UNIT–5

UNDECIDABILITY

The problems solved by the Turing Machine are classified into two classes.

- ✓ those that have an algorithm
- ✓ those that are only solved by the Turing Machines.

In the first case, the TM halts whether or not it accepts its input. In the latter case, the TM may run forever on inputs they do not accept.

Recursive Language

A language is recursive if there exists a Turing Machine that accepts every string of the language and rejects the string that is not in the language.

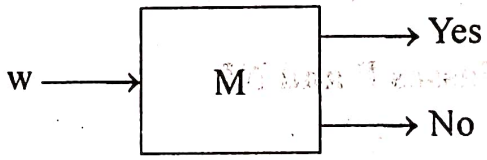

Fig.5.1

Decidable and Undecidable Problems

A problem whose language is recursive is said to be decidable, otherwise it is undecidable.

A problem is said to be undecidable if there is no algorithm that takes as input, an instance of the problem and determines whether the output to that instance is 'Yes' or 'No'.

Examples :

- ✓ The set of strings of equal number 0's and 1's is a decidable problem.
- ✓ Turing Machine M halts with empty tape is an undecidable problem.
- ✓ Does the Turing Machine halt on input w is an undecidable problem.

5.1 A LANGUAGE THAT IS NOT RECURSIVELY ENUMERABLE

A language is recursively enumerable if there exists a Turing Machine that accepts every string of the language and does not accept strings that are not in the language.

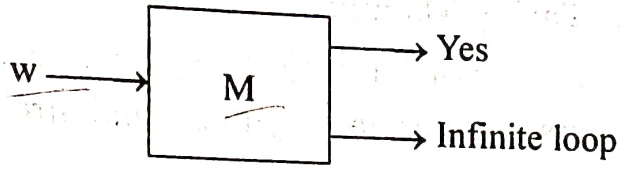

Fig.5.2

The long range goal of proving the undecidability consisting of pairs such that

(i) M is a Turing Machine encoded in binary.

(ii) w is a strings of 0's and 1's

(iii) M accepts input w.

If the problem with the binary inputs is undecidable, then Turing Machine with any other alphabets is undecidable.

The inputs given to the Turing Machine should be in a well-formed representation using binary values.

The *diagonalization language* L_d consists of all those strings w such that the TM represented by w does not accept the input w. L_d has no Turing Machine that accepts it.

5.1.1 Enumerating the Binary Strings

It is necessary to assign integer values to the binary strings so that each string corresponds to one integer, and each integer corresponds to one string.

Eg :

Let *w* be the binary string.

Take 1w as a binary integer i.

This implies w as the i^{th} string.

Likethat we can say ε as the first string, 0 the second, 1 the third, 00 the fourth and so on. Strings are ordered by length and strings of equal length are ordered lexicographically. Therefore, the integer i can be represented by w_i.

5.1.2 Codes for Turing Machine

To represent a TM $M = (Q, \{0,1\}, \vdash, \delta, q_1, B, F)$ as a binary string, first assign integers to states, tape symbols and directions L and R.

- ✓ Assume the states are $q_1, q_2, \ldots . q_k$ for some K. Using the integers available in the suffix of each state, the string can be represented as $q_1 \rightarrow 0, q_2 \rightarrow 00, q_3 \rightarrow 000$, etc.
- ✓ Assume the tape symbols 0, 1, B are represented as $x_1, x_2, x_3 \ldots x_m$. Where

  $x_1 \rightarrow 0$

  $x_2 \rightarrow 1$

  $x_3 \rightarrow B$
- ✓ Assume the directions are represented as D_1 and D_2, where

  $L \rightarrow D_1 \rightarrow 0$

  $R \rightarrow D_2 \rightarrow 00$

After representing each state, symbol and direction using integers, we can encode the transition function.

Suppose $\delta(q_i, x_j) = (q_k, x_l, D_m)$ for some integers *i, j, k, l, m.*

Then the encoded string is given by $0^i 1 0^j 1 0^k 1 0^l 1 0^m$.

The code for the entire Turing Machine M consists of all the strings for the transitions in some order, separated by pairs of 1's.

i.e., $C_1 11 C_2 11 C_3 11 C_4 11 \ldots C_{n-1} 11 C_n$

Where each of the C's is the code for one transition of M.

Example

Obtain the code for $\langle M, 1011 \rangle$, where

$$M = (\{q_1, q_2, q_3\}, \{0, 1\}, \{0, 1, B\}, \delta, q_1, B, \{q_2\})$$

has moves

$$\delta(q_1, 1) = (q_3, 0, R)$$
$$\delta(q_3, 0) = (q_1, 1, R)$$
$$\delta(q_3, 1) = (q_2, 0, R)$$
$$\delta(q_3, B) = (q_3, 1, L)$$

[Nov/Dec-2003 & 2004]

Solution

Assume the tape symbols be encoded as

$0 \text{—} x_1$

$1 \text{—} x_2$

$B \text{—} x_3$

And directions be encoded as

$L \text{—} D_1$

$R \text{—} D_2$

And the transitions are encoded as follows

(i) $\delta(q_1, 1) = (q_3, 0, R)$

$C_1 \Rightarrow 0^1 1 0^2 1 0^3 1 0^1 1 0^2$

(ii) $\delta(q_3, 0) = (q_1, 1, R)$

$C_2 \Rightarrow 0^3 1 0^1 1 0^1 1 0^1 1 0^2$

(iii) $\delta(q_3, 1) = (q_2, 0, R)$

$C_3 \Rightarrow 0^3 1 0^2 1 0^2 1 0^1 1 0^2$

(iv) $\delta(q_3, B) = (q_3, 1, L)$

$C_4 \Rightarrow 0^3 1 0^3 1 0^3 1 0^2 1 0^1$

Then the complete code is given by

$C_1 11 C_2 11 C_3 11 C_4$

M = 0100100010100 11 0001010100100 11

00010010010100 11 0001000100010010

∴ Code for ⟨ M, 1011 ⟩, is given by

= ⟨0100100010100 11 0001010100100 11

00010010010100 11 0001000100010010, 1011⟩

5.1.3 The Diagonalization Language

[Apr/May'04 & Nov/Dec'04 & Apr/May'05]

Some integers do not belong to any Turing Machine. If w_i is not a valid TM code, then take the Turing Machine M_i to be the TM with one state and no transitions. So for these values of i, M_i is a Turing Machine that halts on any input.

∴ $L(M_i) = \phi$ if $w_i \notin$ valid TM code

Definition

The *diagonalization language* L_d, is the set of strings w_i where w_i is not in $L(M_i)$.

L_d consists of all strings w such that the TM M whose code is w does not accept the input w.

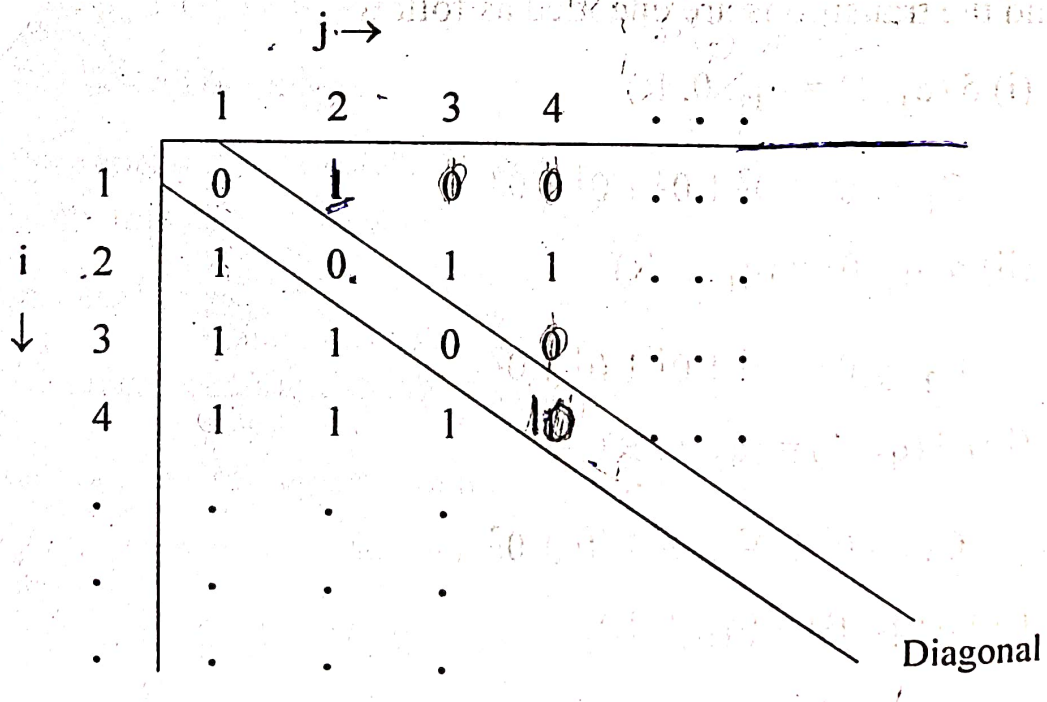

Fig.5.3 L_d – Diagonalization Language

From the table in Fig(5.3), it is clear that whether the Tm M_i accepts the input string w_j or not.

if (i, j) = 1, Yes it is accepted

if (i, j) = 0, No, it is rejected.

