Unit –VI: COMPUTABILITY

Note1:
A Turing machine halts if it enters a state q, scanning a tape symbol X, and there is no move in this situation, i.e., $\delta(q,X)$ is undefined.

Note 2:
We can always assume that a Turing machine halts if it accepts.

Note 3:
Unfortunately, it is not always possible to require that a Turing machine halts even if it does not accept.

Note 4:
Remember that there are three possible outcomes of executing a Turing machine over a given input. The Turing machine may

- Halt and accept the input;
- Halt and reject the input; or
- Never halt.

Recursively Enumeranle Languages and Recursive Languages:

Definition: Recursively Enumerable language

A Language L is said to be recursively enumerable if there exists a Turing Machine M such that L=L(M)

Note:
If L is recursively enumerable language, then L=L(M) for some TM M, and

- If the string w is in L then M halts in a final(or accepting) state
- If the string w is not in L then M may halt in a non-final state or loop forever.(may halt or may not halt)

Definition: Recursive Language:

A language L is said to be recursive language if there exists a Turing Machine M such that L=L(M) and M halts on all inputs.

Note:
If L is recursive language, then L=L(M) for some TM M, and

- If the string w is in L then M halts in a final(or accepting) state
- If the string w is not in L then M may halt in a non-final state or no transition is possible (does not go into infinite loop) (definitely halts).

A Turing Machine of this type corresponds to the notion of "algorithm"

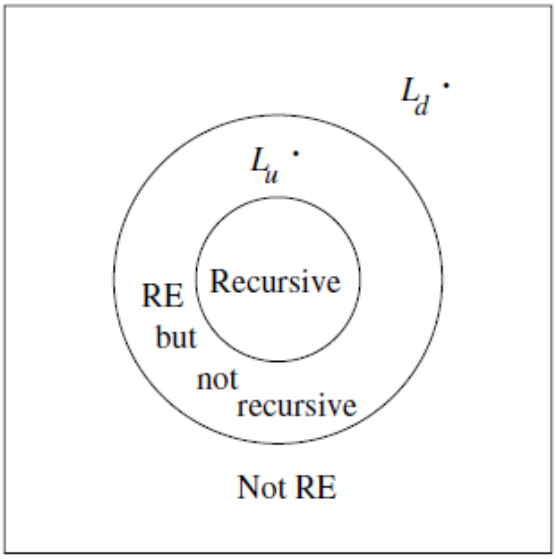
Note 1:

The set of recursive languages is subset of set of recursively enumerable languages

Note 2:

Following are three classes of languages that we can consider now.
a. The recursive languages
b. The languages that are recursively enumerable but not recursive
c. The non –recursively enumerable languages (non RE)

The relation among these languages is given below.

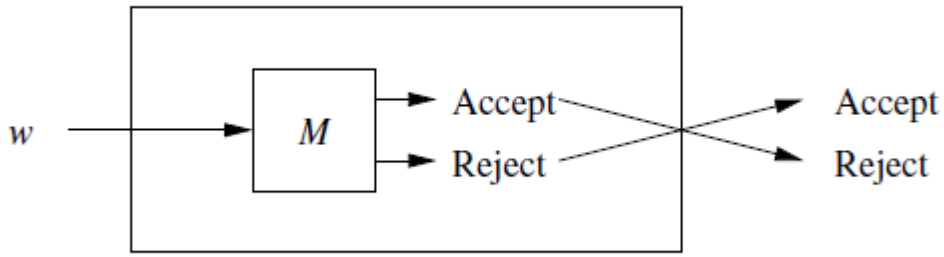


Theorem:

If L is a recursive language, then $\bar{L}$ is also recursive.

Proof:

Let L=L(M) for some TM M that always halts. We construct a TM $\bar{M}$ such that $\bar{L}$ = L($\bar{M}$) that is guaranteed to halt. The construction of $\bar{M}$ is shown in the following figure.



Here M is modified as follows to create $\bar{M}$

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

Since M is guaranteed to halt, we know that $\bar{M}$ is also guaranteed to halt. Also $\bar{M}$accepts exactly those strings that M does not accept.Thus $\bar{M}$ accepts $\bar{L}$

Decidable and undecidable problems:

Note: a problem with only two answers yes/no can be considered as a language.

