UNIT V
UNDECIDABILITY

A language that is not Recursively Enumerable (RE) – An undecidable problem that is RE – Undecidable problems about Turing Machine – Post's Correspondence Problem - The classes P and NP.

5.1. BASIC DEFINITIONS

5.1.1 Decidable Problem:

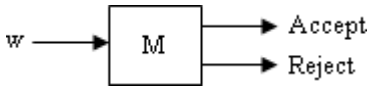
If and only if there exists an algorithm to solve the problem in finite time and determine whether the answer is 'yes' or 'no'.

5.1.2. Undecidable Problem:

If and only if there exists no algorithm to solve the problem in infinite time and determine whether the answer is 'yes' or 'no'.

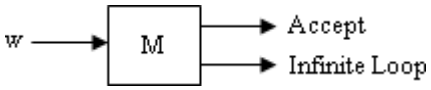
5.1.3. Recursive Language:

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



5.1.4. Recursively Enumerable Language:

A language is recursively enumerable if there exists a TM that accepts every string of the language, and does not accept strings that are not in the language. The strings that are not in the language may be rejected and it may cause the TM to go to an infinite loop.



5.2. A LANGUAGE THAT IS NOT RECURSIVELY ENUMERABLE

A language L is recursively enumerable if L = L(M) for some TM M. Recursive or decidable languages that are not only recursively enumerable, but are accepted by a TM.

To prove undecidable, the language consisting of pairs(M, w) such that;

(1) M is a Turing Machine (suitably coded, in binary) with input alphabet {0, 1}.
(2) 'w' is a string of 0's and 1's.
(3) M accepts input 'w'.

If this problem with inputs restricted to the binary alphabet is undecidable, then surely the more general problem, where TM's may have any alphabet, is undecidable.

5.2.1. Enumerating the Binary Strings:

To assign integers to all the binary strings so that each string corresponds to one integer, and each integer corresponds to one string.

5.2.2. Codes for Turing Machines:

A binary code for Turing Machines so that each TM with input alphabet {0, 1} may be thought of as a binary string.

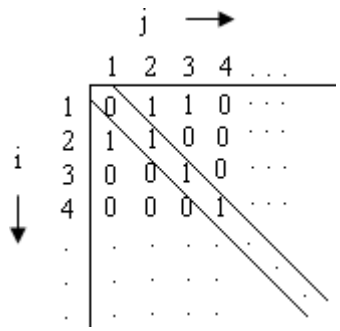
To represent a TM M = (Q, {0, 1}, Γ, δ, q_1, B, F) as a binary string, we must first assign integers to the states, tape symbols and directions L and R.

- We shall assume the states are q_1, q_2, ……., q_k for some 'k'. The start state will always be 'q_1' and 'q_2' will be the only accepting state. Note that, since we may assume the TM halts whenever it enters an accepting state, there is never any need for more than one accepting state.
- We shall assume the tape symbols are X_1,X_2,…..,X_m for some 'm'. X_1 always be the symbol '0', X_2 will be '1' and X_3 will be B. However, other tape symbols can be assigned to the remaining integers arbitrarily.
- We shall refer to direction L as D_1 and direction R as D_2.

Once we have established an integer to represent each state, symbol and direction, we can encode the transition function 'δ'. Suppose one transition rule is $\delta(q_i, X_j) = (q_k, X_l, D_m)$, for some integers i, j, k, l and m. We shall code this rule by the string $0^i1o^j1o^k1o^l1o^m$.

5.2.3. The Diagonalization Language (L_d):

The undecidable problem can be proved by the method of diagonalization. Construct a list of words over (0, 1)* in canonical order, where 'w_i' is the i^{th} word and M_j is TM. This can be represented as a table.



Construct a language L_d using the diagonal entries. The value'0' means, w_i is not in $L(M_j)$ and '1' means w_i is in $L(M_j)$.

To construct L_d, we complement the diagonal. For instance, the complemented diagonal would begin 1,0,0,0,…… The trick 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.

Proof that L_d is not Recursively Enumerable:

Theorem:

L_d is not a recursively enumerable language. That is, there is no Turning Machine that accepts L_d.

Proof:

Suppose Ld were L(M) for some TM M. Since Ld is a language over alphabet {0,1}, M would be in the list of TM's we have constructed, since it includes all TM's with input alphabet {0,1}. Thus, there is atleast one code for M, say 'i', that is, $M=M_i$.

- If w_i is in L_d, then M_i accepts w_i. But then, by definition of L_d, w_i is not in L_d, because L_d contains only those w_i such that M_j does not accept w_j.
- Similarly, if w_i is not in L_d, then M_i does not accept w_i. Thus, by definition of L_d, w_i is in L_d.

Since w_i can neither be in L_d nor fail to be in L_d, we conclude that there is a contradiction of our assumption that M exists. That is, L_d is not a recursively enumerable language.

