Properties of Reccursive and Recursively Enumerable Languages

Defn of Recursive Enumerable language. (R.E)

A language is RE if some TM accepts it.

Let L be a R.E Language and M be the TM that accepts it. For string w,

if $w \in L$, then M halts on a F.S (Final state)

if $w \notin L$, then M halts in a ~~FS~~ non FS or loops forever.

Recursive language

A language is recursive if there is a membership algm for it. OR. A language is recursive, if some TM accepts it and halt on any input string.

Let L be a recursive language and M be the TM that accepts it. For string w

if $w \in L$, then M halts in a FS

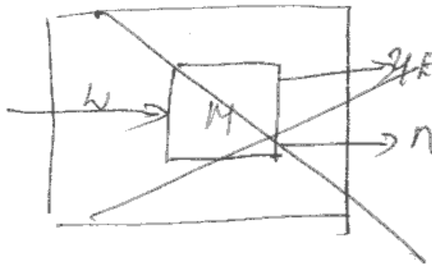
if $w \notin L$, then M halts on a non FS.

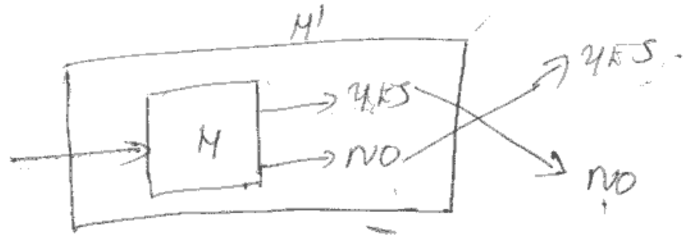
Closure properties of the classes of recursive and recursive enumerable (r.e) sets.

1) The complement of a recursive language is recursive ← