UNIT - 5
UNDECIDABILITY

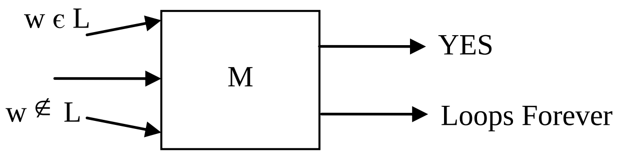
5.1 RECURSIVE AND RECURSIVELY ENUMERABLE LANGUAGES

5.1.1 Recursively Enumerable Language

A language $L \subseteq \sum^*$ is recursively enumerable if there exist a Turing machine, M that accepts every string, $w \in L$ and does not accept strings that are not in L.

If the input string is accepted, M halts with the answer, "YES".

If the string is not an element of L, then M may not halt and enters into infinite loop.

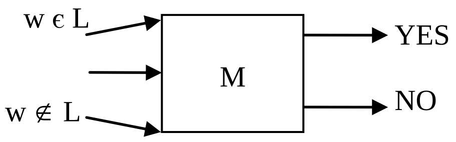


The language, L is Turing Acceptable.

5.1.2 Recursive Language

A language is said to be recursive if there exists of Turing machine, M that accepts every string, $w \in L$ and rejects those strings that are not in L.

If the input is accepted, M halts with the answer," YES"



F the Turing machine doesn't accept the string.

If $w \notin L$, then M halts with answer, "NO". This is also called as Turing Decidable language.

5.1.3 PROPERTIES OF RECURSIVE AND RE LANGUAGES

1. The Union of two recursively enumerable languages is recursively enumerable.

2. The language L and its complement $\overline{L}$ are recursively enumerable, then L and I are recursive.
3. The complement of a recursive language is recursive.
4. The union of two recursive language is recursive.
5. The intersection of two recursive language is recursive.
6. The intersection of two recursively enumerable language is recursively enumerable

5.1.4 Proofs on the Properties

Property-1

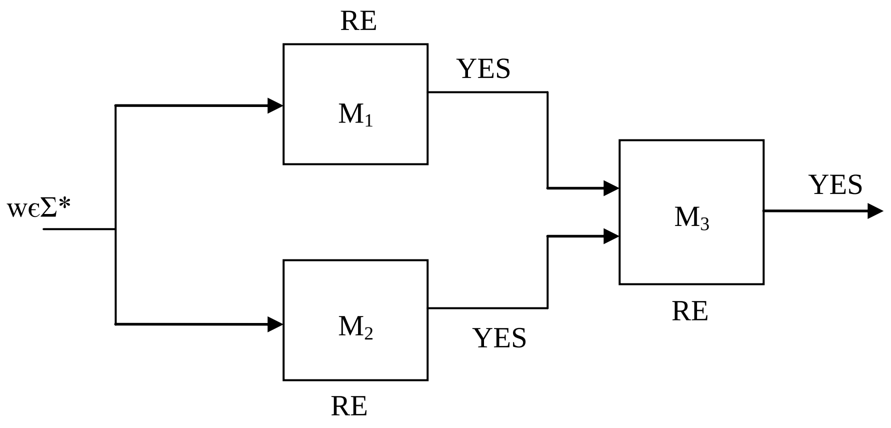
The union of two recursively enumerable languages is recursively enumerable.

Proof:

Let L_1 and L_2 be two recursively enumerable languages accepted by the Turing machines M_1 and M_2.

If a string $w \in L_1$ then M_1 returns "YES", accepting the input string: Else loops forever. Similarly if a string $w \in L_2$ then M_2 returns "YES", else loops forever.

The Turing machine M_3 that performs the union of L_1 and L_2 is given as



Here the output of M_1 and M_2 are written on the input tape of M_3. The turning machine, M_3 returns "YES", if at least one of the outputs of M_1 and M_2 is "YES". The M_3 decides on L_1UL_2 that halts with the answer, "YES" if $w \in L_1$ or $w \in L_2$. Else M_3 loops forever if both M_1 and M_2 loop forever.

Hence the union of two recursively enumerable languages is also recursively enumerable.

Property – 2

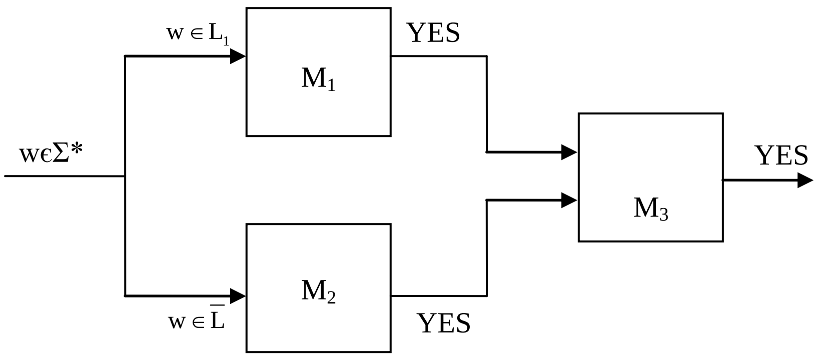
A language is recursive if and only if both it and its complement are recursively enumerable.

Proof

Let L and $\overline{L}$ be two recursively enumerable languages accepted by the Turing machines M_1 and M_2. If a string, $w \in L$, it is accepted by M_1 and M_1 halts with answer "YES". Else M_1 enters into infinite loop.

If a string, $w \in \overline{L}\ [w \notin L]$, then it is accepted by M_2 and M_2 halts with answer "YES". Otherwise M_2 loops forever.

The Turing machine, M_3 that simulates M_1 and M_2 simultaneously is given as



From the above design of TM, if $w \in L$, if $w \in L$, then M_1 accepts w and halts with "YES".

If $w \notin L$, then M_2 accepts $w[w \in \overline{L}]$ and halts with "YES".

Since M_1 and M_2 are accepting the complements of each other, one of them is guaranteed to halt for every input, $w \epsilon \Sigma^*$.

Hence M_3 is a Turing machine that halts for all strings.

Thus if the language and its complement are recursively enumerable, then they are recursive.

Property - 3

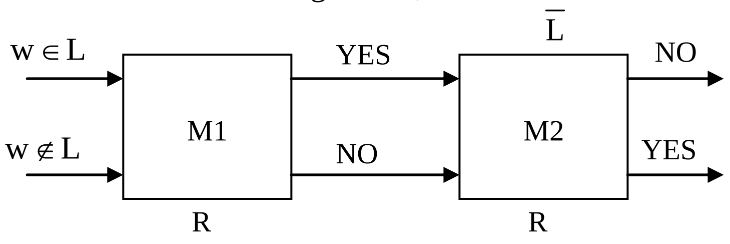
The complement of a recursive language is recursive.

Proof

Let L be a recursive language accepted by the turning machine, M_1.

Let $\overline{L}$ be a recursive language accepted by the Turing machine M_2.

The construction of M_1 and M_2 are given as,



Let $w \in L$, then M_1 accepts w and halts with "YES".

M_1 rejects w if $w \notin L$ and halts with "NO"

M_2 is activated once M_1 halts.

M_2 works on $\overline{L}$ and hence if M_1 returns "YES", M_2 halts with "NO".

If M_1 returns "NO", then M_2 halts with "YES"

Thus for all w, where $w \in L$ or $w \notin L$, M_2 halts with either "YES" or "NO"

Hence the complement of a recursive language is also recursive.

Property – 4

The union of two recursive language is recursive.

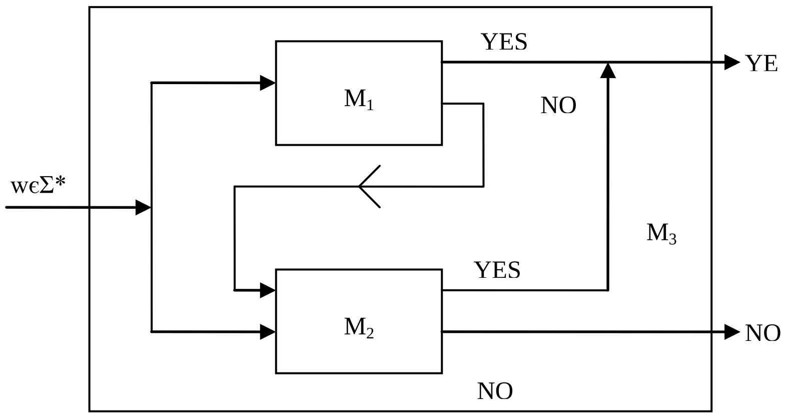
Proof:-

Let L_1 and L_2 be two recursive languages that are accepted by the Turing machines M_1 and M_2, given by

$L(M_1) = L_1$

$L(M_2) = L_2$

Let M_3 be the Turing machine constructed by the union of M_1 and M_2. M_3 is constructed as follows.



The Turing machine M_3 first simulates M_1 with the input string, w.

If $w \in L_1$, then M_1 accepts and thus M_3 also accepts since $L(M_3) = L(M_1) \cup L(M_2)$.

If M_1 rejects string $\left[w \notin L_1\right]$, then M_3 simulates M_2. M_3 halts with "YES" if M_2 accepts 'w', else returns "NO".

Hence M_3, M_2, M_1 halt with either YES or NO on all possible inputs.

Thus the union of two recursive languages is also recursive.

Property – 5

The intersection of two recursive language is recursive.

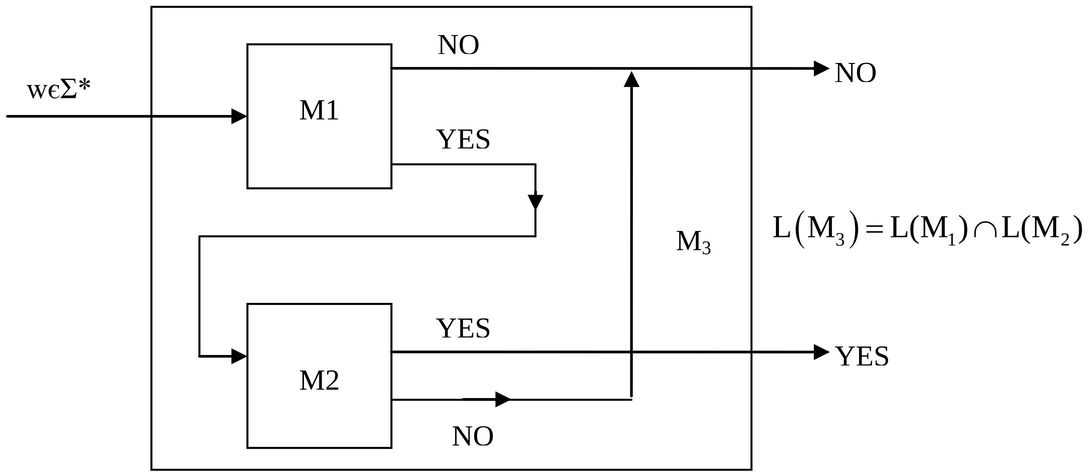
Proof:-

Let L_1 and L_2 be two recursive languages accepted by M_1 and M_2 where

$L(M1) = L_1$

$L(M2) = L_2$

Let M_3 be the Turing machine that is constructed by the intersection of M_1 and M_2, M_3 is constructed as follows.



