of IDs is similar except that k changes to $k-1$ if the R/W head moves to the left and to $k+1$ if the head moves to the right.

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 & n $\}$.

Relation between LBA and context-sensitive languages

The set of strings accepted by nondeterministic LBA is the set of strings generated by the context-sensitive grammars, excluding the null string.

If L is a context-sensitive language, then L is accepted by a linear bounded automaton. The converse is also true.

The Definition Of an algorithm

An algorithm is defined as step by step procedure to solve a given problem. The procedure is finite sequence of instructions that have to be executed to complete a task. The algorithm is terminated after finite number of steps for any input.

Decidability: As an algorithm terminates eventually, the TM also terminates. The TM halts in following two situations:

* When a TM reaches a final state, it halts.
* When a TM reaches a state q when the scanned input symbol is a and if the transition $\delta(q, a)$ is not defined, it halts.

But, there are some TM that never halt on some inputs in any of the above situations. So, we have to make a distinction between the languages that are accepted by TM & halts on all inputs & a TM that never halts on some input strings.

Recursively enumerable language: A language $L \subseteq \Sigma^*$ is recursively enumerable if and only if there exists a TM M such that $L = T(M)$ where $T(M)$ is the language accepted by TM.

Recursive language: A language $L \subseteq \Sigma^*$ is recursive if & only if there exists a TM that satisfies the following two conditions:

* If $w \in L$ then w is accepted by TM on reaching the final state & the machine halts.
* If $w \notin L$, then w is rejected by TM without

Decidable language: 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.

Undecidable language: A problem/language is undecidable if it is not decidable.

Decidable languages

Theorem: Prove that DFA is decidable language

OR

A_{DFA} is decidable.

Proof: Let $M=(Q,\Sigma,\delta,q_0,F)$ be a DFA. We have to construct a TM M' that always halts & accepts L(M). We know that DFA always ends in some state after reading the string w. Now, we shall construct a TM M' that simulates DFA.
Let us define TM as shown below:

1) If w is the string to be scanned by DFA M, then (M, w) is input to TM M'.
2) Simulate M and input w in TM M'. Here, TM M' checks whether input (M, w) is valid input. If (M, w) is invalid, the TM M' rejects (M, w) and halts. If (M, w) is valid input, M' writes the initial state q_0 and leftmost input symbol of w. It updates the state using δ & reads the next symbol in w.
3) If simulation ends in an accepting state of M, then TM M' accepts w. If the simulation ends in

This, is evident that TM M' accepts (M, w) if and only if w is accepted by DFA M.

Theorem: prove that context-free grammar G is decidable

or

A_{CFG} is decidable.

Proof: We convert a CFG into CNF. Then any derivation of w of length k requires 2k-1 steps if the grammar is in CNF. So for checking whether the input string w of length k is in L(G), It is enough to check derivations in 2k-1 steps. We know that there are only finitely many derivations in 2k-1 steps. Now we design a TM M that halts as follows:

1) Let G be a CFG in CNF & w an input string (G, w) is an input for M.
2) If K=0, list all the single-step derivations. If K≠0, list all the derivations with 2k-1 steps.
3) If any of the derivations in step 2 generate the given string w, M accepts (G, w) otherwise M rejects.

(G, w) is represented by representing its four components V_N, T, P, S of G & input string w. The next step of the derivation is got by the production to be applied.

M accepts (G, w) if & only if w is accepted by the CFG G.

Theorem: A_{CSG} is decidable.

Proof: In case of context-sensitive grammar we construct $W_i = \{\alpha \in (V_N \cup T)^* \mid S \overset{i}{\Rightarrow}_G \alpha$ in i or fewer steps and $|\alpha| \le n\}$. There exists a natural number k such that $W_k = W_{k+1} = W_{k+2} = \cdots$

So $w \in L(G)$ if & only if $w \in W_k$. The construction of W_k is the key idea used in the construction of a TM accepting A_{CSG}. Now we can design a TM M as follows:

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

Undecidable Languages

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

Proof: Let $\mathcal{L}$ be the set of all languages over Σ. Then a member of $\mathcal{L}$ is a subset of Σ^*. we show that $\mathcal{L}$ is uncountable.

we prove this by contradiction. If $\mathcal{L}$ were countable then $\mathcal{L}$ can be written as a sequence $\{L_1, L_2, L_3, \ldots\}$ we write Σ^* as a sequence $\{w_1, w_2 \cdots\}$. So L can be represented as

Where

$$x_{ij} = \begin{cases} 1 & \text{if } w_j \in L_i \\ 0 & \text{otherwise} \end{cases}$$

Using this representation we write L_i as an infinite binary seq^n.

