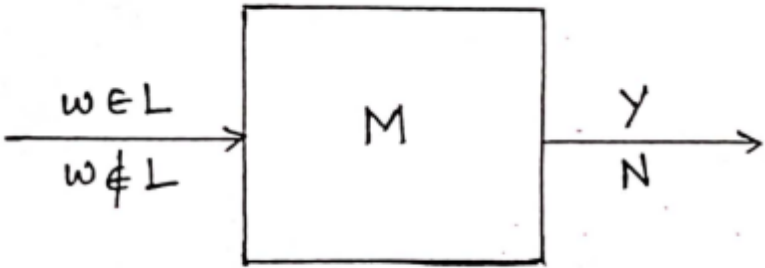
Recursive Language.

Definition: A language $L \subseteq \Sigma^*$ representing a problem over Σ, is said to be Recursive Language, if there exist a Turing Machine M which always halts when given any input $w \in \Sigma^*$, whether $w \in L$ or $w \notin L$. If $w \in L$ then M halts will output Y indicating that the string w is in L, and if $w \notin L$ then M halts will output N indicating that w does not belong to L.

Recursive Language is also known as Turing Decidable Language. The following figure shows the concept behind Recursive Language.

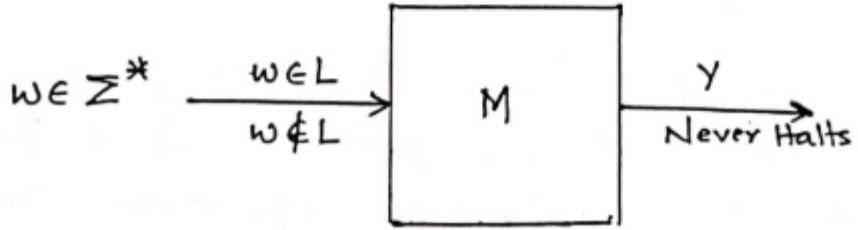


where $L \subseteq \Sigma^*$ and $w \in \Sigma^*$

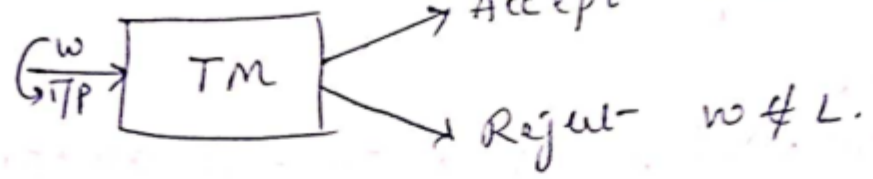
Recursively Enumerable Language.

Definition: A language $L \subseteq \Sigma^*$ is said to be Recursively Enumerable Language, if there exist a Turing Machine M that halts by producing o/p Y for input strings $w \in L$ and never halts for input strings $w \notin L$. ie If $w \notin L$ then M continues in an infinite loop.

Recursive Enumerable Language is also known as Turing Acceptable Language.

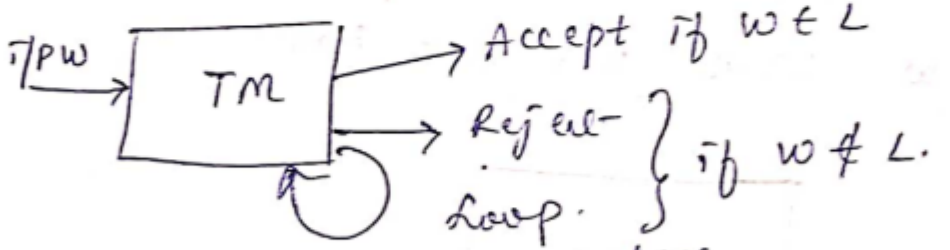


Recursive Language (REC)



TM Always halts.

Recursively Enumerable Language: (RE)



only acceptance criteria is defined property alone. remaining features are not defined property.

we cannot say if TM will halt or not.

Language accepted by TM are known as RE langs.

Theorem 8.4

If L_1 and L_2 are both recursively enumerable languages over Σ, then $L_1 \cup L_2$ and $L_1 \cap L_2$ are also recursively enumerable.