The Turing machine M_3 simulates M1 with the input string,w.

If $w \notin L_1$, then M_1 halts along with M_3 with answer "NO", since $L(M_3)=L(M_1) \cap L(M_2)$. If then M_1 accepts with the answer "YES" and M_3 simulates M_2.

If M_2 accepts the string, then the answer of M_2 and M_3 are "YES" and halts. Else, M_2 and M_3 halts with answer "NO".

Thus, the intersection of two recursive languages is recursive.

Property – 6

Intersection of two recursively enumerable languages is recursively enumerable.

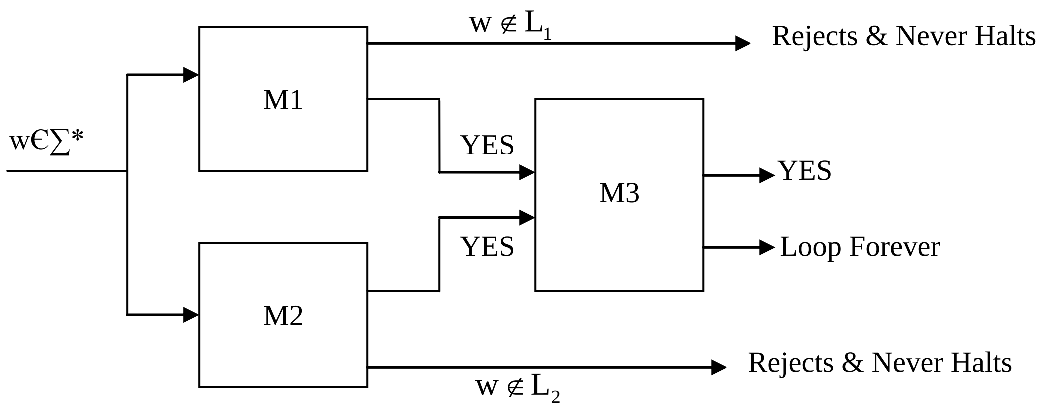
Proof:-

Let L_1 and L_2 be two recursively enumerable languages accepted by the Turing machine M_1 and M_2.

If a string $w \in L_1$ then M_1 returns "YES" accepting the input. Else will not halt after rejecting $w \notin L_1$.

Similarly if a string, $w \in L_2$, then M_2 returns "YES" else rejects 'w' and loop forever.

The Turing machine, $M_3 = M_1 \cap M_2$ is given as



Here the output of M_1 and M_2 are written the input tape of M_3. The machine, M_3 returns "YES" if both the outputs of M_1 and M_2 is "YES".

If at least one of M_1or M_2 is NO it rejects 'w' and never halts.

Thus M_3 decides on $L_1 \cap L_2$ that halts if and only if $w \in L_1$ and $w \in L_2$. Else M_3 loops forever along with M_1 or M_2 or both

Hence the intersection of two recursively enumerable languages is recursively enumerable.

5.2 THE HALTING PROBLEM

- The halting problem is the problem of finding if the program/machine halts or loop forever.
- The halting problem is un-decidable over Turing machines.

Description

- Consider the Turing machine, M and a given string ω, the problem is to determine whether M halts by either accepting or rejecting ω, or run forever.
- **Example**

```
while (1)
{
    prinf("Halting problem");
}
```

- The above code goes to an infinite loop since the argument of while loop is true forever.
- Thus it doesn't halts.
- Hence Turing problem is the example for undecidability.
- This concept of solving the halting problem being proved as undecidable was done by Turing in 1936.
- The undecidability can be proved by reduction technique.

Representation of the halting set

The halting set is represented as,

$$h(M, \omega) = \begin{cases} 1 & \text{if } M \text{ halts on input } \omega \\ 0 & \text{otherwise} \end{cases}$$

where,

$M \rightarrow$ Turing machine

$\omega \rightarrow$ Input string

Theorem

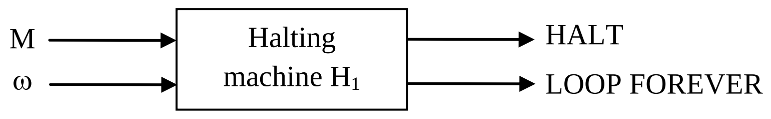
Halting problem of Turing machine is unsolvable / undecidable.

Proof

The theorem is proved by the method of proof by contradiction.

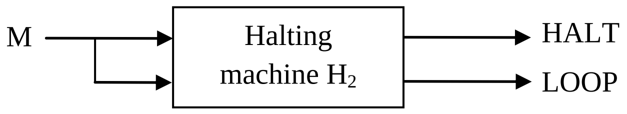
Let us assume that Turing machine is solvable / decidable.

Construction of H_1



- Consider, a string describing M and input string, ω for M.
- Let H_1 generates "halt" if H_1 determines that the turing machine, M stops after accepting the input, ω.
- Otherwise H_1 loops forever when, M doesn't stops on processing ω.

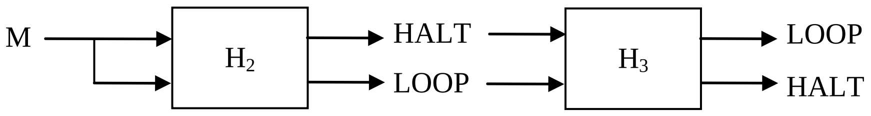
Construction of H_2



H_2 is constructed with both the inputs being M.