PROBLEM:

Ex:

Obtain the code for (M, 1011) where $M = (\{q_1, q_2, q_3\}, \{0, 1\}, \{0,1,B\}, \delta, q_1, B, \{q_2\})$, δ is,

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

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

Soln:

1 – represented as two zero's

0 – represented as one zero

L – represented as one zero

R – represented as two zero's

B – represented as threee zero's

(M, 1011) = 0 | 00 | 000 | 0 | 00 || 000 | 0 | 0 | 00 | 00 || 000 | 00 | 00 | 0 | 00 || 000 | 000 | 000 | 00 | 0

5.3. AN UNDECIDABLE PROBLEM THAT IS RE

5.3.1. Recursive Language:

A language L recursive if L=L(M) for some TM M such that;

1. If 'w' is in L, then accepts.
2. If 'w' is not in L, then M eventually halts, although it never enters an accepting state.

If we think of the language L as a "problem" as will be the case frequently, then problem L is called **decidable** if it is a recursive language, and it is called **undecidable** if it is not a recursive language.

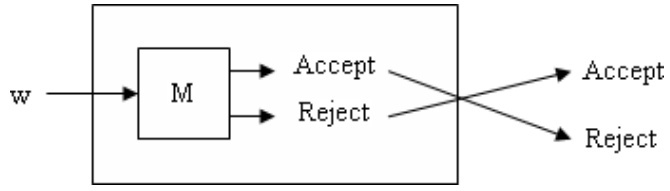
5.3.2. Complements of Recursive and RE languages:

Theorem:

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

Proof:

Let L = L(M) for some TM M that always halts. We construct a TM M such that $\overline{L}$ = L($\overline{M}$) by the construction.



That is, $\overline{M}$ behaves just like M. However, M is modified as follows to create $\overline{M}$;

(1) The accepting states of M are made non accepting states of $\overline{M}$ with no transitions, ie., in these states $\overline{M}$ will halt without accepting.
(2) $\overline{M}$ has a new accepting state 'r'; there are no transitions from 'r'.
(3) For each combination of a non accepting state of M and a tape symbol of M such that M has no transition (ie., M halts without accepting), add a transition to the accepting state 'r'.

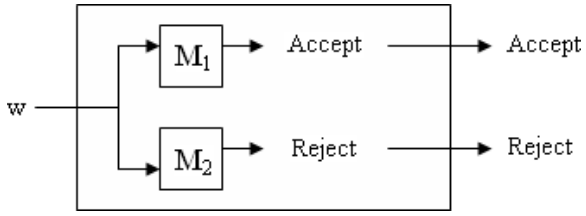
Theroem:

If L and $\overline{L}$ are recursively enumerable then L is recursive.

Proof:

Let M1 be the L and M2 be the $\overline{L}$. Construct M to simulate M1 and M2 simultaneously, since 'w' is either in L or $\overline{L}$.

M accepts 'w' if M1 accepts it and M rejects 'w' if M2 accepts it. Thus M will always say either "Accept" or "reject". Since M is accepts L. Thus L is recursive.



The important consequences of the properties that we have seen are,

If L and $\overline{L}$ are complementary then only one of the following should be true.

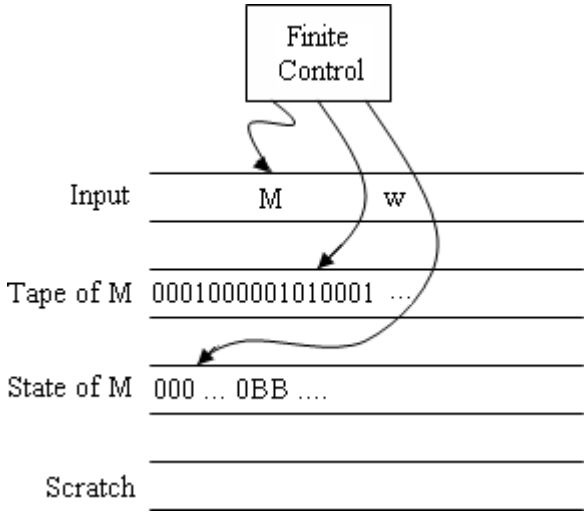
(1) Both L and $\overline{L}$ are recursive.
(2) Neither L nor $\overline{L}$ is recursively enumerable.
(3) One of the L is recursively enumerable and other is not recursively enumerable.

5.3.3. The Universal Language:

We define L_u, the universal language, to be the set of binary strings that encode in the pair (M,w), where M is a TM with the binary input alphabet, and 'w' is a string in (0, 1)*, such that 'w' is in L(M). That is, L_u is the set of strings

representing a TM and an input accepted by that TM. We shall show that there is a TM U, often called the universal Turing Machine such that $L_u = L(U)$.