$L_1 : x_{11}\, x_{12}\, x_{13} \ldots x_{ij} \ldots$

$L_2 : x_{21}\, x_{22}\, x_{23} \ldots x_{2j} \ldots$

$\vdots$

$L_i : x_{i1}\, x_{i2}\, x_{i3} \ldots \ldots$

Fig: Representation of L.

We define a subset L of Σ^* by the binary seq^n $y_1 y_2 y_3 \ldots$ where $y_i = 1 - x_{ii}$. If $x_{ii} = 0$, $y_i = 1$ & if $x_{ii} = 1$, $y_i = 0$. Thus according to our assumption the subset L of Σ^* represented by the infinite binary sequence $y_1 y_2 y_3 \ldots$ should be L_k for some natural number k. But $L \neq L_k$ since $w_k \in L$ if & only if $w_k \notin L_k$. This contradicts our assumption that L is countable. Therefore L is uncountable. As I is countable, $\bar{L}$ should have some members not corresponding to any TM in I.

Hence proved...

Theorem : A_{TM} is undecidable

Proof: 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

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

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:

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

Let us see at the action of D on the input $\langle D \rangle$.

According to the construction of D.

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

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

Halting problem:

Given a TM M & an input string w with its initial configuration q_0, after some (or all) computations do the machine M halts? In other words, if M is a TM, we have to identify whether (M, w) halts or does not halt when w is applied at the input. Given the description of an arbitrary TM M & the input string w, we are looking for a style TM that will predict whether or not the computation of M applied to w will halt.

Note: Alan Turing in 1990 proved that the halting problem is undecidable & therefore cannot be solved.

A reduction technique is used to prove the undecidability of halting problem of a TM. Using this technique, a problem A is reducible to problem B if a soln to the problem B can be used to solve the problem A. Thus,

* If A is reducible to B & B is decidable, then, A is decidable.
* If A is reducible to B & B is undecidable, then A is undecidable.

Theorem: $HALT_{TM} = \{(M, w) |$ The TM M halts on input w$\}$ is undecidable.

proof: we assume that $HALT_{TM}$ is decidable, & get a contradiction. Let M_1 be the TM such that $T(M_1) = HALT_{TM}$ & 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)

Observe the following points to prove:

* When M_1 accepts (M, w) (see step 4), the TM M halts on w.
i.e. it is first case (first alternative in step 5)

* In the second case (second alternative in step 5) M_2 rejects (M, w).

So, it follows from definition of M_2 that the TM M_2 halts eventually & hence, $L(M_2) = \{(M, w) :$ The TM accepts $w\}$ which is undecidable. This is a contradiction. So, the TM M halts on input w is undecidable.

The post correspondence problem (pcp)

Defn: The pcp can be stated as follows:
Given two sequence of n strings on some alphabet Σ say:

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

we say that there exists a post correspondence solution for pair (A, B) if there is a non empty sequence of integers $i, j, \cdots, k$, such that

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

The post correspondence problem is to devise an algorithm that will inform us, for any (A, B) whether or not there exists a PC-solution.

(Eg) Does the pcp with two lists $x = (b, bab^3, ba)$ & $y = (b^3, ba, a)$ have a soln?

Soln. The required sequence is given by
$i_1 = 2, i_2 = 1, i_3 = 1, i_4 = 3$ i.e $(2, 1, 1, 3)$ & $m = 4$

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

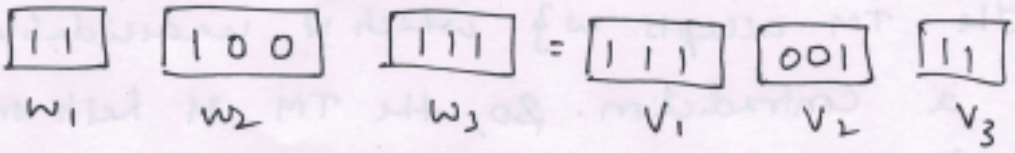
Eg) Let $\Sigma = \{0,1\}$. Let A is w_1, w_2, w_3 as shown below

$w_1 = 11,\ w_2 = 100,\ w_3 = 111$

Let B is v_1, v_2, v_3 as shown below

$v_1 = 111,\ v_2 = 001,\ v_3 = 11$

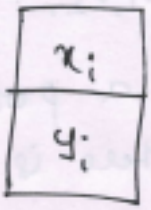
Sol:) The required sequence is given by i=2 3 2



Thus pcp has a solution

Eg) Explain how a pcp can be treated as a game of dominoes.