The i^{th} row be called as the characteristic vector for the language $L(M_i)$, where 1's in this row correspond to the members of this language.

In order to find L_d, we need to complement the diagonal. In the table (Fig.5.3) the diagonal value be 0001 And then the complemented diagonal = 1110.

Therefore, L_d contains

$w_1 = e$

$w_2 = 0$

$w_3 = 1$

and does not contain $w_4 = 00$ and so on.

Diagonalization

The process of complementing the diagonal to construct the characteristic vector of a language that cannot be the language that appears in any row is called Diagonalization.

Then the complement of the diagonal cannot be the characteristic vector of any Turing Machine.

5.1.4 L_d is not Recursively Enumerable

Theorem : 5.1

L_d is not a recursively enumerable language.

i.e., There is no Turing Machine that accepts L_d.

Proof :

Suppose L_d is accepted by some TM M defined by L(M). Since L_d is a language over alphabet {0, 1}, M would be in the list of Turing machines constructed, where it includes all turing machines with input alphabet {0, 1}. So there may be atleast one code for M, say i, that is $M = M_i$.

Then whether wi is in Ld.

By definition

$$L_d = \{w_i \mid M_i \text{ does not accept } w_i\}$$

Here we have two possibilities

✓ $w_i \in L_d$

This means that (i, i) entry is '0' and so M_i does not accept w_i. But our assumption here is that there exists a Turing machine M_i, which accepts w_i. There is a contradiction.

✓ $w_i \notin L_d$

This means that (i, i) entry is '1' and so M_i accept w_i. But by definition of L_d, M_i does not accepts w_i. So there is a contradiction.

Thus it is clear that L_d is not recursively enumerable, and L_d is not Recursive too.

PROBLEMS

Write the code for the Turing Machine

$$M = (\{q_0, q_1, q_2, q_3, q_4\}, \{0, 1\}, \{0, 1, X, Y, B\}, \delta, q_0, B, \{q_4\})$$

where δ is given by

$$\delta(q_0, 0) = (q_1, X, R) \qquad \delta(q_2, 0) = (q_2, 0, L)$$
$$\delta(q_0, Y) = (q_3, Y, R) \qquad \delta(q_2, X) = (q_0, X, R)$$
$$\delta(q_1, 0) = (q_1, 0, R) \qquad \delta(q_2, Y) = (q_2, Y, L)$$
$$\delta(q_1, 1) = (q_2, Y, L) \qquad \delta(q_3, Y) = (q_3, Y, R)$$
$$\delta(q_1, Y) = (q_1, Y, R) \qquad \delta(q_3, B) = (q_4, B, R)$$

Solution

Assume the tape symbols and states be encoded as

$0 — x_1$ $\quad q_0 = 0$

$1 — x_2$ $\quad q_1 = 00$

$X — x_3$ $\quad q_2 = 000$

$Y — x_4$ $\quad q_3 = 0000$

$B — x_5$ $\quad q_4 = 00000$

And directions be encoded as

$L — D_1$

$R — D_2$

And the transitions are encoded as follows

(i) $\delta(q_0, 0) = (q_1, X, R)$

$C_1 = 01010010001 00$

(ii) $\delta(q_0, Y) = (q_3, Y, R)$

$C_2 = 01000010000100001 00$

(iii) $\delta(q_1, 0) = (q_1, 0, R)$

$C_3 = 001010010100$

(iv) $\delta(q_1, 1) = (q_2, Y, L)$

$C_4 = 0010010001000010$

(v) $\delta(q_1, Y) = (q_1, Y, R)$

$C_5 = 0010000100100001 00$

(vi) $\delta(q_2, 0) = (q_2, 0, L)$

$C_6 = 0001010001010$

(vii) $\delta(q_2, X) = (q_0, X, R)$

$C_7 = 0001000101000100$

(viii) $\delta(q_2, Y) = (q_2, Y, L)$

$C_8 = 0001000010001000010$

(ix) $\delta(q_3, Y) = (q_3, Y, R)$

$C_9 = 0000100001000010000100$

(x) $\delta(q_3, B) = (q_4, B, R)$

$C_{10} = 0000100000100000100000100$

Then the complete code is given by

$C_1\ 11\ C_2\ 11\ C_3\ 11\ C_4\ 11\ C_5\ 11\ C_6\ 11\ C_7\ 11\ C_8\ 11\ C_9\ 11\ C_{10}$.

$= \langle 0101001000100110100001000010000100110010100101001$
$1101001001000100001011001000010010000100111$
$0001010001010\ 11\ 0001000101000100\ 1100010000100010000100$
$11\ 0000100001000010000100\ 1100001000001000001000001000\rangle$

5.2 AN UNDECIDABLE PROBLEM THAT IS RE

The structure of RE language is refined into two classes.

- ✓ An algorithm which recognizes the language as well as the time of deciding whether the input string is not in the language. Such a Turing Machine always halts regardless of whether or not it reaches an accepting state.
- ✓ Languages consisting of RE languages that are not accepted by any Turing Machine with the guarantee of halting.

5.2.1 Recursive Languages

A language L is recursive if L = L(M) for some Turing Machine M such that

(i) If w Σ L, then M accepts and goes to halt state.

(ii) If w ∉ L, then M halts even though it never enters an accepting state.

A language which is having a definite algorithm is called *Recursive* language. If a problem is defined using recursive language, then that problem is *decidable* or else it is undecidable.

The following figure 5.4 suggests the relationship between three classes of languages.

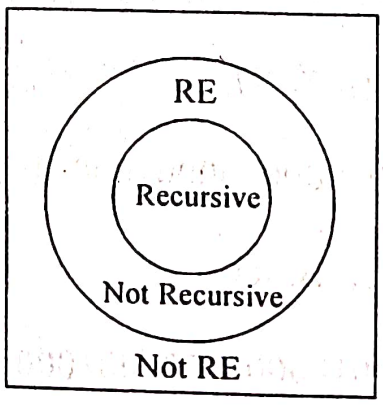

Fig.5.4 Relationship Diagram

5.2.2 Complements of Recursive and RE languages

The recursive languages are closed under complementation.

Theorem : 5.2

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

Proof :

Let L be a recursive language and M a Turing Machine that halts on all inputs and accepts L. Construct a Turing Machine M^1 from M such that

(i) The accepting states of M are made non accepting states of M^1 with no transitions.

(ii) Create a new accepting state P^1 which has no transitions.

(iii) For each pair of non accepting sate of M and a tape symbol of M such that M has no transition, Make the transition to the accepting state P^1.

If M enters a final state on input w, then M^1 halts without accepting.

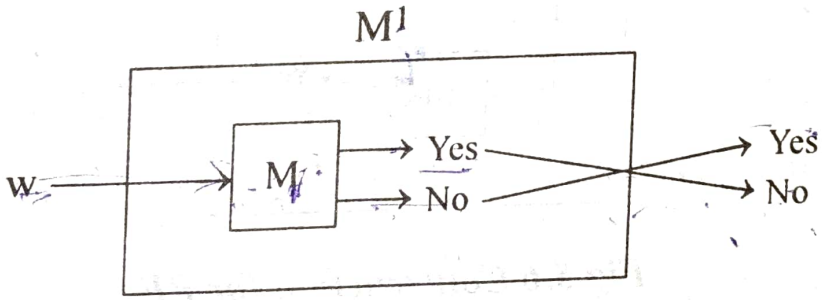

Fig.5.5 Turing Machine $\overline{M}$

If M halts without accepting, M enters into final state.

M	M^1
* Accept → final state	* Rejects
* Rejects	* Accept → final state

From Fig.5.5, it is clear that M^1 has either of the events accepts or rejects. So, M^1 accepts I which is also recursive.

Theorem : 5.3

If a language L and its complement $\overline{L}$ are both recursively enumerable, then L is recursive.

Proof :

Let M_1 and M_2 be the Turing Machine which accepts the languages L and $\overline{L}$ respectively.

Construct M to simulate simultaneously M_1 and M_2.