- It is easiest to describe U as a multitape TM.
- In the case of U, the transitions of M are stored initially on the first tape, doing with the string 'w'.
- A second tape will be used to hold the simulated tape of M, using the same format as for the code of M.
- That is, tape symbol Xi of M will be represented by 0i, and tape symbols will be separated by single 1's.
- The third tape of U holds the state of M, with state qi represented by 'i', 0's.



The operation of U can be summarized as follows;

(1) Examine the input to make sure that the code for M is a legitimate code for some TM. If not, U halts without accepting. Since invalid codes are assumed to represent the TM with no moves, and such a TM accepts to inputs.

(2) Initialize the second tape to contain the input 'w', in its encoded form. That is, for each '0' of 'w', place 10 on the second tape, and for each '1' of 'w', place 100 there. Note that the blanks on the simulated tape of M, which are represented by 1000.

(3) Place '0', the start state of M, on the third tape, and move the head of U's second tape to the first simulated cell.

(4) To simulate a move of M.

(5) If M has no transition that matches the simulated state and tape symbol, then no transition will be found. Thus, M halts in the simulated configuration.
(6) If M enters its accepting state, then U is accepts.

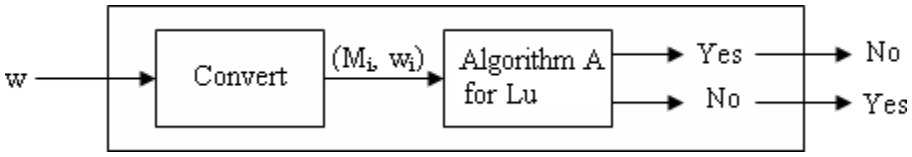
5.3.4. Undecidability of the Universal Language:

Theorem:

Universal Language L_u is not recursive.

Proof:

We know that $\overline{L_d}$ is not recursive, by reducing $\overline{L_d}$ to L_u.



- Assume that L_u is recursive. Then $\overline{L_d}$ must be recursive too.
- Since we know that $\overline{L_d}$ is not recursive we can conclude that L_u is not recursive.
- Construct an algorithm for L_u which accepts (M_i, w_i) if w_i is in $L(M_i)$. Thus we have an $\overline{L_d}$. This is contradiction. Hence L_u is not recursive.

5.4. UNDECIDABLE PROBLEMS ABOUT TURING MACHINES

5.4.1. Turing Machines that accept the Empty language:

In this, we are using two languages, called L_e and L_{ne}. Each consists of binary strings. If 'w' is a binary string, then it represents some TM, M_i.

If $L(M_i) = \phi$, that is, Mi does not accept any input, then 'w' is in L_e. Thus, L_e is the language consisting of all those encoded TM's whose language is empty. On the other hand, if $L(M_i)$ is not the empty language, then 'w' is in L_{ne}. Thus, L_{ne} is the language of all codes for Turing Machines that accept atleast one input string. Define the two languages are,

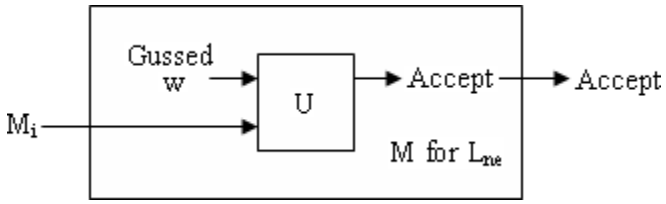
- $L_e = \{M \mid L(M) = \phi\}$
- $L_{ne} = \{M \mid L(M) \neq \phi\}$

Theorem:

L_{ne} is recursively enumerable.

Proof:

In this, a TM that accepts L_{ne}. It is easiest to describe a non deterministic TM M.



The operation of M is as follows;

(1) M takes as input a TM code M_i.
(2) Using its nondeterministic capability, M guesses an input 'w', that Mi might accept.
(3) M test whether M_i accepts 'w'. For this part, M can simulate the Universal TM U that accepts L_u.
(4) If M_i accepts 'w', then M accepts its own input, which is M_i.

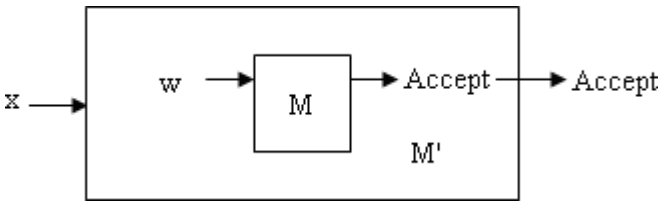
If M_i accepts even one string M will guess that string and accept M_i. However, if $L(M_i) = \phi$, then no guess 'w' leads to acceptance by M_i, so M does not accept M_i. Thus, $L(M) = L_{ne}$.

Theorem:

Lne is not recursive.