A problem with two answers (Yes/No) is called decidable if the corresponding language is a recursive language (decides on all inputs) and the problem is called undecidable if it is not a decidable.

Examples of decidable languages and problems:

1. Given a DFA does it accept a given word (Acceptance problem for DFA)
   The corresponding language is
   A_{DFA}={<B,w> / B is a DFA that accepts the string w}

   A_{DFA} is decidable

2. Given two DFA's, do they accept the same language(Equivalence problem for DFA)
The corresponding language is

   E_{DFA}={<A,B>/ A,B are DFA's , L(A) = L(B)}

E_{DFA} is decidable

3. A_{CFG}={<G,w> / G is a CFG that accepts the string w} is decidable

Examples of undecidable languages:

1. A_{TM} = {<M,w> / M is a Turing machine and M accepts w} is undecidable

(i.e., Given a TM, does it accept a input string is undecidable)

Proof:

Suppose that A_{TM} is decidable and is decided by a TM H that halts on all inputs.

Then

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

Using H, now construct another TM D which on input < M >, runs H on input < M, < M > > and outputs opposite of H.(if H exists then definitely D exists)

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

Finally, run D with its own description <D> as input. According to the construction of D, we must have

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

This means that D accepts < D > if D does not accept < D >,which is a contradiction. Hence neither D nor H can exist.

Hence A_{TM} is undecidable.

Halting problem of a Turing Machine:

The halting problem of a Turing machine is given as follows.

"Does a given Turing machine M halts on given input string w".

The corresponding language is given as

$HALT_{TM}$ = { <M,w> / M is a Turing Machine and M halts on input w}

Theorem:

$HALT_{TM}$ = { <M,w> / M is a Turing Machine and M halts on input w} is undecidable.

Proof:

Suppose that $HALT_{TM}$ is decidable.

So by definition of decidability, there exists a TM say R such that L(R) = $HALT_{TM}$ and R halts on all its inputs.

Now construct a TM S as follows.

1. For the TM S, <M,w> is the input
2. Run TM R on input <M,w>
3. If R rejects <M,w> (i.e., TM M does not halt on w), reject

4. If R accepts <M,w> (i.e., TM M halts on w), simulate M on w until it halts.

5. If M accepts w, then S accepts <M,w> , otherwise if M does not accept w, then S does not accept <M,w> .

Now since R halts on all its inputs, S also halts on all inputs. Also, by the above construction, we can show that

$L(S) = \{<M,w> / \text{the TM M accepts w}\}$

So the turing machine S is a decider for A_{TM}. So A_{TM} is decidable which is a contradiction to the fact that A_{TM} is undecidable.

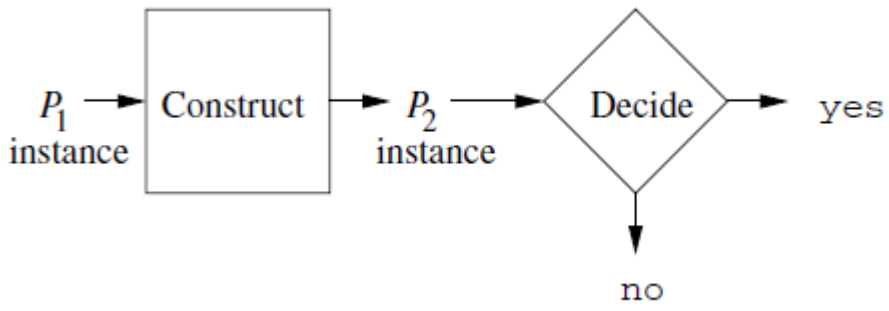
The contradiction arises because of our assumption that $HALT_{TM}$ is decidable.

So $HALT_{TM}$ is undecidable.