M_2	M_1	M
0	1	Accepts
1	0	Rejects

M accepts w if M_1 accepts w

M rejects w if M_2 accepts w

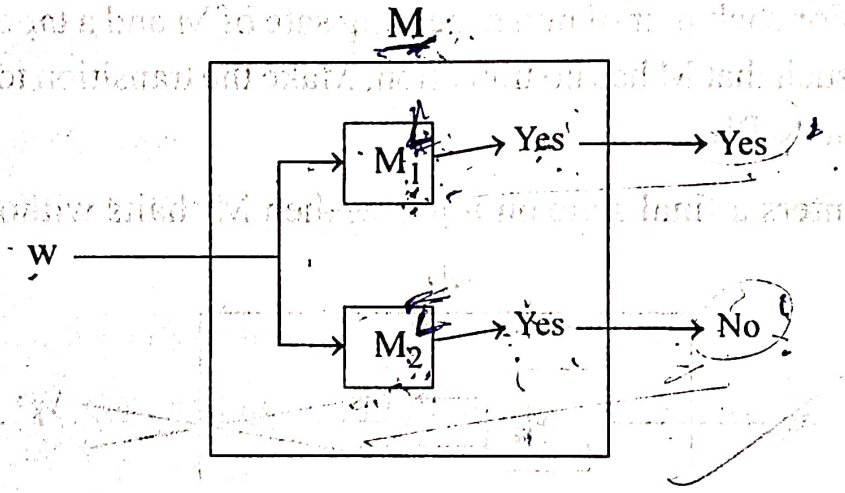

Fig.5.6 Construction for RE

w is in either L or $\overline{L}$, So exactly one of M_1 or M_2 will accept.

Obviously M will always say either 'Yes' or 'No' but not both.

5.2.3 The Universal Language

The diagonalization language is the language consisting of strings w_i such that M_i does not accept w_i.

$$L_d = \{w_i \mid w_i \Sigma L(M_i)\}$$

Consider the binary string representation $\langle M, w \rangle$ such that M accepts w called as universal language L_u.

$$L_u = \{\langle M, w \rangle \mid M \text{ accepts } w\}$$

L_u is the set of strings representing a TM and an input accepted by that TM. So there is a TM U called as Universal Turing Machine.

Theorem : 5.4

L_u is recursively enumerable.

Proof :

In order to prove this theorem, it is necessary to construct a Turing Machine U that accepts L_u. The Turing Machine U consists of a three track input tape where the first track holds the input tape($\langle M, w \rangle$), the second track contains the tape of M where tape symbols are written in unary form and the third track represents the state q_i which is also in unary form.

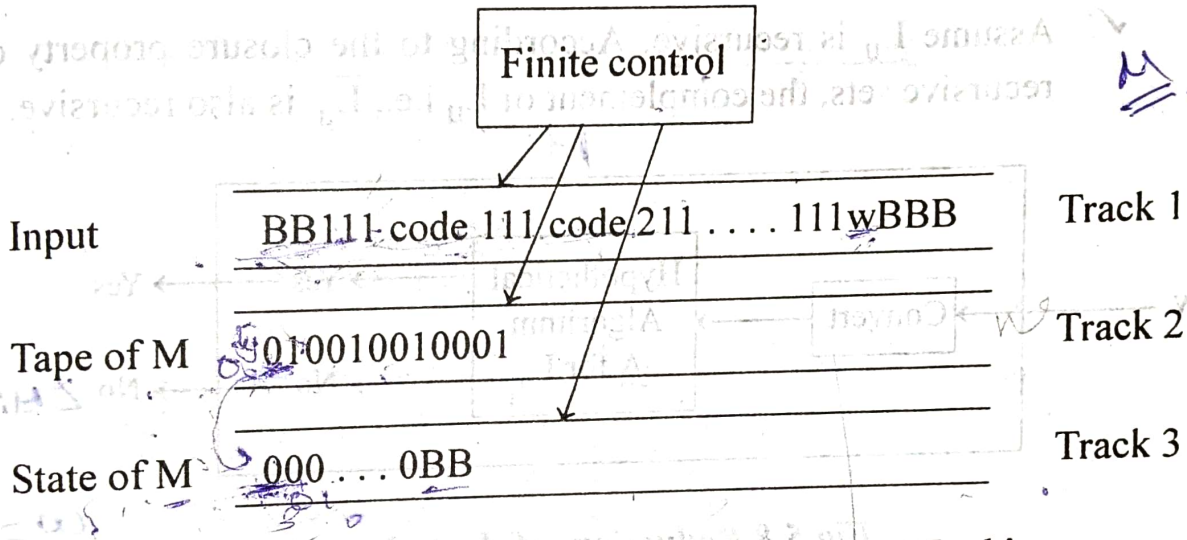

Fig.5.7 Organization of Universal Turing Machine

The operation of U are as follows.

1. First make sure that the code for M is a legitimate code for some Turing Machine M. Otherwise it halts without accepting.
2. Initialize the second tape with the input w in its encoded form. Keep 0 the start state of M on the third tape and move the head of U's second tape to the first simulated cell.
3. If 0^i is the current state with 0^j the current input symbol appeared on track three and two respectively then U finds the corresponding transition of the form $0^i 10^j 10^k 10^l 10^m$ on track 1 and replaces 0^i by 0^k and 0^j by 0^l.
4. Move the head on tape two to the position corresponding to the value of M.
5. The Universal Turing Machine U accepts $0^i 10^j 10^k 10^l 10^m$ if M has the transition of the form $\delta(0^i, 0^j) = (0^k, 0^l, 0^m)$. Otherwise M halts without accepting.

Thus U simulates M and accepts w. Thus L_u is recursively enumerable.

5.2.4 Undecidability of the Universal Language

A problem that is RE but not recursive like L_u, is undecidable. because reducing the problem L_u to another problem T can be used to show there is no algorithm to solve T, regardless of whether or not T is RE.

Theorem : 5.5

L_u is RE but not recursive.

Proof :

- ✓ It is already proved that L_u is RE.
- ✓ Assume L_u is recursive. According to the closure property of recursive sets, the complement of L_u i.e., $\overline{L}_u$ is also recursive.

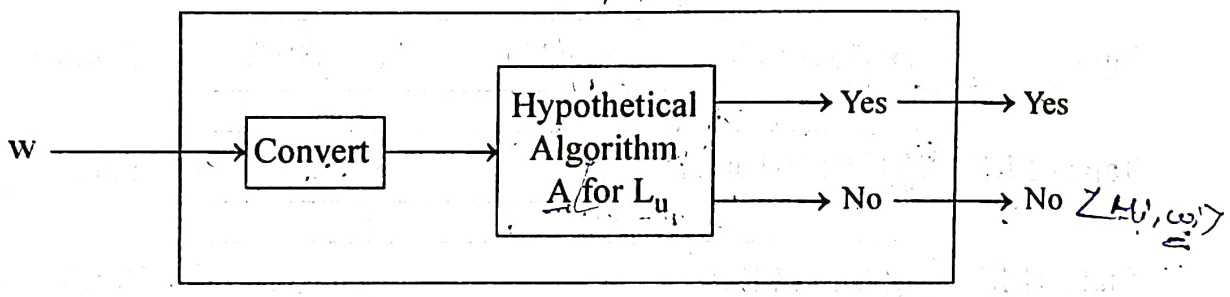

Fig.5.8 Reduction of L_d to L_u

Suppose A is an algorithm recognizing L_u .$\overline{L}_d$ can be recognized as follows.

- ✓ Given string w in (0+1)* determined by an easy calculation, the value of i such that $w = w_i$. Integer i in binary is the corresponding code for some TM M_i.
- ✓ Give the input $\langle M_i, w_i \rangle$ to algorithm A and accept w if and only if M_i accepts w_i.
- ✓ So the constructed algorithm also accepts w if and only $w = w_i$ which is in $L(M_i)$. This is the algorithm constructed for $\overline{L}_d$.

But no such algorithm exists and also our assumption of there is an algorithm A for L_u exists is false.

Hence L_u is RE but not recursive.

5.3 UNDECIDABLE PROBLEMS ABOUT TURING MACHINE

Undecidable problems are all about Turing Machines. From the Rice's theorem, *'Any non–trivial property of Turing Machines that depends only on the language the Turing Machine accepts must be undecidable'.*

5.3.1 Reductions

✓ P_1 reduces to P_2

If there is an algorithm used to convert instance of a problem P_1 to instances of a problem P_2 with the same answer, then it is said that P_1 reduces to P_2.

Thus if P_1 is not recursive, then P_2 cannot be recursive.

If P_2 is non–RE, then P_1 cannot be RE.