Proof:

In this, we must design an algorithm that converts an input that is a binary coded pair(M, w) into a TM M' such that $L(M') \neq \phi$ if and only if M accepts input 'w'. The construction of M' is shown in following fig.



If M does not accept 'w', then M' accepts none of its input's, ie., $L(M') = \phi$. However, if M accepts w, then M' accepts every input, and thus L(M') surely is not ϕ. M' is designed to do the following;

(1) M' ignores its own input 'x'. Rather it replaces its input by the string that represents TM M and input string 'w'. Since M' is designed for a specific pair (M, w) which has some length n, we may construct M' to have a sequence of states $q_0, q_1, \ldots\ldots, q_n$, where q_0 is the start state.
    a. In state q_i, for i=0, 1, ……, n-1, M' writes the (i+1), bit of the code for (M, w) goes to state q_{i+1}, and moves right.
    b. In state q_n, M' moves right, if necessary replacing any nonblanks by blanks.

(2) When M' reaches a blank in state qn, it uses a similar collection of states to reposition its head at the left end of the tape.

(3) Now, using additional states, M' simulates a universal TM U on its present tape.

(4) If U accepts, then M' accepts. If U never accepts, then M' never accepts either.

5.4.2. Rice's Theorem and Properties of the RE Languages:

All nontrivial properties of the RE languages are undecidable, that is, it is impossible to recognize by a TM. The property of the RE languages is simply a set of RE languages. The property of being empty is the set {ϕ) consisting of only the empty language.

A property is trivial if it is either empty, or is all RE languages. Otherwise, it is nontrivial.

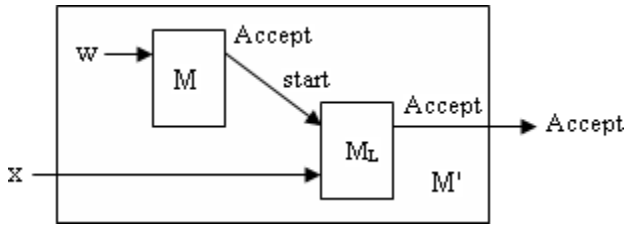
Theorem: (Rice's Theorem)

Every nontrivial property of the RE languages is undecidable.

Proof:

Let P be a nontrivial property of the RE languages. Assume to begin that ϕ, the empty language, is not in P. Since, P is nontrivial, there must be some nonempty language L that is in P. Let M_L be a TM accepting L.

We shall reduce Lu to Lp, thus proving that Lp is undecidable , since Lu is undecidable. The algorithm to perform the reduction takes as input a pair (M, w) and produces a TM M'. The design of M' is given by the following fig.



L(M') is ϕ if M does not accept 'w', and L(M') = L if M accepts 'w'. L(M') = L if M accepts w. The TM M' is constructed to do the following;

(1) Simulate M on input 'w'. Note that 'w' is not the input to M'; rather M' writes M and 'w' onto one of its tapes and simulates the universal TM U on that pair.
(2) If M does not accept 'w', then M' does nothing else. M' never accepts its own input, x, so L(M')=ϕ. Since we assume ϕ is not in property P, that means the code for M' is not in L_P.
(3) If M accepts w, then M' begins simulating M_L on its own input 'x'. Thus M' will accept exactly the language L. Since L is in P, the code for M' is in L_P.

We observe that constructing M' from M and 'w' can be carried out by an algorithm. Since this algorithm turns (M, w) into an M' that is in L_P if and only if (M, w) is in L_u, this algorithm is a reduction of L_u to L_p and proves that the property P is undecidable.

5.5. POST's CORRESPONDENCE PROBLEM

In this section, we will discuss the undecidability of strings and not of Turing Machines. The undecidability of strings is determined with the help of Post's Correspondence Problem(PCP).

Our goal is to prove this problem about strings to be undecidable, and then use its undecidability to prove other problems undecidable by reducing PCP to those.

5.5.1. Definition of Post's Correspondence Problem:

An instance of Post's Correspondence Problem(PCP) consists of two lists of strings over some alphabet Σ; the two lists must be of equal length. We generally refer to the A and B lists, and write A = $w_1, w_2, \ldots\ldots w_k$ and B = $x_1, x_2, \ldots\ldots x_k$, for some integer 'k'. For each 'i', the pair (w_i, x_i) is said to be a corresponding pair.

We say this instance of PCP has a solution, if there is a sequence of one or more integers i_1,i_2…..i_m that, when interpreted as indexes for strings in the A and B lists, yield the same string. That is, $w_{i1}, w_{i2},$……..$w_{im} = x_{i1}, x_{i2},$……x_{im}. We say the sequence i_1,i_2, ….., i_m is a solution to this instance of PCP.

Ex.1:

Consider the correspondence system as given bellows,

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

Does this PCP have a solution?