H_2 determines M and halts if M halts otherwise loops forever.

Construction of H_3

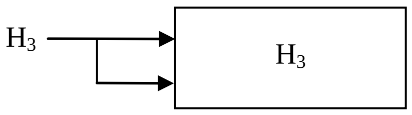


Let H_3 be constructed from the outputs of H_2.

If the outputs of H_2 are HALT, then H_3 loops forever.

Else, if the output of H_2 is loop forever, then H_3 halts.

Thus H_3 acts contractor to that of H_2.



- Let the output of H_3 be given as input to itself.
- If the input is loop forever, then H_3 acts contradictory to it, hence halts.
- And if the input is halt, then H_3 loops by the construction.
- Since the result is incorrect in both the cases, H_3 doesnot exist.
- Thus H_2 doesnot exist because of H_3.
- Similarly H_1 doesnot exist, because of H_2.

Thus halting problem is undecidable.

5.3 PARTIAL SOLVABILITY

Problem types

There are basically three types of problems namely

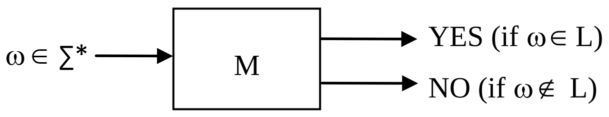
- Decidable / solvable / recursive
- Undecidable / unsolvable
- Semi decidable / partial solvable / recursively enumerable

Decidable / solvable problems

A problem, P is said to be decidable if there exists a turing machine, TM that decides P.

Thus P is said to be recursive.

Consider a Turing machine, M that halts with either ‘yes’ or ‘no’ after computing the input.



The machine finally terminates after processing

It is given by the function,

$$F_p(\omega) = \begin{cases} 1 & \text{if } p(\omega) \\ 0 & \text{if } \neg p(\omega) \end{cases}$$

The machine that applies $F_p(\omega)$ is said to be turing computable.

Undecidable problem

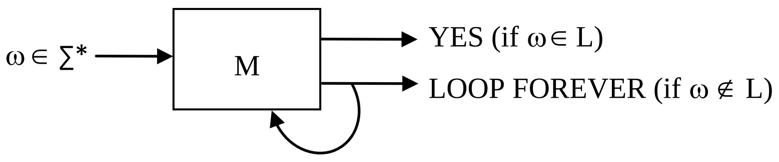
A problem, P is said to be undecidable if there is a Turing machine, TM that doesn’t decides P.

Semi decidable / partial solvable / recursively enumerable

A problem, P is said to be semi decidable, if P is recursively enumerate.

A problem is RE if M terminates with ‘YES’ if it accepts $\omega \in L$; and doesn’t halt if $\omega \notin L$.

Then the problem is said to be partial solvable / Turing acceptable.



Partial solvability of a machine is defined as,

$$F_p(\omega) = \begin{cases} 1 & \text{if } p(\omega) \\ \text{undefined} & \text{if } \neg\, p(\omega) \end{cases}$$

Enumerating a language

Consider a k-tape turing machine. Then the machine M enumerates the language L (such that $L \subseteq \sum^*$) if

- The tape head never moves to the left on the first tape.
- No blank symbol (B) on the first tape is erased or modified.
- For all $\omega \in L$, where there exists a transition rule, δ_i on tape 1 with contents

$$\omega_1 \# \omega_2 \# \omega_3 \# \ldots \# \omega_n \# \omega \# \quad (\text{for } n \geq 0)$$

Where $\omega_1, \omega_2, \omega_3, \ldots\ldots, \omega_n, \omega$ are distinct elements on L.

If L is finite, then nothing is printed after the # of the left symbol

That is,

- If L is a finite language then the TM, M either
    - Halts normally after all the elements appear on the first tape (elements are processed)

      or

    - Continue to process and make moves and state changes without scanning/printing other string on the first tape.

If the language, L is finite, the Turing machine runs forever.

Theorem

A language $L \subseteq \sum^*$ is recursively enumerable if and only if L can be enumerated by some TM.

Proof

Let M_1 be a Turing machine that enumerates L.

And let M_2 accepts L. M_2 can be constructed as a k-tape Turing machine $[k(M_2) > k(M_1)]$.

M_2 simulates M_1 and M_1 pauses whenever M_2 scans the '#' symbol.

M_2 compares its input symbols to that of the symbols before '#' while, M_1 is in pause.

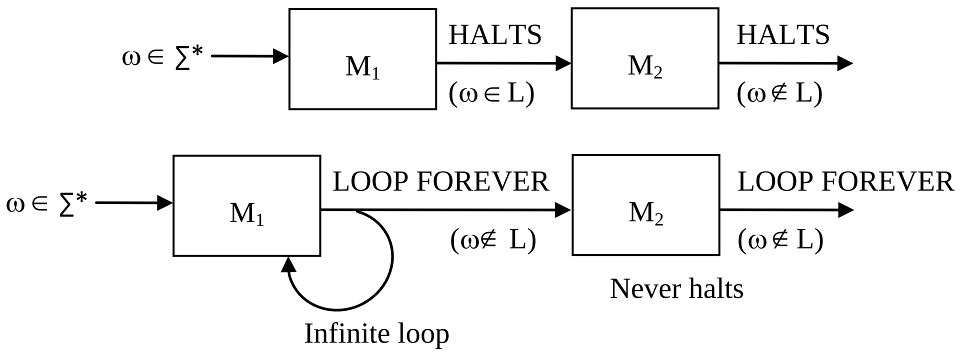
If the comparison finds a match of the string, M_2 accepts L.