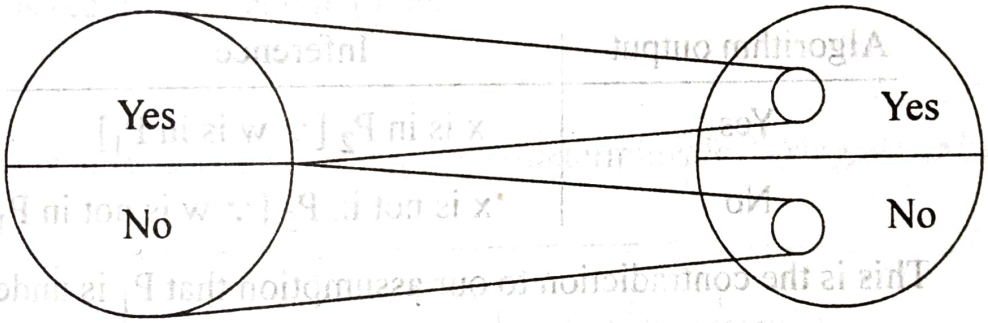

Fig.5.9 Reduction from P_1 to P_2

So, Any instance of P_1 that has 'Yes' answer can be turned into an instance of P_2 with a 'Yes' answer and every instance of P_1 with 'No' answer can be turned into an instance of P_2 with 'No' answer.

It is true that only a small fraction of P_2 is a target of the reduction.

A reduction can be defined as

'A reduction from P_1 to P_2 is Turing Machine that takes an instance of P_1 written on its tape and halts with an instance of P_2 on its tape'.

Theorem : 5.6

If there is a reduction from P_1 to P_2 then

(i) If P_1 is undecidable, then So is P_2

(ii) If P_1 is non–RE, then So is P_2

Proof :

(i) If P_1 is undecidable, then So is P_2.

Assume P_1 is undecidable. Let A be the algorithm which converts the instances of P_1 to instances of P_2.

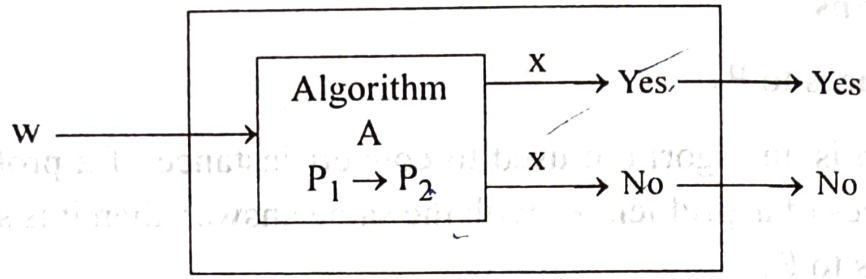

Fig.5.10 Reduction from P_1 to P_2 (Undecidable)

From Fig.(5.10), suppose w be the instance of P_1, given to the algorithm A that converts w into an instance x of P_2.

Algorithm output	Inference
Yes	x is in P_2 [$\because$ w is in P_1]
No	x is not in P_2 [$\because$ w is not in P_1]

This is the contradiction to our assumption that P_1 is undecidable.

Thus, If P_1 is undecidable, then P_2 is also undecidable.

(ii) If P_1 is non–RE, then So is RE

Assume that P_1 is non–RE, but P_2 is RE. Since P_2 is RE, have an algorithm to reduce P_1 to P_2 that is there is a Turing Machine gives 'Yes' output if its input is in P_2 but may not halt if its input is not in P_2.

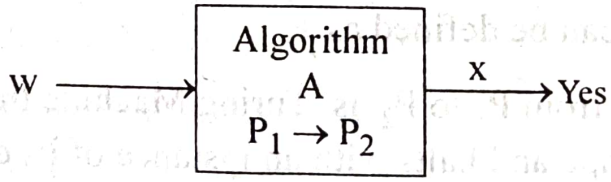

Fig.5.11 Reduction from P_1 to P_2 (RE)

If w is in P_1, then x is in P_2, So the TM will accept w.

If w is not in P_1, then x is not in P_2, So the TM may or may not halt but will not accept w.

This is a contradiction to our assumption.

Thus if P_1 is non–RE, then P_2 is non RE.

5.3.2 Turing Machines that Accept the Empty Language

There are two languages involving Turing Machines called L_e and L_{ne}, where L_e is the empty language and L_{ne} is the non–empty language.

If $L(M_i) = \phi$, M_i does not accept any input, then w is in L_e.

The language consisting of those encoded TM's whose language is empty is called empty language L_e.

If $L(M_i) \neq \phi$, then that language is called non-empty language L_{ne}

$\therefore \quad L_e = \{M \mid L(M) = \phi\}$

$L_{ne} = \{M \mid L(M) \neq \phi\}$

Theorem : 5.7

L_{ne} is recursively enumerable.

Proof :

The construction is based on a non–deterministic Turing Machine.

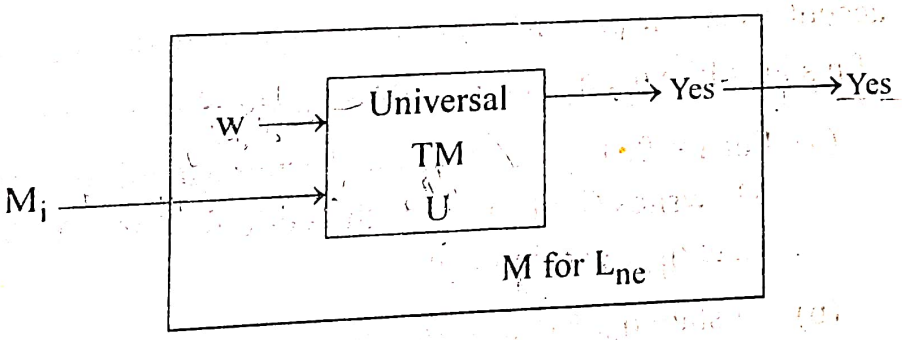

Fig.5.12 Construction of L_{ne} (M)

The theorem is proved at follows.

(i) A Turing Machine code M_i is given as input to the TM.

(ii) M guesses an input w in a right way that M_i might accept w_i.

(iii) M is simulated to the Universal Turing Machine U, which tests whether M_i accepts w.

(iv) If M_i accepts w, then M accepts w.

Thus M_i accepts any string w_i that will be guessed right by the Turing Machine M. If $L(M_i) = \phi$, then no guess is made by Turing Machine M, So M does not accept M_i.

Thus $L(M) = L_{ne}$

Theorem : 5.8

L_{ne} is not recursive.

Proof :

The algorithm of this theorem should be designed in such a way that it converts an input that is a binary–coded pair (M,w) into a Turing Machine M^1 such that $L(M^1) = \phi$ if and only if M accepts input w.

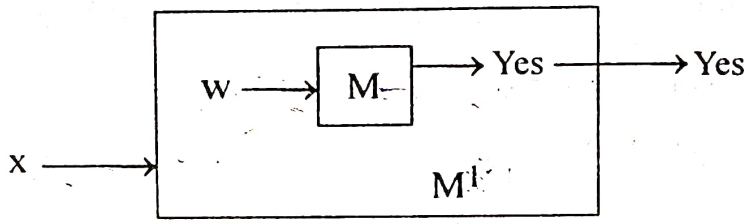

Fig.5.13 Algorithm of L_{ne} about recursiveness

The Turing Machine M^1 is designed to perform the below operations.

1. M^1 ignores its own input x rather it replaces its input by w, the input string accepted by Turing Machine M. M^1 is designed to accept a specific pair (M,w) whose length is n, having a sequence of n states like $q_0, q_1 \ldots q_n$, where q_0 is the start state.
   (a) For i = 0, 1 . . . n–1, if the Turing Machine is at state q_i, M^1 writes (i +1)st bit of the code for (M,w) and goes to state q_{i+1} moving right.
   (b) In state q_n, M^1 moves right by replacing any nonblanks to blanks.
2. When the Turing Machine M^1 reaches a blank in state q_n, it uses a similar collection of states to reposition its head at the left end of the tape.
3. M^1 stimulates a universal Turing Machine U on its present tape. If U accepts, then M^1 accepts. if U does not accept, M^1 never accepts. The simulation made here is by reduction of L_u to L_{ne}.

Assume L_{ne} is recursive. Then the algorithm for L_u is as follows.

1. Convert (M,w) to the Turing Machine M^1 by the reduction of L_u to Line.
2. By the hypothetical algorithm of Line, Tell

(i) if $L(M^1) \neq \phi$, M does not accept w

(ii) If $(M^1) \neq \phi$, M accepts w.

By the algorithm of L_u, this is not true. Our assumption is contradictory and conclude that L_{ne} is not recursive.

5.3.3 Rice's Theorem and Properties of RE Languages

All nontrivial properties of the RE languages are undecidable i.e., It is not possible to recognize the property by a Turing Machine.

Eg : the property of the RE language which is the language is context–free.