Soln:

In this case, PCP has a solution. We find, $w_{i1}, w_{i2},$……..$w_{im} = x_{i1}, x_{i2},$……x_{im}

Let m = 4, ie) $i_1 = 2, i_2 = 1, i_3 = 1, i_4 = 3$

(2) (1) (1) (3)

List A: 10111 1 1 10

List B: 10 111 111 0

ie) $w_{i1}, w_{i2},$……..$w_{im} = x_{i1}, x_{i2},$……x_{im}

101111110 = 101111110

∴The solution list is {2, 1, 1, 3}

Ex.2:

Consider the correspondence system as given bellows,

	List A	List B
i	w_i	x_i
1	1	10
2	0	10
3	010	01
4	11	1

Does this PCP have a solution?

Soln:

In this case, PCP has a solution. We find, $w_{i1}, w_{i2},\ldots\ldots w_{im} = x_{i1}, x_{i2},\ldots\ldots x_{im}$

Let m = 6, ie) $i_1 = 1, i_2 = 2, i_3 = 1, i_4 = 3, i_5 = 3, i_6 = 4$

	(1)	(2)	(1)	(3)	(3)	(4)
List A:	1	0	1	010	010	11
List B:	10	10	10	01	01	1

ie) $w_{i1}, w_{i2},\ldots\ldots w_{im} = x_{i1}, x_{i2},\ldots\ldots x_{im}$

10101001011 = 10101001011

$\therefore$ The solution list is {1, 2, 1, 3, 3, 4}

Ex.3:

Consider the correspondence system as given bellows,

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

Does this PCP have a solution?

Soln:

In this case, PCP has no solution. Suppose that the PCP instance has a solution, then we say, $w_{i1}, w_{i2},\ldots\ldots w_{im} = x_{i1}, x_{i2},\ldots\ldots x_{im}$. But in this case, it would not have the equal string [$\therefore w_i \neq x_i$]. So we find the partial solution.

ie) If $i_1 = 1$, $i_2 = 3$, then the two corresponding strings from lists A and B would have to begin.

List A: 10101 …...

List B: 101011…..

ie) $w_{i1}, w_{i2},\ldots\ldots w_{im} = x_{i1}, x_{i2},\ldots\ldots\ldots x_{im}$

10101 $\neq$ 101011

$\therefore$ The two strings can never become equal.

5.5.2. The Modified PCP (MPCP):

- It is easier to reduce Lu to PCP, if we first introduce an intermediate version of PCP, which we call the Modified Post's Correspondence Problem or MPCP.
- In the Modified PCP, there is the additional requirement on a solution that the first pair on the A and B lists must be the first pair in the solution.
- An instance of MPCP is two lists A = $w_1, w_2, \ldots\ldots w_k$ and B = $x_1, x_2, \ldots\ldots. x_k$, and a solution is a list of '0' or more integers $i_1, i_2 \ldots . i_m$ such that, $\mathbf{w_1 w_{i1} w_{i2},\ldots\ldots.. w_{im} = x_1 x_{i1} x_{i2},\ldots\ldots\ldots x_{im}}$.
- Notice that the pair (w_1, x_1) is forced to be at the beginning of the two strings, even though the index '1' is not mentioned at the front of the list that is the solution.
- Unlike PCP, where the solution has to have atleast one integer on the solution list, in MPCP the empty list could be a solution, if $w_1 = x_1$.

Ex:

Consider the correspondence system as given bellows,

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

Does this MPCP have a solution?

Soln:

In this case, MPCP has no solution. In proof, observe that any partial solution has to begin with index 1, so the two strings of a solution would begin;

List A: 1 …...

List B: 111…..

Thus, the next index would have to be '1' yielding;

List A: 11 …...

List B: 111111 …..

ie) $w_{i1}, w_{i2},\ldots\ldots w_{im} = x_{i1}, x_{i2},\ldots\ldots x_{im}$

11 ≠ 111111

∴The solution list is {1, 1}

∴The B string remains three times as long as the A string, and the two strings can never become equal.

5.5.3. Reducing MPCP to PCP:

An instance of MPCP with alphabet Σ, we construct an instance of PCP $C=y_0,y_1,\ldots\ldots y_{k+1}$, and $D=z_0,z_1, \quad z_{k+1}$ as follows;

(1) First, we introduce a new symbol '*' that, in the PCP instance, goes between every symbol in the strings of the MPCP instance. For i=1,2, k, let y_i be w_i with a '*' after each symbol of w_i, and let z_i be xi with a '*' before each symbol of x_i.

(2) $y_0 = *y_1$, and $z_0 = z_1$. That is, the o^{th} pair looks like pair1, expect that there is an extra '*' at the beginning of the string from the first list.

(3) A final pair ($, *$) is added to the PCP instance, $y_{k+1} = \$$ and $z_{k+1} = *\$$

Ex:

An instance of MPCP is defined as,

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

Construct an instance of PCP?