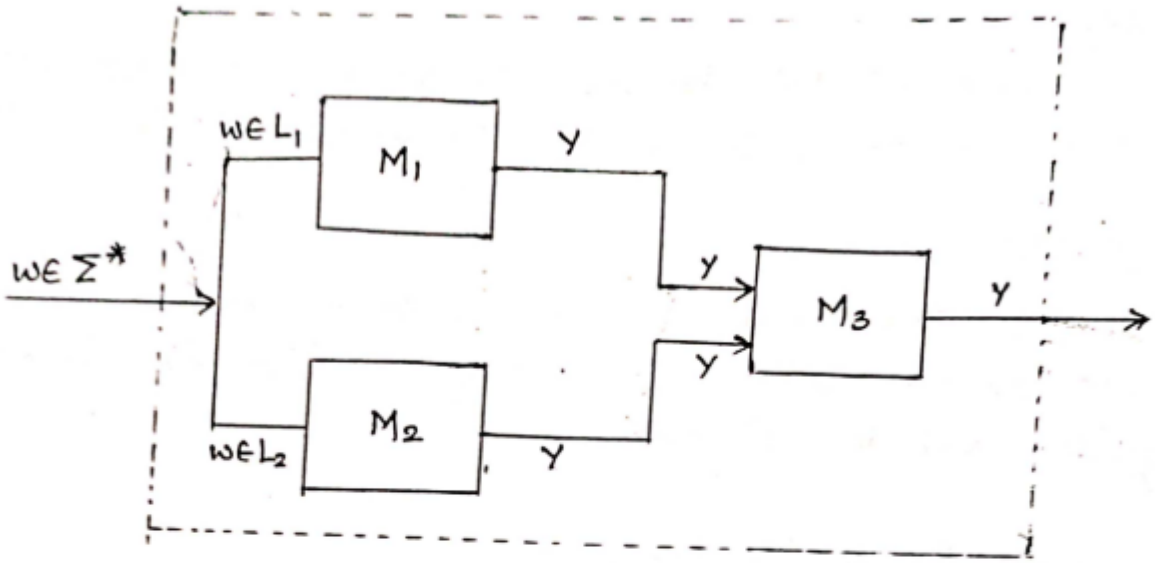
Proof

Suppose that T_1 is a TM that accepts L_1, and T_2 is a TM that accepts L_2. The algorithms we describe to accept $L_1 \cup L_2$ and $L_1 \cap L_2$ will both process an input string x by running T_1 and T_2 simultaneously on the string x. We might implement the algorithm by building a two-tape TM that literally simulates both TMs simultaneously, one on each tape, or we might simply use transition diagrams for both TMs to trace one move of each on the input string x, then two moves of each, and so forth.

To accept $L_1 \cup L_2$, the algorithm is simply to wait until one of the two TMs accepts x, and to accept only when that happens. If one rejects, we abandon it and continue with the other. If both eventually reject, we can reject and halt, though this is not required in order to accept $L_1 \cup L_2$. If both TMs loop forever, then we never receive an answer, but the only case in which this can occur is when $x \notin L_1 \cup L_2$.

To accept $L_1 \cap L_2$, the algorithm is to wait until both TMs accept x, and to accept only when that happens. This time, if one TM accepts, we continue with the other; if either TM ever rejects, we can reject and halt, though again this is not necessary.

Theorem 1: If L_1 and L_2 are Recursively Enumerable Languages then $L_1 \cup L_2$ is also Recursively Enumerable.



Let M_1 be a TM for accepting L_1 and M_2 be a TM for accepting L_2. Now construct a TM M_3 having $\{Y\}$ as the set of i/p symbol. M_1, M_2 and M_3 are arranged as shown in figure.

If at any stage, there is an o/p from any one of M_1 or M_2, then on the first o/p from either M_1 or M_2, the machine M_3 is activated and the o/p from M_1 or M_2, whichever is available is written on the tape of M_3. If the i/p tape of M_3 contains a symbol Y then the machine M_3 halts by producing o/p Y. Else it continues in an infinite loop, indicating that $L_1 \cup L_2$ is recursively enumerable.

Hence $L_1 \cup L_2$ is recursively enumerable since there exist a machine that halts by producing o/p Y for i/ps $w \in L_1$ or $w \in L_2$ or $w \in$ both L_1 & L_2. Otherwise the machine continues in an infinite loop.