A property of RE language is a set of RE language. The properties like

(i) Context–free – the set of all CFL's

(ii) empty – the set consisting of empty language.

A property is *trivial* if it is either empty which is satisfied by no language or is all RE languages, or else it is nontrivial.

Rice's Theorem :

Theorem : 5.9

Every non–trivial property of the RE language is undecidable.

Proof :

Let ρ be a non–trivial property of the RE languages.

Assume that ϕ is not in ρ. Since ρ is nontrivial, there must be some nonempty language L that is in ρ. Here M_L be a TM accepting L. This is the reduction of L_u to L_ρ, where Lu is undecidable.

The design of M^1 is

(i) if $L(M^1) = \phi$, if M does not accept *w*

(ii) If $(M^1) = L$, if M accepts *w*.

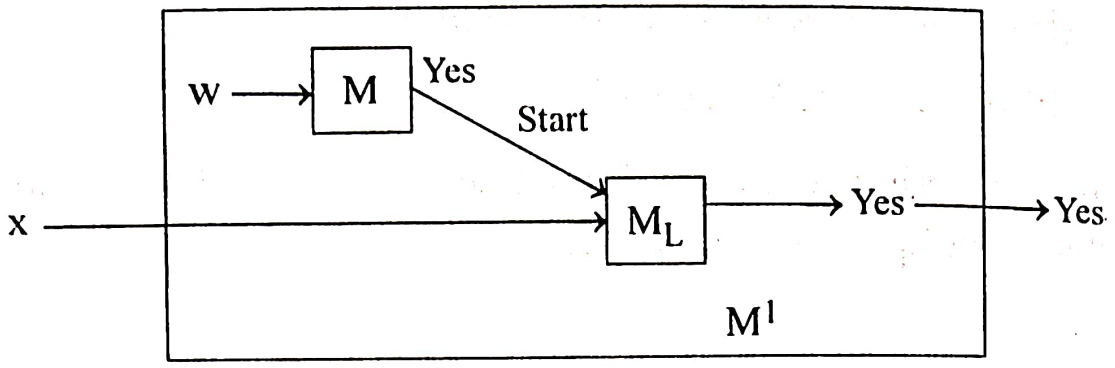

Fig.5.14 M^1 used in Rice's Theorem

M^1 is a two–tape TM. One tape is used to simulate M on *w* and another tape is used to simulate M_L on input x to M^1.

The TM M^1 is constructed as follows.

(i) M^1 ignores its input and simulates M on w, If M does not accept w, then M^1 does not accept x. If M accepts w, then M^1 simulates M_L on x, accepting x if and only if M_L accepts x, Thus M^1 either accepts ϕ or L depending on M.

It is possible to use the hypothetical M_ρ to determine if $L(M^1)$ is in ρ. Since $L(M^1)$ is in ρ if and only if $\langle M,w\rangle$ is in L_u, we have an algorithm for L_u, a contradiction. Thus ρ must be undecidable.

The consequence of the Rice's theorem tells that

(a) the following properties of RE sets are not decidable.

(i) emptiness

(ii) finiteness

(iii) regularity

(iv) context–freedom

(b) The following properties of RE sets are RE

(i) $L \neq \phi$

(ii) *w* is in L for some fixed word w

SOLVED PROBLEMS

1. By Rice's theorem, the following problems are decidable. However tell whether they are RE or non RE?

(a) L(M) Contains atleast two strings.

Recursively Enumerable.

(b) L(M) is infinite

Non-Recursively Enumerable

(c) L(M) is context–free

Recursively Enumerable.

2. Tell whether each of the following are recursive, RE–but–not–recursive, or non–RE.

(a) The set of all TM codes for TM's that halt on every input.

Recursive

(b) The set of all TM codes for TM's that halt on no input.

Recursive

(c) The set of all TM codes for TM's that halt on atleast one input.

Recursively Enumerable but not Recursive

(d) The set of all TM codes for TM's that fail to halt on atleast one input.

Non-Recursively Enumerable

5.4 POST'S CORRESPONDENCE PROBLEM

In this, the undecidable problems about Turing Machines are reduced to undecidable problems about real things. The goal is to prove that problem about strings to be undecidable.

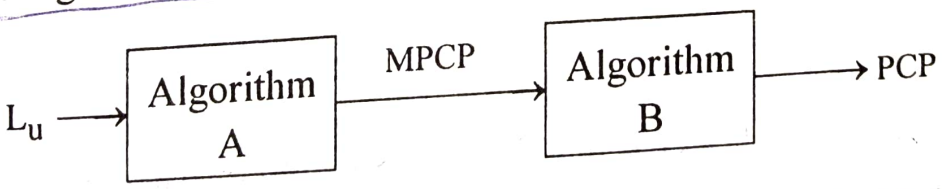

Fig.5.15 Reduction of L_u to PCP to prove the undecidability

We reduce L_u to modified PCP then to PCP using algorithm.

5.4.1 Definition

An instance of Post's correspondence problem (PCP) consists of two lists of strings over Σ

$A = w_1, w_2, \ldots w_k$

$B = x_1, x_2, \ldots x_k$ for some integer k.

The instance of PCP has a solution if there is any sequence of integers $i_1, i_2, \ldots i_m$ with $m \geq 1$ such that

$$w_{i_1}, w_{i_2}, w_{i_3} \ldots w_{i_m} = x_1, x_2, \ldots x_k$$

is a solution to this instance of PCP.

Example : 1

Let $\Sigma = \{0, 1\}$. Let A and B be strings. Find the instance of PCP.

	List A	List B
i	w_i	x_i
1	1	111
2	10111	10
3	10	0

Solution :

Take M = 4.

Take this combination 2, 1, 1, 3

By concatenating strings in this series

$$w_2\, w_1\, w_1\, w_3 = x_2\, x_1\, x_1\, x_3$$

$$101111110 = 101111110$$

$\therefore$ instance of PCP = 2, 1, 1, 3.

Example : 2

Let $\Sigma = \{0, 1\}$. Let A and B be lists of three strings each. Find the instance.

	List A	List B
i	w_i	x_i
1	10	101
2	011	11
3	101	011

Suppose the instance has a solution $i_1, i_2, \ldots i_m$. No string beginning with instances of w match with instances of x.

$$w_1 w_3 \neq x_1 x_3$$

$$10101 \neq 101011$$

Thus this instance of PCP has no solution.

5.4.2 Modified PCP

In order to simplify the reduction of L_u to PCP, an intermediate version of PCP called Modified Post's correspondence problem is used.

The ***Modified PCP*** is the following

Given lists A and B, of k strings each from Σ^*

$$A = w_1, w_2, \ldots w_k$$

$$B = x_1, x_2, \ldots x_k$$

It has the solution $i_1, i_2, \ldots i_r$ such that

$$w_1, w_{i_1}, w_{i_2} \ldots w_{i_r} = x_1 x_{i_1} x_{i_2} \ldots x_{i_r}$$

The difference between the MPCP and PCP is that in the MPCP, a solution is required tobe stored with the first string on each list.

Example :

Let $\Sigma = \{0, 1\}$. Let A and B be the list of strings defined as

	List A	List B
i	w_i	x_i
1	1	10
2	110	0
3	0	11

Find the instance of MPCP.

Solution :

MPCP is given by

$$w_1, w_{i_1}, w_{i_2} \ldots w_{i_r} = x_1 x_{i_1} x_{i_2} \ldots x_{i_r}$$

$$10110 = 10110$$

Where the instance is given 1, 3, 2.

Construction of PCP from MPCP

Let the lists of MPCP be

$$A = w_1, w_2, \ldots w_k \qquad B = x_1, x_2, \ldots x_k$$

Assume $*$ and \$ are symbols not present in the alphabet Σ of the MPCP instance.

Construct a new PCP instance

$$C = y_0, y_1, \ldots y_{k+1} \qquad D = z_0, z_1, \ldots z_{k+1}$$

designed as follows.

1. For i = 1, 2, . . . k, Let y_i be w_i with a * after each symbol of w_i and Let z_i be x_i with a * before each symbol of x_i.
2. $y_0 = *y_1$ and $z_0 = z_1$.

   The 0^{th} pair looks like pair 1, except that there is an extra * at the beginning of the string from the first list. Since we are reducing from MPCP, the 0^{th} pair will be the only pair in the PCP instance where both strings begin with the same symbol. So any solution to this PCP instance begins with index 0.
3. $y_{k+1} = \$$ and $z_{k+1} = *\$$

Theorem :5.10

MPCP reduces to PCP

Proof :

Assume that $i_1, i_2 \ldots i_m$ is a solution to the given MPCP instance with lists A and B.