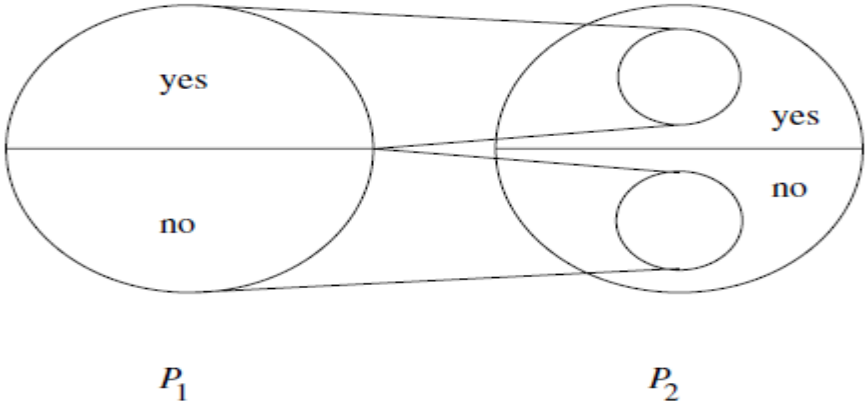
Reductions:

In general, if we have an algorithm to convert instances of a problem P1 to instances of a problem P2 that have the same answer, then we say that P1 reduces to P2.

The reduction strategy is illustrated in the following figure.



A reduction must turn any instance of P1 that has a "yes" answer into an instance of P2 with a "yes" answer, and every instance of P1 with a "no" answer must be turned into an instance of P2 with a "no" answer. This is illustrated in the following figure.



If we could solve problem P_2, then we could use its solution to solve problem P_1. Also if there is a reduction from P_1 to P_2, then following will hold.

a) If P_1 is undecidable, then so is P_2.

b) If P_1 is non-RE, then so is P_2.

Rice's Theorem:

Definition 1:

A property of RE languages is simply a set of RE languages. We say L satisfies the property P if $L \in P$.

Definition 2:

For any property P, define language L_P to consist of Turing Machines which accept a language in P i.e., $L_P = \{<M> \mid L(M) \in P\}$

Also decidability of a property P means that decidability of the language L_P i.e., deciding if a language represented as a TM satisfies the property P.

Ex: 1. {<M> | L(M) is infinite}

2. $L_e = \{<M> \mid L(M) = \Phi\}$

Definition 3: Trivial Properties:

A property is trivial if it is either empty(i.e., satisfied by no language at all) or is all RE languages(i.e., satisfied by all languages) . otherwise it is nontrivial property

Ex:

1. P = { L | L is recognized by a TM with an even number of states}

This is a **trivial property** because for any RE language we have a TM and even if that TM is having odd number of states we can make an equivalent TM having even number of states by adding one extra state. Thus this property satisfied by all the RE languages

2. $P=\{L \mid L \subseteq \Sigma^*\}$

This is a **trivial property** since all languages are subset of Σ* and hence this property satisfied by all recursively enumerable languages.

3. P = { L | L is a recognized by a TM and is finite}

This is a **non trivial property** since it cannot satisified by some Languages and satisfied by some languages.

RICE's theorem:

Statement:

Every nontrivial property of the RE languages is undecidable.

POST CORRESPONDENCE PROBLEM (PCP PROBLEM):

The PCP problem is given as follows.

Given two lists of the strings over some alphabet Σ and of equal length, whether we can pick a sequence of corresponding strings from the two lists and form the same string by concatenation.

An instance of PCP problem is, Given the two lists of strings $A=w_1,w_2,\ldots\ldots\ldots,w_k$ and $B=x_1,x_2,\ldots\ldots\ldots,x_k$ for some integer k. And for each i, the pair (w_i, x_i) is said to be the corresponding pair.

This instance has a solution, if there is a sequence of one or more integers $i_1,i_2,i_3,\ldots\ldots\ldots,i_m$ such that

$$w_{i_1} w_{i_2} \cdots w_{i_m} = x_{i_1} x_{i_2} \cdots x_{i_m}.$$

The sequence $i_1,i_2,i_3,\ldots\ldots\ldots,i_m$ is a solution to this instance of PCP.

Ex:
Let Σ = {0,1} and let A and B are given as follows.

	List A	List B
i	w_i	x_i
1	11	111
2	100	001
3	111	11

The above instance of PCP has a solution. The solution is 1, 2, 3 since

$$w_1 w_2 w_3 = x_1 x_2 x_3$$

Ex:
Let Σ = {0,1} and let A and B are given as follows.

	List A	List B
i	w_i	x_i
1	110	110110
2	0011	00
3	0110	110

The above instance of PCP has a solution. The solution is 2,3,1 since

$$w_2 w_3 w_1 = x_2 x_3 x_1$$

Ex:
Let Σ = {0,1} and let A and B are given as follows.