Soln:

	List A	List B
i	y_i	z_i
0	*1*	*1*1*1
1	1*	*1*1*1
2	1*0*1*1*1*	*1*0
3	1*0*	*0
4	$	*$

Theorem:

MPCP reduces to PCP.

Proof:

First, suppose that $i_1,i_2\ldots..i_m$ is a solution to the given MPCP instance with lists A and B. Then we know $w_1w_{i1}w_{i2},\ldots\ldots..w_{im} = x_1x_{i1}x_{i2},\ldots\ldots\ldots x_{im}$. If we were to replace the w's by y's and the x's by z's, we would have two strings that were almost the same, $y_1,y_{i1}, y_{i2},\ldots\ldots y_{im}$, and $z_1,z_{i1}, z_{i2}, \quad z_{im}$.

The difference is that the first string would be missing a '*' at the beginning, and the second would be missing a '*' at the end. That is,

$$*y_1,y_{i1}, y_{i2},\ldots\ldots y_{im} = z_1,z_{i1}, z_{i2}, \quad z_{im}*$$

However, $y_0 = *y_1$ and $z_0 = z_1$, so we can fix the initial '*' by replacing the first index by '0'.

$$*y_0,y_{i1}, y_{i2},\ldots\ldots y_{im} = z_0,z_{i1}, z_{i2}, \ldots\ldots\ldots z_{im}*$$

We can take care of the final '*' by appending the index k+1. Since y_{k+1} = $, and z_{k+1} = *$, we have:

$$y_0,y_{i1}, y_{i2},\ldots\ldots y_{im}, y_{k+1} = z_0,z_{i1}, z_{i2},\ldots\ldots\ldots\ldots z_{im}, z_{k+1}$$

We show that $0,i_1,i_2,\ldots\ldots..i_m,k+1$ is a solution to the instance of PCP. We claim that $i_1,i_2\ldots\ldots..i_m$ is a solution to the MPCP instance. The reason is that if we remove the *'s and the final $ from the strings,

$$y_0y_{i1}\ y_{i2},\ldots\ldots\ y_{im}y_{k+1} = z_0z_{i1}\ z_{i2},\ldots\ldots\ldots\ z_{im}\ z_{k+1}$$

we get $w_1w_{i1}w_{i2},\ldots\ldots..w_{im} = x_1x_{i1}x_{i2},\ldots\ldots\ldots\ldots x_{im}$.

5.5.4. Completion of the Proof of PCP Undecidability:

To complete the chain of reductions by reducing Lu to MPCP. That is, given a pair (M, w), we construct an instance (A, B) of MPCP such that TM M accepts input 'w' if and only if (A, B) has a solution.

The essential idea is that MPCP instance (A, B) simulates, in its partial solutions, the computation of M on input 'w'. Let M = (Q, Σ, Γ, δ, q_0, B, F) be a TM and let w in Σ* be an input string. We construct an instance of MPCP as follows;

(1) The first pair is:

List A	List B
#	#q_0w#

This pair, which must start any solution according to the rules of MPCP, begins the simulation of M on input 'w'.

(2) Tape symbols and the separator # can be appended to both lists. The pairs,

List A	List B	
X	X	
#	#	for each X in Γ.

(3) To simulate a move of M, we have certain pairs that reflect those moves. For all 'q' in Q-F, p in Q, and X, Y and Z in Γ we have;

List A	List B	
qX	Yp	if δ(q, X) = (p, Y, R)
zqX	pZY	if δ(q, X) = (p, Y, L); 'Z' is an tape symbol
q#	yp#	if δ(q, B) = (p, Y, R)
zq#	pZY#	if δ(q, B) = (p, Y, L); 'Z' is an tape symbol

(4) If the ID at the end of the B string has an accepting state, then we need to allow the partial solution to become a complete solution. Thus, if 'q' is an accepting state, then for all tape symbols 'X' and 'Y' there are pairs;

List A	List B
XqX	q
XqY	q
YqX	q
YqY	q
Xq	q
Yq	q
qX	q
qY	q

(5) Finally, once the accepting state has consumed all tape symbols, it stands alone as the last ID on the string. That is the remainder of the two strings is q#. We use the final pair;

List A	List B
q##	#

Ex:

Consider the TM M and w=01, where M=({q_1, q_2, q_3}, {0,1}, {0,1,B}, δ, q_1, B, {q_3}) and δ is given by,

q_i	δ(q_i, 0)	δ(q_i, 1)	δ(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	-	-	-

Reduce the above problem to PCP and find whether that PCP has a solution or not?