We know that, according to MPCP

$$w_1 w_{i_1} w_{i_2} \ldots w_{i_m} = x_1 x_{i_1} x_{i_2} \ldots x_{i_m}$$

Replace w's by y's and x's by z's. Then we have the strings like $y_1 y_{i_1} y_{i_2} \ldots y_{i_m}$ and $z_1 z_{i_1} z_{i_1} \ldots z_{i_m}$. The only difference is that the first string would be missing a $*$ at the beginning and the second string would be missing a $*$ at the end.

$$* y_1 y_{i_1} y_{i_2} \ldots y_{i_m} = z_1 z_{i_1} z_{i_2} \ldots z_{i_m} *$$

Using the construction rule,

Put $y_0 = *y_1$ and $z_0 = z1$. Then fix the initial * by replacing the first index by 0.

$$y_0 y_{i_1} y_{i_2} \ldots y_{i_m} = z_0 z_{i_1} z_{i_1} \ldots z_{i_m} *$$

Append the index k+1, i.e., $y_{k+1} = \$$ and $z_{k+1} = *\$$

$$\therefore \quad y_0 y_{i_1} y_{i_2} \ldots y_{i_m} y_{k+1} = z_0 z_{i_1} z_{i_2} \ldots z_{i_m} z_{k+1}$$

Thus 0, i_1, $i_2 \ldots i_m$, k+1 is a solution to the instance of PCP.

For MPCP, i_1, $i_2 \ldots i_m$ is a solution. If we remove the *'s and the final \$ from the string $y_0 y_{i_1} y_{i_2} \ldots y_{i_m} y_{k+1}$, we get the string

$$w_1 w_{i_1} w_{i_2} \ldots w_{i_m}$$

If we remove *'s and \$ from the string, $z_0 z_{i_1} z_{i_2} \ldots z_{i_m} z_{k1}$, we get

$$x_1 x_{i_1} x_{i_2} \ldots x_{i_m}$$

So for PCP

$$y_0 y_{i_1} y_{i_2} \ldots y_{i_m} y_{k+1} = z_0 z_{i_1} z_{i_2} \ldots z_{i_m} z_{k+1}$$

which follows that

$$w_1 w_{i_1} w_{i_2} \ldots w_{i_m} = x_1 x_{i_1} x_{i_2} \ldots x_{i_m}$$

Thus a solution to the PCP instance implies a solution to the MPCP instance. Thus there is a reduction of MPCP to PCP, which confirms that if PCP were decidable, MPCP would also be decidable.

Undecidability of PCP

Given a pair (M,w), construct an instance (A, B) of MPCP such that TM M accepts input w if and only if (A, B) has a solution.

The idea is that MPCP instance (A, B) simulates in its partial solutions, the computation of M on input w. That is, partial solutions will consist of strings that are prefixes of the sequence of ID's of M :

$\# \alpha_1 \# \alpha_2 \# \alpha_3 \# \ldots .$

where α_1 is the initial ID of M with input w and

$\alpha_i \vdash \alpha_{i+1}$ for all ;

The string from the B list will always be one ID ahead of the string from the A list, unless M enters an accepting state.

The construction of MPCP instance is designed by using the theorem of semi–infinite tapes which says that the turing machines head never prints a blank and never moves left from its initial head position.

So an ID of a Turing machine will always be a string of the form αqB where α and β are strings of nonblank symbols and q is a state. It is allowed that β to be empty if the head is at the blank immediately to the right of α, rather than placing a blank to the right of the state.

Let $M = (Q, \Sigma, \vdash, \delta, q_0, B, F)$ be a TM and let w in Σ^* be an input string. An instance of MPCP is constructed as follows.

1. The first pair is

List A	List B
#	$\# q_0 w \#$

The remaining pairs are grouped as follows.

2. Tape symbols and the separator # can be appended to both lists

List A	List B
X	X for each X in ⊢
#	#

3. For each q in Q–F, p in Q, and x, y, z in ⊢

List A	List B	
qX	Yp	if $\delta(q, X) = (p, Y, R)$
ZqX	pZY	if $\delta(q, X) = (p, Y, L)$
q#	Yp#	if $\delta(q, B) = (p, Y, R)$
Zq#	pZY#	if $\delta(q, B) = (p, Y, L)$

4. For each q in F, and X and Y in ⊢

List A	List B
XqY	q
Xq	q
qY	q

5. For each q in F

List A	List B
q # #	#

To complete the solution, we have to use the five kinds of pairs generated.

Example ***Let us convert the TM***

$M = \{q_1, q_2, q_3\}, \{0, 1\}, \{0, 1, B\}, \delta, q_1, B, \{q_3\}$

where δ is given by :

q_i	$\delta(q_i, 0)$	$\delta(q_i, 1)$	$\delta(q_i, B)$
q_1	$(q_2, 1, R)$	$(q_2, 0, L)$	$(q_2, 1, L)$
q_2	$(q_3, 0, L)$	$(q_1, 0, R)$	$(q_2, 0, R)$
q_3	—	—	—

and input string be w = 01. Find the solution.

Solution :

(i) Start with the first pair

A : #

B : # $q_1$01 #

(ii) MPCP instance is constructed as follows

Rule	List A	List B	Source
1.	#	# q_1 01 #	
2.	0	0	
	1	1	
	#	#	
3.	$q_1 0$	$1 q_2$	from $\delta(q_1, 0) = (q_2, 1, R)$
	$0 q_1 1$	$q_2 00$	from $\delta(q_1, 1) = (q_2, 0, L)$
	$1 q_1 1$	$q_2 10$	
	$0 q_1 \#$	$q_2 01 \#$	from $\delta(q_1, B) = (q_2, 1, L)$
	$1 q_1 \#$	$q_2 11 \#$	
	$0 q_2 0$	$q_3 00 \#$	from $\delta(q_2, 0) = (q_3, 0, L)$
	$1 q_2 0$	$q_3 10 \#$	
	$q_2 1$	$0 q_1$	from $\delta(q_2, 1) = (q_1, 0, R)$
	$q_2 \#$	$0 q_2 \#$	from $\delta(q_2, B) = (q_2, 0, R)$
4.	$0 q_3 0$	q_3	
	$0 q_3 1$	q_3	
	$1 q_3 0$	q_3	
	$1 q_3 1$	q_3	
	$0 q_3$	q_3	
	$1 q_3$	q_3	
	$q_3 0$	q_3	
	$q_3 1$	q_3	
5.	$q_3 \# \#$	#	

M accepts the input 01 by the sequence of moves

$q_1 01 \vdash 1 q_2 1 \vdash 10 q_1 \# \vdash 1 q_2 01 \# \vdash q_3 101$

The following computation of M leads to the partial solution.

$\# q_1 01 \# 1 q_2 1 \# 10 q_1 \# 1 q_2 01 \# q_3 101 \# q_3 \# \# q_3 \# \#$

The first pair is given by

A : #

B : # q_1 01 #

The only way to extend the partial solution is for the string from the A list to be a prefix of the remainder, q_1 01 #.

From rule(3), use the pair (q_1 0, 1q_2) to the first pair. Then the partial solution is given by

A : # q_1 0

B : # q_1 01 # 1q_2

This pair can be further extended by copying pairs from rule (2) until getting the state in the second ID.

A : # q_1 0 1 # 1

B : # q_1 0 1 # 1 q_2 1 # 1

To simulate an another move, the appropriate pair is selected from rule(3) i.e., ($q_2$1, 0q_1).

A : # q_1 01 # 1q_2 1

B : # q_1 01 # 1 q_2 1 # 10 q_1

Next use rule(2) pairs to copy the next three symbols : #, 1, 0. But to go that for would be a mistake, since the next move of M moves the head left, and the 0 just before the state is needed in the next rule(3) pair.

Thus, copy the next two symbols, leaving the partial solution.

A : # q_1 01 # 1 q_2 1 # 1

B : # q_1 01 # 1 q_2 1 # 10 q_1 # 1

Next use the pair (0 q_1#, q_2 01 #) from rule (3).

A : # q_1 01 # 1 q_2 1 # 10 q_1#

B : # q_1 01 # 1 q_2 1 # 10 q_1 # 1q_2 01 #

Next use another rule (3) pair ($1q_20$, q_310) which leads to acceptance.

A : # q_1 01 # 1 q_2 1 # 10 q_1# 1 q_2 0

B : # q_1 01 # 1 q_2 1 # 10 q_1 # 1q_2 01 # q_3 10

Next use pairs from rule (4) to eliminate all but q_3 from the ID. Next use rule (2) to copy symbols as necessary.

A : # q_1 01 # 1 q_2 1 # 10 q_1# 1 q_2 01 # q_3 101 # q_3 01 # q_3 1 #

B : # q_1 01 # 1 q_2 1 # 10 q_1 # 1q_2 01 # q_3 101 # q_3 01 # q_3 1 # q_3 #

Finally use the pair (q_3 # #, #) from rule (5) to finish the solution.