	List A	List B
i	w_i	x_i
1	01	0101
2	1	10
3	1	11

Here $|w_i| < |x_i|$ for each i=1,2,3 i.e., the list B is having strings of greater lengths than that of list A. so to get the same string after concatenation is difficult. So this instance of PCP has no solution.

Modified PCP (MPCP) problem:

An instance of MPCP is given as follows. Given the two lists of strings $A=w_1,w_2,\ldots\ldots,w_k$ and $B=x_1,x_2,\ldots\ldots,x_k$ for some integer k. And for each i, the pair (w_i, x_i) is said to be the corresponding pair.

This instance has a solution, if there is a sequence of zero or more integers $i_1,i_2,i_3,\ldots\ldots,i_m$ such that

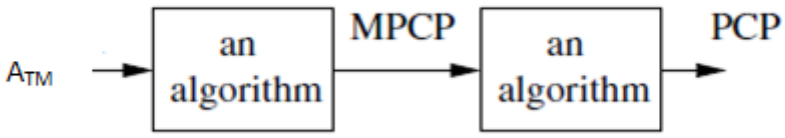
$$w_1 w_{i_1} w_{i_2} \cdots w_{i_m} = x_1 x_{i_1} x_{i_2} \cdots x_{i_m}$$

So in MPCP, the first pair on the two lists A and B must be the first pair in the solution.

Note:

Post's Correspondence Problem is undecidable.

First we can reduce Modified PCP to the original PCP. Then we can reduce A_{TM} to the modified PCP. The chain of reductions is shown in the following figure.



Since A_{TM} is undecidable, we conclude that PCP is undecidable.

Note:

PCP is an important example of an undecidable problem. PCP is a good choice for reducing to other problems and thereby proving them undecidable.

Classes of P and NP:

P consists of all those languages or problems accepted by some deterministic Turing Machine that runs in some polynomial amount of time, as a function of its input length.

So a 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)

Ex:

1. Finding the minimum weight spanning tree of a graph.

2. Sorting an array

NP is the class of languages or problems that are accepted by nondeterministic Turing Machine with a polynomial bound on the time taken along any sequence of nondeterministic choices.

So a language L is in class NP, if there exists 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.

Ex:

1. Travelling Salesman Problem

NP- hard and NP-completeness:

Def: Polynomial Time reduction:

If we can transform instances of one problem P1 in polynomial time into instances of a another problem P2 that has the same answer yes or no, then we say that Problem P1 is polynomial – time reducible to problem P2.

NP-Hard:

A language L is said to be NP-hard if for every language L' in NP there is a polynomial time reduction of L' to L. Here we cannot able to prove that L is in NP.

NP-complete:

A language L is said to be 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

Ex: 1.The SAT problem (or) Satisfiability problem- "Given a Boolean expression, is it satisfiable? " is NP-complete.

2. Hamilton Circuit problem

3. Travelling Salesman problem

Boolean satisfiability Problem(SAT Problem):

A *literal* is a either a propositional variable or the negation of a propositional variable.

Ex: Variables, such as x and y, are called *positive literals* and Negated variables, such as

$\neg x$ and $\neg y$, are called *negative literals*.

A *clause* is a disjunction (*or*) of one or more literals.

Ex:

$$(x \vee \neg y \vee z)$$

is a clause.

A *clausal formula*, which we will just call a formula, is a conjunction (*and*) of one or more clauses.

For example,

$$(\neg x \vee y) \wedge$$
$$(\neg y \vee z) \wedge$$
$$(x \vee \neg z \vee y)$$

is a (clausal) formula.

Definition:

A boolean clausal formula φ is satisfiable if there exists an assignment of values to its variables that makes φ true.

For example, formula

$$(\neg x \vee y) \wedge$$
$$(\neg y \vee z) \wedge$$
$$(x \vee \neg z \vee y)$$

is satisfiable since Choosing

$$x = \text{false}$$
$$y = \text{true}$$
$$z = \text{true}$$

The above formula becomes true.

The satisfiability problem, usually called SAT, is the following language.

$$\text{SAT} = \{\varphi \mid \varphi \text{ is a Boolean clausal formula that is satisfiable}\}.$$