Here M_2 is a semi acceptor TM for L

- Scans the input string, ω
- Runs the transition rules of M_1
- If M_1 outputs ω, then ω is accepted and M_1 hats

If $\omega \in L$, M_1 will output ω and M_2 will eventually accept 'ω' and halts.

If $\omega \notin L$, then M_1 will never provide an output ω and so M_2 will never halt.



Thus M_2 is partially solvable / Turing acceptable for L.

5.4 POST CORRESPONDENCE PROBLEM

Post correspondence problem, known as PCP is an unsolvable combinatorial problem. This Undecidable problem was formulated by Emil Post in 1946.

Since PCP is simpler than halting problem, this is used in proofs of undecidability / un-solvable problems.

PCP definition

The input instance of the problem has two lists of non-null strings $\{\alpha_1, \alpha_2, \ldots\ldots\ldots\alpha_n\}$ and $\{\beta_1,\beta_2,\ldots\ldots\ldots\ldots\beta_n\}$ over an alphabets, Σ. [Σ having at most two symbols].

The task is to find a sequence of integers $i_1, i_2, \ldots\ i_k$ [ik for $1 \leq k \leq N$] such that

$$\alpha_i, \alpha_{i2}, \ldots\ldots\alpha_{ik} = \beta_{i1}, \beta_{i2} = \beta_{ik}$$

The PCP is a decision problem that whether the above mentioned sequence exists or not.

EXAMPLE

1. **For Σ = {a, b} with A = {a, aba^3, ab} and B = {a^3, ab, b}, Does the PCP with A and B have a solution?**

Solution:

The sequence obtained from A and B = (2, 1, 1, 3) as,

A_2	A_1	A_1	A_3
aba^3	a	a	ab
B_2	B_1	B_1	B_3
ab	a^3	a^3	b

Thus $A_2A_1A_1A_3 = B_2B_1B_1B_3 = aba^3a^3b = aba^6b$

The PCP given has a solution (2,1,1,3) with the two lists of elements.

2. **Let Σ = {0, 1}. Let A and B be the lists of three strings defined as**

	A	B
I	w_i	x_i
1	1	111
2	10111	10
3	10	0

Solution:

Consider the sequence (2, 1, 1, 3)

$A_2A_1A_1A_3 \Rightarrow w_2w_1w_1w_3 = 101111110$

$B_2B_1B_1B_3 \Rightarrow X_2X_1X_1X_3 = 101111110$

Thus the PCP has (2, 1, 1, 3) sequences as solution

5.5 RICE'S THEOREM

- Rice's theorem is named after the inventor Henry Gordon Rice.
- This theorem is also called as Rice- Myhill – Shapiro theorem.

Theorem

If R is a property of languages that are satisfied by some but not all recursively enumerable languages, then the decision problem,

P_R : Given a TM,M, does L(M) have property, R?
is unsolvable.

Explanation

Consider a language L such that

L = {M | L(M) has some property P}

where,

P is non-trivial of TM,M

Then the language L is undecidable.

That is, every non-trivial property of the language of Turing machines is Undecidable

Proof

Consider a Turing machine, that recognizes empty language, ϕ that does not have the property, P. If ϕ has the property, P, then complement of P is considered

Thus the language is Undecidable due to complementation.

To arrive at a contradiction, assume that P is decidable

Then there is a Turing machine B that recognizes and halts on processing the descriptions of B that satisfy P

Using the Turing machine, M another Turing machine, A is constructed that accepts the language {(M, w) | M) is the Turing machine that accepts w.

As M is Undecidable, then B cannot exist and thus the problem P must be Undecidable Let MP be a Turing machine that satisfies P [P → non trivial] A is computed as

(i) On the input x, M run on the string, w until it accepts. If it doesn't accept, C (M,w) will run forever.

(ii) Run MP on x A accepts if and only MP accepts.

Here C(M, w) accepts the same language as Mp if M accepts w. C(M, w) accepts empty language , ϕ if M does not accept w.

If M accepts w, C(M, w) has the property, P, otherwise, C(M, w) is passed to B

If B accepts, C(M, w) accept the input (M, w) If B rejects, C(M, w) reject

5.6 UNIVERSAL TURING MACHINE

Motive of UTM

A single Turing machine has a capability of performing a function such as addition, multiplication etc.

For computing another function, other appropriate Turing machine is used. To do so, the machine has to be re-written accordingly.

Hence Turing proposed “Stored Program Computer” concept in 1936 that executes the program/instructions using the inputs, stored in the memory.

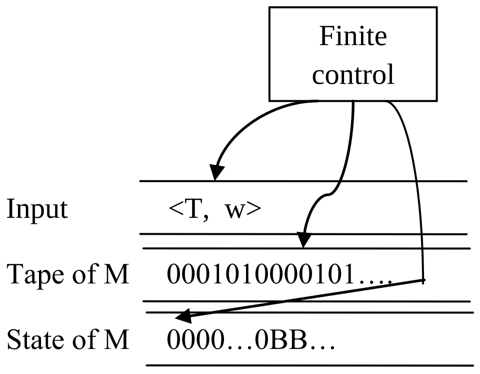
The instructions and inputs are stored on one or more tapes.

Concept of UTM

The universal Turing machine, T_u takes over the program and the input set to process the program.