Sol:) Let each domino contain some x_i in the upper half, & the corresponding substring of y in the lower-half. A typical domino is shown as:



upper-half

lower-half.

The pcp is equivalent to placing the dominoes one after another as a sequence. To win the game, the same string should appear in the upper-half & in the lower-half. So winning the game is equivalent to a solution of the pcp.

Supplementary Examples.

Eg If L is a recursive language over Σ, show that $\bar{L}$ is also recursive.

Soln: As L is recursive, there is a TM M that halts & $T(M) = L$. We need to construct a TM M_1 such that $T(M_1) = \bar{L}$ & M_1 eventually halts.

M_1 is obtained by modifying M as follows:

1. Accepting states of M are made nonaccepting states of M_1.
2. Let M_1 have a new state q_f. After reaching q_f M_1 does not move in further transitions.
3. If q is a non accepting state of M and $\delta(q, x)$ is not defined, add a transition from q to q_f for M_1.

As M halts, M_1 also halts. Also M_1 accepts w if and only if M does not accept w. So $\bar{L}$ is recursive.

Eg If L & $\bar{L}$ are both recursively enumerable, show that L & $\bar{L}$ are recursive.

Soln: Let M_1 & M_2 be two TMs such that $L = T(M_1)$ & $\bar{L} = T(M_2)$. We construct a new two-tape TM M that simulates M_1 on one tape & M_2 on the other.

If the input string w on M is in L, then M_1 accepts w and we declare that M accepts w. If $w \in \bar{L}$, then M_2 accepts w & we declare that M halts without accepting. Thus in both cases, M eventually halts. By the construction of M it is clear that $T(M) = T(M_1) = L$. Hence L is recursive.

(Eg) Show that $\bar{A}_{TM}$ is not recursively enumerable.

Soln: We have seen that A_{TM} is recursively enumerable. If $\bar{A}_{TM}$ were also recursively enumerable, then A_{TM} is recursive. This is a contradiction since A_{TM} is not recursive by thm. Hence $\bar{A}_{TM}$ is not recursively enumerable.

(Eg) Show that the union of two recursively enumerable languages is recursively enumerable & the union of two recursive languages is recursive.

Soln: Let L_1 & L_2 be two recursive languages & M_1, M_2 be the corresponding TMs that halt. We design a TM M as a two-tape TM as follows:

1) w is an input string to M
2) M copies w on its second tape.
3) M simulates M_1 on the first tape. If w is accepted by M_1, then M accepts w.
4) M simulates M_2 on the second tape. If w is accepted by M_2 then M accepts w.

M always halts for any input w.

Thus $L_1 \cup L_2 = T(M)$ & hence $L_1 \cup L_2$ is recursive.

If L_1 & L_2 are recursively enumerable, then the same conclusion gives a proof for $L_1 \cup L_2$ to be recursively enumerable. As M_1 & M_2 need not halt, M need not halt.

Complexity

If a problem/language is decidable, it means that the problem is computationally solvable in principle. But it may require enormous amt of computation time & enormous amount of memory to solve it completely.

P stands polynomial time: This class of problems can be solved by a deterministic algorithm in a polynomial time.

NP stands for non-deterministic polynomial time: This class of problems can be solved by non-deterministic algorithm in a polynomial time.

Growth Rate of functions

Def^n: 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

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

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

Note: $f(n) = O(g(n))$ is not an equation. It expresses a relation between two functions f and g.

Eg: Let $f(n) = 4n^3 + 5n^2 + 7n + 3$

Prove that $f(n) = O(n^3)$.