The complete solution is given by

A : # q_1 01 # 1 q_2 1 # 10 q_1# 1 q_2 01 # q_3 101 # q_3 01 # q_3 1 # q_3 # #

B : # q_1 01 # 1 q_2 1 # 10 q_1 # 1q_2 01 # q_3 101 # q_3 01 # q_3 1 # q_3 # #

Theorem :5.11

Post's correspondence problem is undecidable.

Proof :

The proof of this theorem is telling how to reduce L_u to MPCP. It can be proved using the statement.

"M accepts w if and only if the constructed MPCP instance has a solution".

if part : MPCP instance has a solution

To prove : M accepts w.

Assume a partial solution begins with

A : #

B : # q_0w #

States and the tape symbols can only be handled by the pairs of rule(3) and all other tape symbols and # must be handled by pairs from rule (2). After reaching the accepting states, the rules (4) and (5) can be used. Thus, unless M reaches an accepting state, all partial solutions have the form.

A : x

B : xy

Where

x – sequence of ID's of M representing a computation of M on input w possibly followed by # and the beginning of the next ID α.

y – Completion of α, # and the beginning of the ID that follows α, upto the point that x ended within α itself.

Thus as long as M does not enter an accepting state, the partial solution is not a solution and also B is longer than A.

Only–if Part : M accepts w

If w is in L(M), then start with the pair from rule (1) and simulate the computation of M on w. And using the rule(3) pair to copy the state from each ID and simulate one move of M. Then by using rule(2), to copy tape symbols and the marker # as needed. If M reaches an accepting state, then the pairs from rule (4) and rule (5) allow the A string to catch upto the B string and form a solution.

Thus if there is a solution M must enter an accepting state and so M accepts w.

SOLVED PROBLEM

1. Find the solution for the following PCP problems.

(a)

i	A	B
1	01	011
2	001	10
3	10	00

Let the instances be 1, 3, 2

$A_{i_1}, A_{i_3}, A_{i_2} = B_{i_1}, B_{i_3}, B_{i_2}$

$0110001 \neq 0110010$ (not equal)

$\therefore$ there is no solution for this problem

(b)

i	A	B
1	01	011
2	001	01
3	10	00

Let the instances be 1, 3, 2

$$A_{i_1}, A_{i_3}, A_{i_2} = B_{i_1}, B_{i_3}, B_{i_2}$$

$$0110001 \neq 0110001 \text{ (equal)}$$

$\therefore$ the solution is 1, 3, 2

(c)

i	A	B
1	ab	bc
2	a	ab
3	bc	ca
4	c	a

Let the instances be 2, 3, 1, 4

$$A_{i_2}, A_{i_3}, A_{i_1}, A_{i_4} = B_{i_1}, B_{i_3}, B_{i_1}, B_{i_4}$$

$$abcabc \neq abcabca$$

$\therefore$ It has no solution.

Sai Ram

5.5 THE CLASSES P AND NP

The problems solvable in polynomial time on a typical computer are exactly the same as the problems solvable in polynomial time on a Turing Machine. The problems which cannot be solvable in polynomial time are called *'intractable problems'*.

5.5.1 Problems solvable in Polynomial Time

The time complexity or running time T(n) of a Turing Machine is after taking the input w of length n, M halts after making atmost T(n) moves, regardless of whether or not M accepts.

For Eg. The complexity of bubble sort algorithm is $T(n) = n^2$.

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

Example : Kruskal's Algorithm

The problems with efficient solutions are all comes under the class P. Consider one such problem MWST (Minimum Weight Spanning Tree).

Consider a graph with n nodes and e edges. Each edge has a weight represented in integers. A spanning tree is a subset of the edges such that all

nodes are connected through edges where there are no cycles. A minimum–weight spanning tree has the least possible total edge weight of all spanning trees.

A well known greedy algorithm is there called Kruskal's Algorithm for finding a MWST.

Algorithm :

1. Maintain the 'connected component' for each node. Initially every node is a connected component by itself.
2. Sort the edges in ascending order of their weights.
3. Select the lowest–weight edge that has not yet been visited.
   - 3.1 Select that edge for the spanning tree.
   - 3.2 Merge the two connected components involved.
4. Repeat the step (3) until all edges have been considered by which the number of edges selected is one less than the number of nodes.

Example

Consider the graph given below. Find MWST.

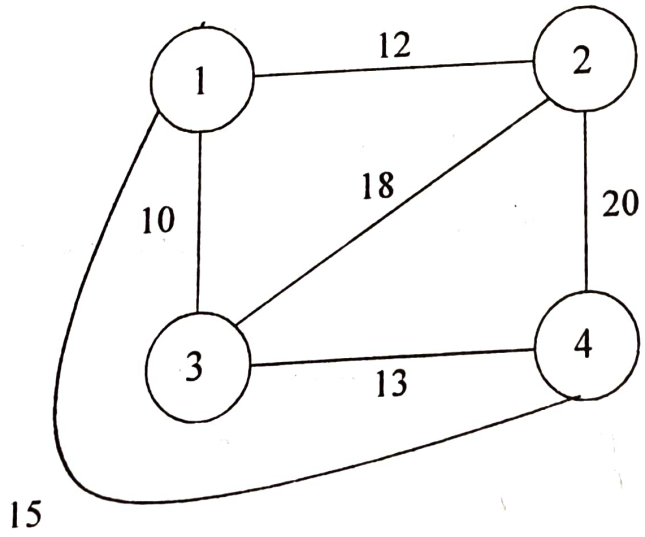

Fig.5.16

Solution :

The ascending order of their weights are

$w = \{10, 12, 13, 15, 18, 20\}$

Number of nodes $N = 4$

1. Select the edge (1,3) with weight 10

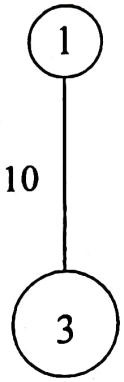

Fig.5.17

2. Select the second lowest weight from the list and check whether it forms any cycle or already included in the list. If it is not so, include that edge to the forest (group of edges).

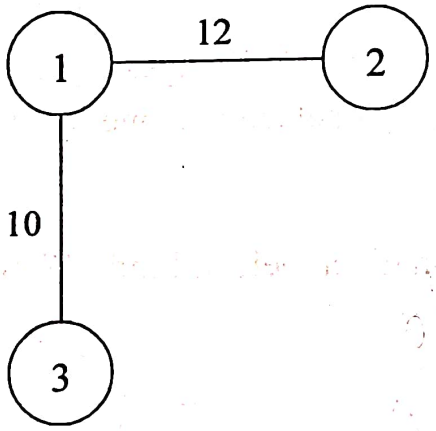

Fig.5.18

3. Like that repeat the steps (2) and (3) till all the nodes are visited.

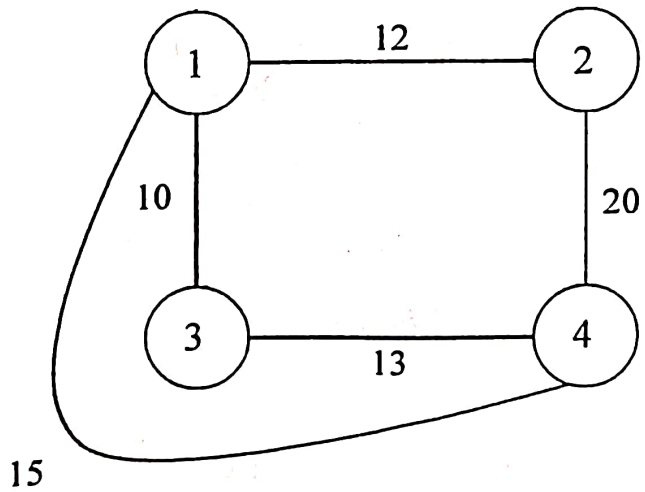

Fig.5.19

$\therefore$ Total weight = 10 + 12 + 13 = 35

It is possible to implement this algorithm using a computer with a graph of m nodes and e edges in time 0(m + e log e). The edges are selected in 0(e) time and the components of the two nodes connected by the edges are found out in 0(m) time. The total time taken by this algorithm is 0(e(e+m). Thus the algorithm is polynomial in size.