Soln:

Rule	**List A**	**List B**	**Source**
(1)	#	#$q_1$01#	
(2)	0 1 #	0 1 #	
(3)	$q_1$0 0$q_1$1 1$q_1$1 0q_1# 1q_1# 0$q_2$0 1$q_2$0 $q_2$1 q_2#	1q_2 $q_2$00 $q_2$10 $q_2$01# $q_2$11# $q_3$00 $q_3$10 0q_1 0q_2#	From δ(q_1, 0) = (q_2, 1, R) From δ(q_1, 1) = (q_2, 0, L) From δ(q_1, 1) = (q_2, 0, L) From δ(q_1, B) = (q_2, 1, L) From δ(q_1, B) = (q_2, 1, L) From δ(q_2, 0) = (q_3, 0, L) From δ(q_2, 0) = (q_3, 0, L) From δ(q_2, 1) = (q_1, 0, R) From δ(q_2, B) = (q_2, 0, R)
(4)	0$q_3$0 0$q_3$1 1$q_3$0 1$q_3$1 0q_3 1q_3 $q_3$0	q_3 q_3 q_3 q_3 q_3 q_3 q_3	

	$q_3 1$	q_3	
(5)	q3##	#	

Now find the sequence of partial solutions, that mimics this computation of M and eventually leads to a solution.

(1) A: #
B: #$q_1$01#

(2) A: #$q_1$01#
B: #$q_1$01#1$q_2$1#

(3) A: #$q_1$01#1$q_2$1#
B: #$q_1$01#1$q_2$1#10q_1#

(4) A: #$q_1$01#1$q_2$1#10q_1#
B: #$q_1$01#1$q_2$1#10q_1#1$q_2$01#

(5) A: #$q_1$01#1$q_2$1#10q_1#1$q_2$01#
B: #$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$101#

(6) A: #$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$101#
B: #$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$101#$q_3$01#

(7) A: #$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$101#$q_3$01#
B: #$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$101#$q_3$01#$q_3$1#

(8) A: #$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$101#$q_3$01#$q_3$1#
B: #$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$101#$q_3$01#$q_3$1#q3#

With only q_3 left in the ID, we can use the pair (q_3##, #) from rule(5) to finish the solution.

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

∴The PCP has a solution. Because it produces same string on list A and list B.

Theorem:

Post's Correspondence Problem is undecidable.

Proof:

The construction of this section shows how to reduce L_u to MPCP. Thus, we complete the proof of undecidability of PCP by proving that the construction is correct, that is;

M accepts ‘w’ if and only if the constructed MPCP instance has a solution. If ‘w’ is in L(M), then we can start with the pair(M, w) from rule1 to rule5, allow the A string to catch up to the B string and form a solution.

In particular, as long as M does not enter an accepting state, the partial solution is not a solution the B string is longer than the A string. Thus, if there is a solution, M must at some point enter an accepting state. That is M accepts w.

5.6. THE CLASSES P AND NP

The classes P and NP of problems solvable in **polynomial time by deterministic and nondeterministic TM’s** and the technique of polynomial time reduction. Also define the notion of “NP-Completeness”.

5.6.1. Problems Solvable in Polynomial Time:

A TM M is said to be of time complexity T(n). If whenever M is given an input ‘w’ of length ‘n’, M halts after making atmost T(n) moves, regardless of whether or not M accepts. Then the language L 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).

5.6.1.1. An Example: Kruskal’s Algorithm:

Kruskal’s algorithm focus on finding a Minimum-Weight Spanning Tree(MWST) for a graph.

Spanning Tree:

A spanning tree is a subset of the edges such that all nodes are connected through these edges, yet there are no cycles.

MWST:

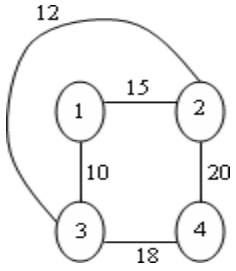
A Minimum-Weight Spanning Tree has the least possible total edge weight of all spanning trees. There is a well-known “Greedy Algorithm”, called Kruskal’s algorithm, for finding a MWST.

Procedure:

(1) Identify the minimal edge connected component.
(2) Find the minimal edge if both the vertices belong to different connected component then add the edge.
(3) Identify minimal edge, if that edge connects both the vertices in the same component then leave the edge. (Otherwise it will form a cycle).
(4) Repeat the process until all the vertices are found.

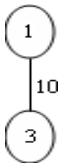
Ex:

Find the MWST using Kruskal's algorithm.

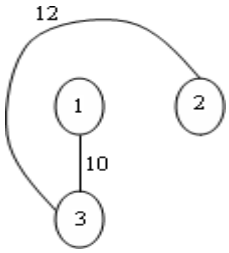


Soln:

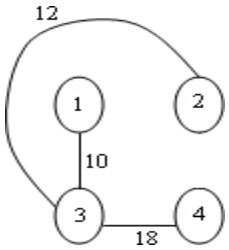
(1) First consider the edge (1, 3) because it has the lowest weight 10.