The program and the inputs are encoded and stored on different tapes of a multi-tape Turing machine.

The T_u thus takes up $\langle T, w \rangle$ where T is the special purpose Turing machine that passes the program in the form of binary string, w is the data set that is to be processed by T.



Encoding T and w

The encoding function, “e” is used to encode both T and w to obtain e(T) and e(w).

The encoding is applied such that the reconstruction of e(T) to T and e(w) to w should also possible without any loss of data.

The encoding function ‘e’ is given as,

$S(\varepsilon) = 0$

$S(a_i) = 0^{i+1}$ $[a_i \in S]$ $[S \rightarrow \text{input symbols / terminals}]$

$S(h_a) = 0$ $[h_a \rightarrow \text{halting state}]$

$S(h_r) = 00$ $[h_r \rightarrow \text{Rejecting / Crash state}]$

$S(q_i) = 0^{i+2}$ $[q_i \rightarrow \text{set of states}]$

$S(S) = 0$ $[S \rightarrow \text{stop}]$

$S(L) = 00$ $[L \rightarrow \text{Left}]$

$S(R) = 000$ $[R \rightarrow \text{Right}]$

We know that the transition move of a Turing machine

$\delta(q_i, T) = (q_i, T, M)$

$\left(\begin{array}{l} q_i, q_j \in Q \\ T \rightarrow \text{terminal symbol} \\ M \rightarrow \text{movement of head} \end{array} \right)$

Which is encoded as,

$$e(m) = 1s(q\,i)1s(T)1s(q\,j)1s(T)1s(M)1$$

Where $e(m) \rightarrow$ encoded transition, m.

$$e(w) = |1s(w_1)|S(w_2)|S(w_3)|.......|S(w_n)|$$

where $e(w) \rightarrow$ encoded input, w

For any Turing machine, T,

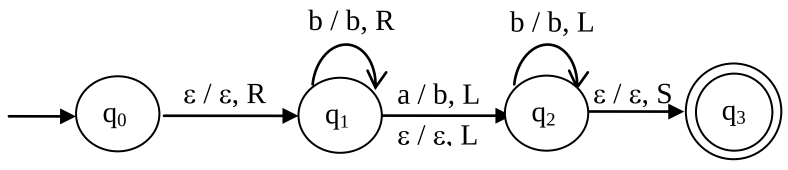
$$e(T) = |s(q)|e(m_1)|e(m_2)|......|e(m_k)|$$

where

$q \rightarrow$ initial state

$m \rightarrow$ transition of TM

Example



The initial state $= \{q_0\}$

Final state $= \{q_3\}$

Inputs = {a,b}

Moves = {L,R,S}

States = $\{q_0,q_1,q_2,q_3\}$

The code for, q_0- first state $\downarrow$

$$\text{Status}\begin{cases} q0 \rightarrow 000 & [q \text{ since } S(q_i)=0^{i+2} \Rightarrow S(q0)=0^{1+2}] \\ q1 \rightarrow 0000, & [S(q_1)=0^{2+2}, \text{ since } q \rightarrow \text{second state}] \\ q2 \rightarrow 00000 & [S(q_2)=0^{3+2}, \text{ since } q_2 \rightarrow \text{third state}] \end{cases}$$

$q3 \rightarrow 0$ [since $q_3 \rightarrow$ halt state, h_a]

$$\text{Inputs}\begin{cases} \varepsilon \rightarrow 0 \ [\text{since } s(\varepsilon) = 0] \\ a \rightarrow 00 \ [\text{since } a(ai) = 0_{i+1};\ a \rightarrow \text{first input; so } i = 1 \Rightarrow s(a) = 0^{1+1}] \\ b \rightarrow 000 \ [s(b) = 0^{2+1}, \text{ since } b \rightarrow \text{second input}] \end{cases}$$

$$\text{Moves}\begin{cases} s \rightarrow 0 \ [\text{since } s(s) = 0] \\ L \rightarrow 00 \ [\text{since } s(L) = 00] \\ R \rightarrow 000 \ [\text{since } s(R) = 000] \end{cases}$$

The transition of the above given Turing machine are,

1. $\delta(q_0, e) = (q_1, \in, R)$
2. $\delta(q_1, b) = (q_1, b, R)$
3. $\delta(q_1, a) = (q_2, b, L)$
4. $\delta(q_1, \in) = (q_2, \in, L)$
5. $\delta(q_2, b) = (q_2, b, L)$
6. $\delta(q_2, \in) = (q_3, \in, s)$

The binary string after applying encoding function

1. $\delta(q_0, \in) = (q_1, \in, R) \Rightarrow 100010100001010001$
2. $\delta(q_1, b) = (q_1, b, R) \Rightarrow 10000100010000100010001$
3. $\delta(q_1, a) = (q_2, b, L) \Rightarrow 1000010010000010001001$

$$4.\delta(q_1, \in) = (q_2, \in, L) \Rightarrow 1000010100000101001$$

$$5.\delta(q_2, b) = (q_2, b, L) \Rightarrow 100000100010000010001001$$

$$6.\delta(q_2, \in) = (q_3, \in, s) \Rightarrow 1000001010101 01$$

$$TM = \{q_0, I, w\}$$

$\hookrightarrow$ Not specified here

$$\rightarrow \langle \underline{000}1000101000010100011000010001000010001$$

$$0001100001001000001000100110000101000001$$

$$010011000001000100000100010011000001010101011$$

Input to the T_u

The universal Turing machine, T_u is always provided with the code for Transitions, e(T) and code for input, e(w) as

$$TM = \langle e(T)e(w) \rangle$$

For example, if the input data, w=”baa”, then

$$e(w) = 10001001001$$

This e(w) will be appended to e(T) of $T_{u.}$

Construction of T_u

As in the figure for universal Turing machine, there are three tapes controlled by a finite control component through heads for each tape.