Soln:

$f(n) = O(n^3)$

take $c = 5$ & $N_0 = 10$

then $f(n) = 4n^3 + 5n^2 + 7n + 3 \leq 5n^3$ 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: If $p(n) = a_k n^k + a_{k-1} n^{k-1} + \ldots a_1 n^1 + a_0$ with $a_k > 0$ then $p(n) = O(n^k)$

Soln: It is given that

$$f(n) = a_k n^k + a_{k-1} n^{k-1} + \ldots a_1 n^1 + a_0$$

Each term in the summation is of the form $a_i n^i$. Since, n is non-negative, a particular term will be negative only if $a_i < 0$ so,

$$|f(n)| = |a_k n^k + a_{k-1} n^{k-1} + \ldots a_1 n^1 + a_0|$$

$$\leq |a_k| n^k + |a_{k-1}| n^{k-1} + \ldots |a_1| n^1 + |a_0|$$

$$= n^k \left[|a_k| + \frac{|a_{k-1}|}{n} + \ldots \frac{|a_1|}{n^{k-1}} + \frac{|a_0|}{n^k} \right]$$

$$\leq n^k (|a_k| + |a_{k-1}| + \ldots |a_1| + |a_0|)$$

$$\leq c^k n^k \text{ where } c = (|a_k| + |a_{k-1}| + \ldots |a_1| + |a_0|)$$

i.e, $f(n) \leq c^k n^k$ where $c = (|a_k| + |a_{k-1}| + \ldots |a_1| + |a_0|)$ for $n > 1$

$\downarrow$

$f(n) \leq c * g(n)$ for $n > n_0$ where $g(n) = n^k$ & $n_0 = 1$

So by definition, we can write

$f(n) = O(g(n))$ or $f(n) \in O(g(n))$

i.e $\boxed{f(n) = O(n^k) \text{ or } f(n) \in O(n^k)}$

The classes P & NP

Defn: A TM M is said to be of complexity T(n) if the following holds: Given an input w of length n, M halts after making at most T(n) moves.

Defn: 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).

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

Eg obtain a time complexity of a TM which accepts $L = \{a^n b^n / n \geq 1\}$

Sol: **Step1:** To construct TM for above language every leftmost a is replaced by X & rightmost b is replaced by Y.

Total no. of moves required = 2n.

Step2: The above procedure is repeated till all n no of a's are replaced by X's & n no of b's replaced by Y.

Step3: Total no of moves = $2n(n) = 2n^2 \cong n^2$

So, the time complexity is given by $O(n^2)$.

Quantum computation

Quantum computers

Defn: A quantum computer is a system built from quantum circuits, containing wires & elementary quantum gates, to carry out manipulation of quantum information.

The digital computer data is represented using binary state 0 & 1 whereas the data in quantum computer is represented mathematically as shown below:

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$$

The qubit can be explained as given below:

(i) Two possible states are represented as $|0\rangle$ & $|1\rangle$

(ii) Unlike classical bit 0 or 1, a qubit can have infinite number of state other than $|0\rangle$ & $|1\rangle$ & can be represented as $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ where $|\alpha|^2 + |\beta|^2 = 1$

(iii) In qubits 0 & 1 are called computational basis states, $|\psi\rangle$ is called a superposition & $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ is called quantum state.

(iv) In digital computer, data can be represented as 0 or 1. But, it is not possible to determine the quantum state on observation. In quantum computer we get a state $|0\rangle$ with probability $|\alpha|^2$ & state $|1\rangle$ with probability $|\beta|^2$.

(v) Multiple qubits can be defined in a similar way.

Eg a two-qubit system has four computational basis states, $|00\rangle, |01\rangle, |10\rangle, |11\rangle$ & quantum states are represented as:

$$|\psi\rangle = \alpha_{00}|00\rangle + \alpha_{01}|01\rangle + \alpha_{10}|10\rangle + \alpha_{11}|11\rangle$$

where

$$|\alpha_{00}|^2 + |\alpha_{01}|^2 + |\alpha_{10}|^2 + |\alpha_{11}|^2 = 1$$

(vi) Normal NOT gate interchanges 0 & 1 whereas in case of qubit NOT gate, $\alpha|0\rangle + \beta|1\rangle$ is changed to $\alpha|1\rangle + \beta|0\rangle$.

Church-Turing thesis

Church's thesis is stated as:

Any "effective computation" or "any algorithmic" procedure that can be carried out by a human being or a team of human beings or a computer, can be carried out by some TM.

In other words, there is an effective procedure to solve a decision problem P if & only if there is a TM that answers yes on inputs $w \in P$ and no for $w \notin P$.

The church thesis predicts that it is unable to construct models of computation more powerful than the existing ones.

As church thesis is closely related to Turing's thesis which states that we cannot go beyond turing machines or their equivalent, it is also called church-Turing thesis.

Since there is no precise def^{n} for "effective computation", "algorithm procedure" church's thesis is not a mathematically precise statement.