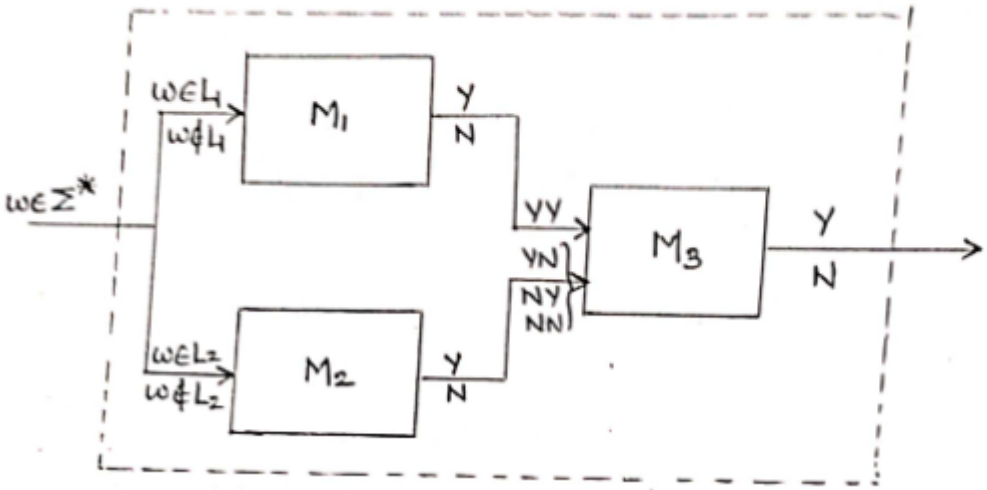
Theorem 8.5

If L_1 and L_2 are both recursive languages over Σ, then $L_1 \cup L_2$ and $L_1 \cap L_2$ are also recursive.

Proof

See Exercise 8.1.

Theorem 2: If L_1 and L_2 are recursive then $L_1 \cap L_2$ is also recursive.



Let M_1 be a TM for deciding L_1 and M_2 be a TM for deciding L_2. Now construct a TM M_3 having $\{Y, N\}$ as the set of symbols. M_1, M_2 and M_3 are arranged as shown in figure. Input to M_3 will be either YY, YN, NY or NN. The machine M_3 halts by producing o/p Y if the i/p contains only Y, otherwise M_3 halts by producing o/p N. ie M_3 returns Y as output if both M_1 and M_2 produces o/p Y.

ie If and only if w belongs to both L_1 and L_2 then M_3 halts by producing o/p Y, else M_3 halts by producing o/p N.

Hence $L_1 \cap L_2$ is also recursive since there exist a machine M_3 that halts by producing o/p Y for i/p strings $w \in L_1 \cap L_2$ and halts by producing o/p N for i/p strings $w \notin L_1 \cap L_2$.

Theorem 8.6

If L is a recursive language over Σ, then its complement L' is also recursive.

Proof

If T is a TM that computes the characteristic function χ_L, the TM obtained from T by interchanging the two outputs computes the characteristic function of L'.

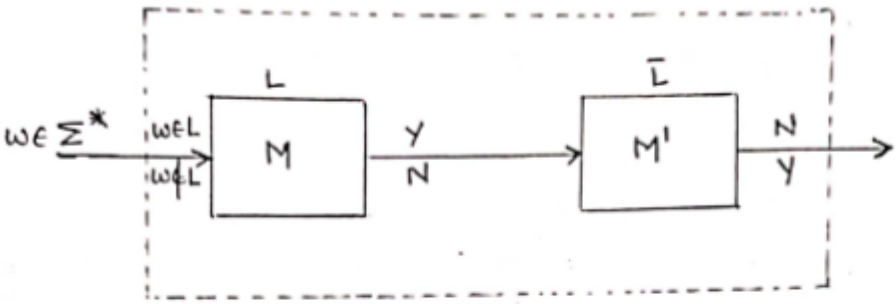
Theorem 8.7

If L is a recursively enumerable language, and its complement L' is also recursively enumerable, then L is recursive (and therefore, by Theorem 8.6, L' is recursive).