Tape -1 $\Rightarrow$ Input tape and also serves as output tape. It contain e(T) e(w).

Tape-2 $\Rightarrow$ Tape of the TM/Working tape during the simulation of TM

Tape -3 $\Rightarrow$ State of the TM, current state of the T in encoded form.

Operation of UTM

- UTM checks the input to verify whether the code for TM=<T,w> is a legitimate for some TM.
  - o If the input is not accepted, UTM halts with rejecting, w
- Initialize the second tape to have e(w), that is to have the input, w in encoded form.

- Place the code of the initial state on the third tape and move the head of the finite state control on the first cell of second tape.
- To simulate a move of the Turing machine, UTM searches for the transition - $o^i 1 o^j 1 o^k 1 o^l 1 o^m$ on the first tape, with o^i (initial state/current state) on tape -3 and o^j (input symbol to be processed) on tape- 2.
- The state transition is done by changing the tape -3 content as o^k as in the transition.
- Replace o^j by o^ℓ on tape-2 to indicate the input change.
- Depending on o^m [$m=1 \Rightarrow$ stop, $m=2 \Rightarrow$ Left, $m=3 \Rightarrow$ Right], move the head on tape-2 to the position of the next 1 to the left/right/stop accordingly
- If TM has no transition, matching the simulated state and tape symbol, then no transition will be found. This happens when the TM stops also.
- If the TM, T enters h_a(accepting state), then UTM accepts the input, w

Thus for every coded pair <T, w>, UTM simulates T on w, if and only if T accepts the input string, w.

Definition of Universal Language [L_u]

The universal language, L_u is the set of all binary strings$[\alpha]$, where α represents the ordered pair <T, w> where

$T \rightarrow$ Turing machine

$w \rightarrow$ any input string accepted by T

It can also be represented as $\alpha = \langle e(T)\ e(w) \rangle$.

Theorem

L_u is the recursively enumerable but not recursive .

Proof

From the definition and operations of UTM, we know that L_u is recursively enumerable.

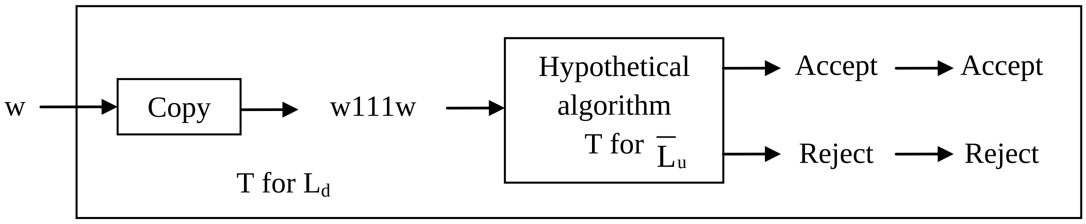
L_u accepts the string w if it is processed by the TM,T. Else, rejects 'w' and the machine doesn't halts forever.

To prove that L_u is not recursive, the proof can be done by contradiction. Let L_u is Turing decidable [recursive], and then by definition L_u (complement of L_u) is Turing acceptable.

We can show that $\overline{L_u}$ is Turing acceptable, that leads to L_d to be Turing acceptable. But we know that L_d is not Turing acceptable.

Hence L_u is not Turing decidable by proof by contradiction.

Proof on $\overline{L_u}$ is during acceptable $\Rightarrow$ L_d is Turing acceptable



Suppose "A" is the algorithm that recognizes L_u .

Then $\overline{L_d}$ is recognizes as follows. Given a string $w \in (0,1)^*$ determined easily, the value of I such that $w = w_i$.

Integer value, I in binary is the corresponding code for TM, T_i. Provide $<T_i, w_i>$ to the algorithm A and accept, w if and only if T_i accepts w_i.

So the algorithm accepts w if and only if $w = w_i$ which is in $L(T_i)$.

This is the algorithm for L_d. Hence L_u is Recursively Enumerable but not recursive.

5.7 TRACTABLE AND INTERACTABLE PROBLEMS

5.7.1 Tractable Problems/Languages

The languages that can be recognized by a Turing machine in finite time and with reasonable space constraint is said to be tractable.

Example: If the language $L_1 \in$ Time (f), then L is tractable and is less complex in nature

Example: If $L_2 \notin$ Time (f), L2 is complex and cannot be tractable in limited time.

Tractable problems are those that can be solved in polynomial time period.

5.7.2 Intractable Problems

The languages that cannot be recognized by any Turing machine with reasonable space and time constraint is called intractable problems.

These problems cannot be solved in finite polynomial time. Even problems with moderate input size cannot achieve feasible solution

5.7.3 Terminologies

Polynomial Time, PSet

The set of languages that can be recognized by a Turing machine with polynomial time complexity.

It is given by,

$$P = U\,Time(Cn^k)$$

$$c>0,\ k>0$$

Polynomial Space, PSpace Set

The set of languages that can be recognized by a TM with polynomial space complexity is said to be PSpace Set.

It is given by,

$$PSpace = U\ Space(Cn^k)$$

$$c>0,\ k \geq 0$$

Non-Deterministic Polynomial Time, NP

The set of languages that are recognized by a Nondeterministic TM with polynomial time is said to be NP.