(2) The next minimal edge is (2, 3), with weight 12. Since 2 and 3 are in different components, we accept this edge and add the node 2 into first connected component. ie;



(3) The third edge is (1, 2) with weight 15. However, 1 and 2 are now in the same component, so we reject this edge and proceed to the fourth edge (3. 4). ie;



(4) Now, we have three edges for the spanning tree of a 4-node graph and so may stop.

When we translate the above ideas to TM's, we face several issues:

- When we deal with TM's, we may only think of problems as languages, and the only output is Yes or No(ie., Accept or Reject). For instance, the MWST problem could be couched as: "Given this graph G and limit weight 'W' or less?
- While we might think informally of the "size" of a graph as the number of its nodes or edges, the input to a TM is a string over a finite alphabet. Thus, problem elements such as nodes and edges must be encoded suitably.

5.6.2. Nondeterministic Polynomial Time:

A fundamental class of problems can be solved by a nondeterministic TM that runs in polynomial time. A language L is in the class NP (Nondeterministic Polynomial) if there is a nondeterministic TM M and a polynomial time complexity T(n) such that L = L(M), and when M is given an input of length 'n', there are no sequences of more than T(n) moves of M.

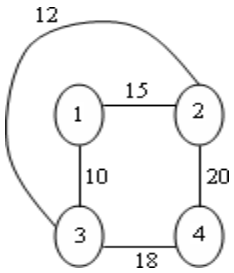
5.6.2.1. An NP Example: The Traveling Salesman Problem:

The input to Traveling Salesman Problem TSP) is the same as to MWST, a graph with integer weights on the edges and a weight limit W.

A **Hamilton Circuit** is a set of edges that connect the nodes into a single cycle, with each node appearing exactly once. Note that the number of edges on a Hamilton Circuit must equal the number of nodes in the graph.

Ex:

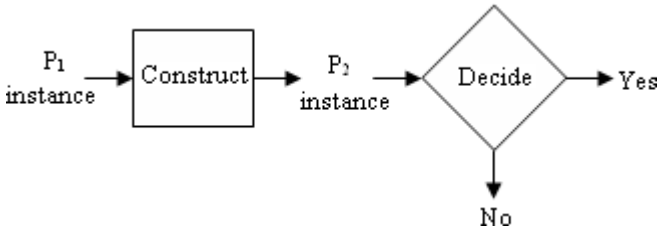
Find Travelling Salesman Problem for the graph,



Soln:

Hamilton Circuit: The cycle (1, 2, 4, 3, 1). The total weight of this cycle is 15+20+18+10 = 63. Thus if 'w' is 63 or more, the answer is "yes", and if w<63 the answer is "no".

5.6.3. Polynomial Time Reductions:



In this, to prove the statement "if P_2 is in P, then so is P_1".

- For the proof, suppose that we can decide membership in P_2 of a string of length 'n' in time $O(n^k)$. Then we can decide membership in P1 of a string of length 'm' in time $O(m^j + (cm^j)^k)$ time; the term m^j accounts for the time to do the translation, and the term $(cm^j)^k$ accounts for the time to decide the resulting instance of P_2.
- Simplifying the expression, we see that P_1 can be solved in time $O(m^j + (cm^j)^k)$. Since c, j and k are all constants, this time is polynomial in 'm', and we conclude P_1 is in P.
- A reduction from P_1 to P_2 is polynomial time if it takes time that is some polynomial in the length of the P_1 instance. Note that as a consequence, the P_2 instance will be of a length that is polynomial in the length of the P_1 instance.

5.6.4. NP – Complete Problems:

Let L be a language(problem) in NP, we say L is NP-Complete if the following statements are true about L;

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

An example of NP- Complete problem is the Travelling Salesman Problem. Since it appears that P≠NP, all the NP-Complete problems are in NP – P, we generally view a proof of NP – Completeness for a problem as a proof that the problem is not in P.

Theorem:

If P_1 is NP-Complete, and there is a Polynomial – time reduction of P_1 to P_2, then P_2 is NP – Complete.

Proof:

To show that every language L in NP Polynomial – time reduces to P_2. We know that there is a polynomial time reduction of L to P_1, this reduction takes some polynomial time P(n). Thus, a string 'w' in L of length 'n' is converted to a string 'x' in P_1 of length atmost P(n).

Also know that there is a polynomial time reduction of P_1 to P_2; Let this reduction take polynomial time q(m). Then this reduction transforms 'x' to some string 'y' in P_2, taking time atmost q(p(n)).

Thus, the transformation of 'w' to 'y' takes time atmost p(n)+q(p(n)), which is a polynomial. We conclude that L is polynomial time reducible to P_2. Since L could be any language in NP, we have shown that all of NP polynomial – time reduces to P_2; that is, P_2 is NP-Complete.