Implementation of Kruskal's Algorithm using Turing Machines

While translating the logic of Kruskal's algorithm to the Turing Machine, we encounter several problems. Some of them are as follows :

- ✓ The output of this MWST is the list of edges. But using Turing Machine, it is possible to produce only two answers like yes or no i.e., accept or reject. So the problem can be refined as 'Given the graph G and weights w, does G has a spanning tree of weight w or less ? So It is hard to compute the complete solution rather than yes/no type.
- ✓ The input of this algorithm is given by the size of a graph, as the number of nodes or edges. But using TM, it is a string over a finite alphabet. So the problem is encoded in a suitable manner. There are two reasons why selecting the inputs differently.
  a. The input size is logarithm of its size, so what can be done in polynomial time using 1 measure can be done in polynomial time using the other measure.
  b. The length of a string representing the input is a measure of the number of bytes a real computer has to read to get its input.

A node represented by an integer is proportional to the logarithm of the integers size and is not '1 byte for any node'.

Example

Consider the graph in Fig.5.16. Find the solution using TM. The inputs are given as code for the graph and weight limits. The code has five symbols like 0, 1, the left paranthesis, right parenthesis and comma.

Solution :

1. Assign integers to all the nodes (m)

2. Begin the code with m in binary and the weight limit w in binary, separated by a comma.

3. If there is an edge between nodes i and j with weight w, place (i, j, w) in the code. The integers i, j and w are coded in binary. The order of placing an edge within the code is not a matter.

Thus, one of the possible codes for the graph with limit w = 35 is 100, 100011 (1, 10, 1100) (1, 11, 1010) (1, 100, 1111) (10, 11, 10010) (10, 100, 10100)

Construction of Turing Machine

The running time of Kruskal's algorithm implemented in a TM is $O(n^2)$. It is constructed by a multitape TM.

1. One tape is used to store the nodes and their current component numbers. The length of the tape is O(n).

2. Another tape is used to hold the least edge–weight while scanning the edges on the input tape. The second track on that tape is used to mark those edges that were selected as the edge of least remaining weight. To scan the lowest–weight, it takes O(n) time.

3. After selecting an edge, place its two nodes on another tape. To search the nodes and components of these nodes, it tracks O(n) time.

4. A tape can be used to hold the two components, i and j being merged when an edge is found to connect two previously unconnected components. Then we need to scan the table and each component to be changed to j which takes O(n) time.

The MWST problem is restated as 'Does graph G has a MWST of total weight W or less", comes under class P.

5.5.2 Non–deterministic Polynomial Time

A language L is in the class NP(nondeterministic polynomial) if there is a non–determinisitc TM M and a polynomial time complexity T(n) such that $L = L(M)$. But it is true that $P \subseteq NP$.

But it appears that NP contains many problems not in P. The reason is that a NTM running in polynomial time has the ability to guess an exponential number of possible solutions to a problem and check each one in polynomial time.

Example : Traveling Salesman Problem

Given a graph G = (V, E) where V – number of vertices and E – the edge connecting two vertices (v_1, v_2). A salesperson has to visit each vertex exactly once and returns to its starting point with a minimum distance.

This can be implemented by a TM with a question that 'whether the graph has a Hamilton circuit of total weight at most w ? A Hamilton circuit is the set of edges that the nodes into a single cycle, with each node appearing once.

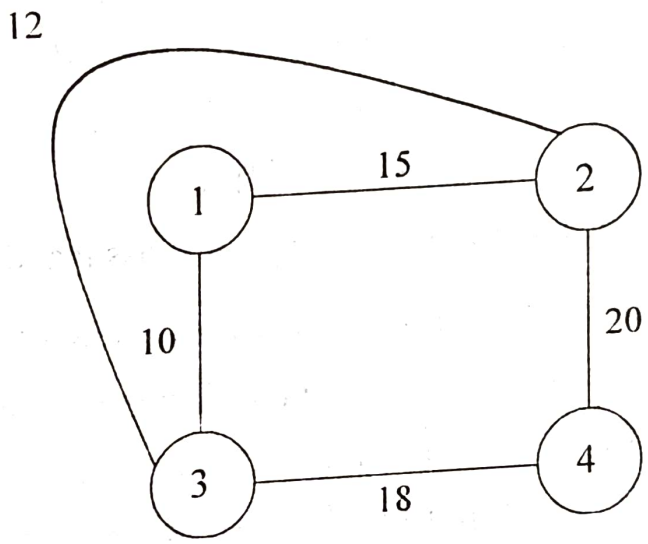

Fig.5.20

In this graph, we have only 1 Hamilton circuit (1, 2, 4, 3, 1)

Where the weight is given by W = 15 + 20 + 18 + 10 = 63.

Thus if $w \geq 63$, the answer is Yes

if $w < 63$, the answer is No.

For a M–node graph, the number of distinct cycles grows as $0(M^1)$. For a non–deterministic computer, it is possible to guess a permutation of the nodes and compute the total weight for the cycle of nodes in that order. So using a multitape NTM. It is possible to guess a permutation in $0(n^2)$ steps and checks its total weight in the same time. Thus TSP comes under the class NP.

5.5.3 Polynomial Time Reductions

A problem P cannot be solved in polynomial time can be proved by the reduction of a problem P_1 which is under class P to P_2.

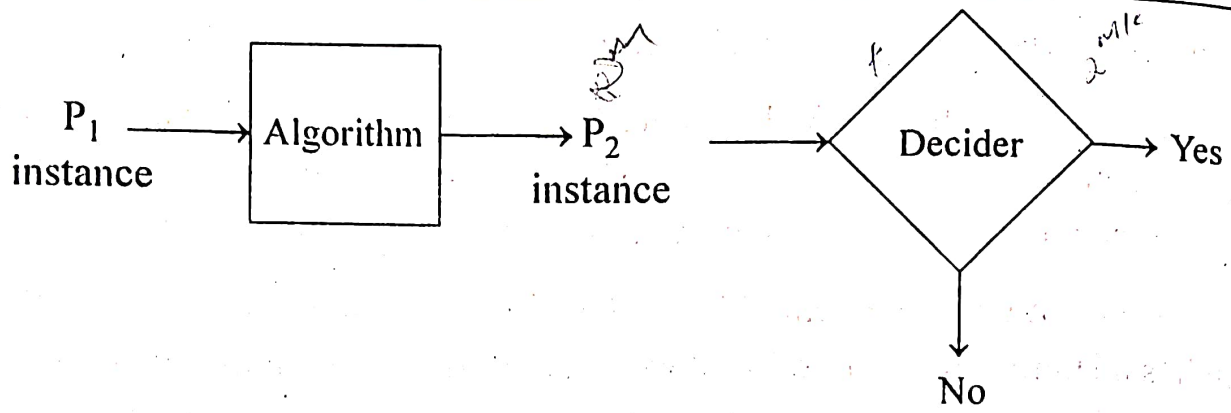

Fig.5.21 Reduction of P_1(class P) to P_2 (class P)

When an instance P_1 of length M is given, the algorithm produces an output string of length 2^M. Which is then given to the hypothetical polynomial– time algorithm for P_2. If the decision algorithm run in $0(n^k)$ time, then on an input of length 2^M, it may take $0(2^{kM})$ which is exponential in M. Thus it is consistent with the situation where P_2 is in class P and P_1 is not in P.

The right way of translation from P_1 to P_2 is that it requires time that is polynomial in the length of its input. If the input instance takes the time of $0(M^j)$ of length M, then the output instance of P_2 cannot be longer than the number of steps taken i.e,. it is at most CM^j for some constant C. Then it is easy to prove that if P_2 is in class P_1 then so is P_1. Suppose the membership in P_2 of a string of length n can be decided by $0(n^k)$. Likethat the membership in P1 can be decided by $0(M^j+(CM^j)^k)$ time for a length of M.

5.5.6 NP– Complete Problems

A language L is NP–complete if it satisfies the following conditions

1. L is in NP
2. For every language L' in NP there is a polynomial–time reduction of L' to L.

Eg : Traveling Salesman Problem.

SUMMARY

- ☑ ***Decidability :*** The problems which can be solved using a definite algorithm are called decidable problems.
- ☑ ***Recursive and Recursively Enumerable Languages :*** The languages accepted by Turing Machines are called recursively enumerable and the subset of RE languages that are accepted by a TM that always halts are called recursive.
- ☑ ***Undecidability :*** If a language is not recursive, then those problems are called undecidable problems.
- ☑ ***Diagonalization Language :*** The set of strings of 0's and 1's which are not in the language of that TM.
- ☑ ***Universal Language :*** The language which consists of strings by which the TM accepts that input. L_u is RE but not recursive.
- ☑ ***Properties of RE sets :*** The properties of RE sets which are undecidable are emptiness, finiteness, regularity, context–freedom.
- ☑ ***Rice's Theorem :*** Any nontrivial property of the languages accepted by Turing Machines is undecidable.
- ☑ ***Post's Correspondence Problem :*** Given two lists of the same number of strings. By PCP, it is decided whether we can pick a sequence of corresponding strings from the two lists and form the same string by concatenation.