Proof

If T is a TM accepting L, and T_1 is another TM accepting L', then here is an algorithm to decide L: For a string x, execute T and T_1 simultaneously on input x, until one halts. (One will eventually accept, because either $x \in L$ or $x \in L'$.) If the one that halts first is T, and it accepts, or the one that halts first is T_1, and it rejects, return 1; otherwise return 0.

Theorem 3: If L is recursive language, then $\bar{L}$ is also recursive



Let M be a TM for deciding L. Now construct a machine M' having {Y, N} as the set of symbols. M and M' are arranged as shown in figure. Input to M' will be either Y or N. The machine M' halts by producing o/p N if the i/p is Y and M' halts by producing o/p Y if the i/p is N.

if $w \in \Sigma^*$ is applied as i/p to the above arrangement

If $w \in L$ then machine M halts by producing o/p Y and M' halts by producing o/p N.

If $w \notin L$ then machine M halts by producing o/p N and M' halts by producing o/p Y.

Hence $\bar{L}$ ie $\Sigma^* - L$ is also recursive since there exist a machine that halts by producing o/p Y for strings $w \in \bar{L}$ and halts by producing o/p N for strings $w \notin \bar{L}$.

Reference: Introduction to Automata Theory, Languages & Computation by John. E. Hopcroft, Jeffery. D. Ullman (page No: 178 – 181)

Definition 8.10 Unrestricted Grammars

An *unrestricted* grammar is a 4-tuple $G = (V, \Sigma, S, P)$, where V and Σ are disjoint sets of variables and terminals, respectively. S is an element of V called the start symbol, and P is a set of productions of the form

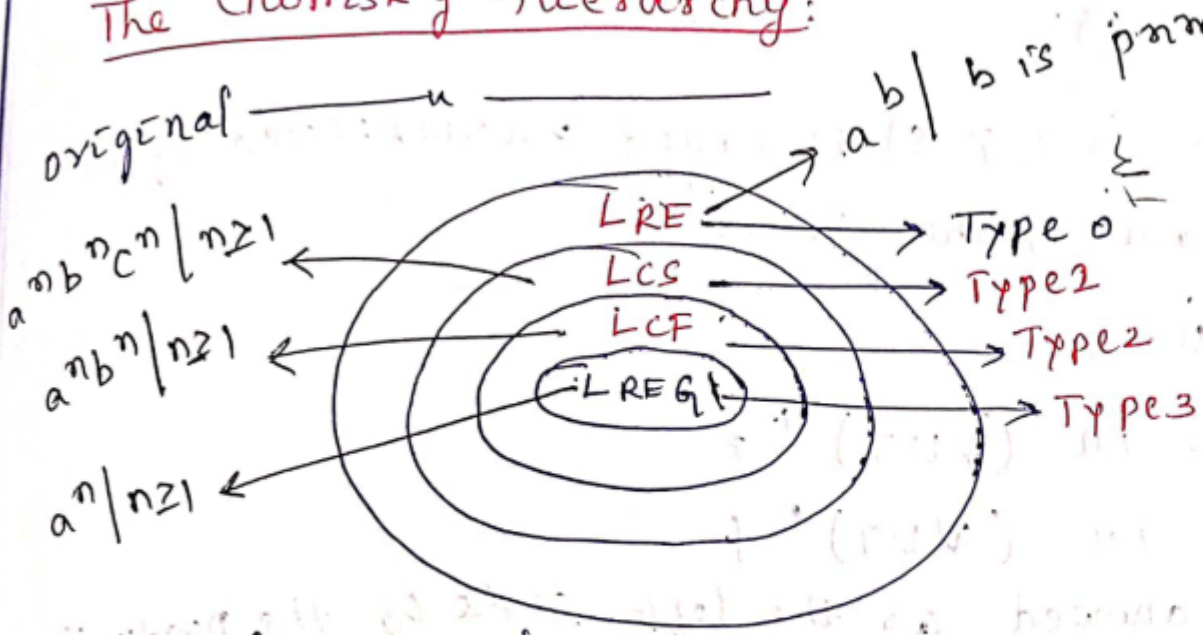
$$\alpha \rightarrow \beta$$

where $\alpha, \beta \in (V \cup \Sigma)^*$ and α contains at least one variable.