It is given by,

$$NP = U\ NTime(Cn^k)$$

$$c>0,\ k \geq 0$$

Non-deterministic Polynomial Space, N Space

The set of languages that can be recognized by a NDTH with polynomial Space complexity

It is given by,

$$NP = U\ NSpace(Cn^k)$$

$$c>0,\ k \geq 0$$

***Example:* The CNF satisfiability Problem**

Satisfiability problem is also called as Boolean Satisfiability /Proportional satisfiability/SAT problem.

The SAT problem is to find if there is any interpretation that satisfies the given Boolean equation.

It is to determine the variables of a given boolean formula can be replaced by 'TRUE' or 'FALSE' in a form that evaluates the formula to 'TRUE'.

If 'TRUE' is obtained, the Boolean equation is said to be satisfiable.

If there is no assignment obtained and the evaluation gives out 'FALSE', then the Boolean equation is said to be unsatisfiable.

Example: Let a= TRUE

b=FALSE

a AND b $\Rightarrow$ FALSE

a AND NOT a $\Rightarrow$ FALSE

a AND NOT b $\Rightarrow$ TRUE

a OR b $\Rightarrow$ TRUE

- The operator 'AND' is denoted by conjuction operator ($\wedge$)
- The OR operator is given by disjunction (V).
- NOT operator is denoted by negation ($\neg$) operator

Example: $a \wedge b \Rightarrow \text{FALSE}$

$a \wedge (\neg a) \Rightarrow \text{FALSE}$

$a \wedge (\neg b) \Rightarrow \text{TRUE}$

$a \vee b \Rightarrow \text{TRUE}$

Terminologies

The variable used in the Boolean expression is called a literal. Literals can be either positive or negative.

The normal variable is called positive literal.

The negation of a variable is called negative literal.

The Boolean clause is said to be Horn clause if it contains at most one positive literal.

A formula is in CNF (Conjugative normal form) if it contains/formed by a conjunction of single classes.

Example: $x_1 \Rightarrow$ Positive literal

$\neg x_2 \Rightarrow$ Positive literal

x_1 V $\neg$ $x_2 \Rightarrow$ Clause

$\neg x_2$ V x_2 V x_3 $\Rightarrow$ Not a horn clause

$$\left.\begin{array}{l}\neg x_1 \\ x_1 \, V \neg x_2\end{array}\right\} \Rightarrow \text{horn clause}$$

$$\neg x_1 \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \neg x_2) \Rightarrow \text{CNF}$$

Thus CNF is a conjuction of clauses to form a formula.

$$C_1 \wedge C_2 \wedge C_3 \wedge ..... \wedge C_n \Rightarrow \text{CNF}$$

where,

$$C_1, C_2, C_3 \wedge ..... \Rightarrow \text{Conjuncts / clauses.}$$

K-satisfiability

- Generalized form of CNF with each clause containing upto 'k' literals.

Encoding instances of CNF SAT:-

The instances of CNF SAT can be encoded as,

$$(x_1 \vee \neg x_2) \wedge (x_2 \vee x_3 \vee \neg x_1) \wedge (\neg x_4 \vee x_2)$$

$$\Rightarrow \wedge x_1 \neg x11 \wedge x11x111 \neg x_1 \wedge \neg x1111x11$$

The encoding scheme omits parenthesis and OR (V) notation. The variables are denoted by unary notation of their subscripts.

Thus CNF-Satisfiability is the language over $\sum = \{\wedge, 1, x, \neg x\}$ containing encodings for all positive literals of the instances in CNF SAT.

When there are k-literals, c-conjucts, v-distinct literals, and n is the length of the string encoding these instances,

$$n \leq k(V+1) + \leq k^2 + 2k$$

If CNF – satisfiable $\in$ NP , then the decision problem CNF-SAT is in NP.

5.8 P AND NP COMPLETENESS

Problems are classified into two, namely P and NP.

5.8.1 P

- ‘P’ refers to the class of problems that can be solved in polynomial time.
- Example: Searching an element in an ordered list, sorting elements in a lists, Multiplication of integers , finding all- pair –shortest path, finding minimum spanning tree of a graph, etc
- These are also referred as tractable problems
- Polynomial – time algorithms are efficient ones that can be solved more rapidly.

5.8.2 NP

- ‘NP’ refers to the class of problems that are solved by non-deterministic polynomial time
- These types of NP problems are known as interactable problems.
- Example: Towers of Hanoi, Traveling Salesman problem, Graph colouring problem, Hamiltonian Circuit problem, Satisfiability Problem, etc.
- If a problem, P is solvable in polynomial time by NDTM, then there is no guarantee that there exists a Deterministic TM that can solve P in polynomial time.
- If P is a set of tractable problem, then $P \subseteq NP$
- Every deterministic TM is a special case of case of NDTM.
- NP complete Problem – Types:-
- The NP type class of problems are classified into
    - NP – COMPLETE
    - NP – HARD

5.8.2.1 NP- Complete Problem

A problem is said to be NP- complete if it belongs to NP class problem and can be solved in polynomial time.

They are also called polynomial -time reducible problems.

A NP-complete problem can be transformed into any other in polynomial time.

5.8.2.2 NP-Hard Problem

- A problem is said to be NP hard if there exists an algorithm for solving it and it can be translated into one for solving another NP-Problem.
- A Problem P_1 is NP-hard if
  - the problem is an NP class problem
  - For any other problem , P_2 in NP, there is a polynomial time reduction of L_2 to L_1
- Every NP complete problem must be NP- hard problem