Table 8.21 | The Chomsky Hierarchy

Type	Languages (Grammars)	Form of Productions in Grammar	Accepting Device
3	Regular	$A \to aB$, $A \to \Lambda$ $(A, B \in V, a \in \Sigma)$	Finite automaton
2	Context-free	$A \to \alpha$ $(A \in V, \alpha \in (V \cup \Sigma)^*)$	Pushdown automaton
1	Context-sensitive	$\alpha \to \beta$ $(\alpha, \beta \in (V \cup \Sigma)^*, \lvert\beta\rvert \geq \lvert\alpha\rvert,$ α contains a variable)	Linear-bounded automaton
0	Recursively enumerable (unrestricted)	$\alpha \to \beta$ $(\alpha, \beta \in (V \cup \Sigma)^*,$ α contains a variable)	Turing machine

The Chomsky Hierarchy:



Noam chomsky, a founder of formal language theory.

RG RL - FA
CFG CFL - PDA
CSG CSL - LBA
RE sets - TM

Extended Hierarchy:

LRE > LREG > LCS > LCF > LDCF > LREG

Here LDCF → Deterministic context-free languages.

LREC → Recursive languages.

Definition 9.14 Post's Correspondence Problem

An instance of Post's correspondence problem (PCP) is a set $\{(\alpha_1, \beta_1), (\alpha_2, \beta_2), \ldots, (\alpha_n, \beta_n)\}$ of pairs, where $n \geq 1$ and the α_i's and β_i's are all nonnull strings over an alphabet Σ. The decision problem is this: Given an instance of this type, do there exist a positive integer k and a sequence of integers $i_1, i_2, \ldots, i_k$, with each i_j satisfying $1 \leq i_j \leq n$, satisfying

$$\alpha_{i_1}\alpha_{i_2}\ldots\alpha_{i_k} = \beta_{i_1}\beta_{i_2}\ldots\beta_{i_k}$$

An instance of the *modified* Post correspondence problem (*MPCP*) looks exactly like an instance of *PCP*, but now the sequence of integers is required to start with 1. The question can be formulated this way: Do there exist a positive integer k and a sequence $i_2, i_3, \ldots, i_k$ such that

$$\alpha_1 \alpha_{i_2} \ldots \alpha_{i_k} = \beta_1 \beta_{i_2} \ldots \beta_{i_k}$$

Instances of *PCP* and *MPCP* are called *correspondence systems* and *modified correspondence systems*, respectively. For an instance of either type, if it is a yes-instance we will say that there is a *match* for the instance, or that the sequence of subscripts is a match, or that the string formed by the α_{i_j}'s represents a match.

8.5. Post's Correspondence Problem

The Post correspondence problem can be stated as follows. Given two sequences of n strings on some alphabet Σ say

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

and

$$B = v_1, v_2, \ldots, v_n$$

we say that there exists a Post correspondence solution for pair (A,B) if there is a nonempty sequence of integers i,j, …, k, such that

$$w_i w_j \ldots w_k = v_i v_j, \ldots, v_k.$$

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

8.4. Halting Problem

The "***Halting Problem***" can informally be stated as "Given a Turing machine M and an input string w with the initial configuration q_0, after some (or all) computations do the machine M halts?" In other words we have to identify whether (M, w) where M is the Turing machine, halts or does not halt when w is applied as the input. The domain of this problem is to be taken as the set of all Turing machines and all w i.e., Given the description of an arbitrary Turing machine M and the input string w, we are looking for a single Turing machine that will predict whether or not the computation of M applied to w will halt.

When we state decidability or undecidability results, we must always know what the domain is, because this may affect the conclusion. The problems may be decidable on some domain but not on another.

It is not possible to find the answer for Halting problem by simulating the action of M on w by a universal Turing machine, because there is no limit on the length of the computation. If M enters into an infinite loop, then no matter how long we wait, we can never be sure that M is in fact in a loop. The machine may be in a loop because of a very long computation. What is required is an algorithm that can determine the correct answer for any M and w by performing some analysis on the machine's description and the input.

Formally, the Halting Problem is stated as "Given an arbitrary Turing machine $M = (Q, \Sigma, \Gamma, \delta, q_0, B, A)$ and the input $w \in \Sigma^*$, does M halt on input w?".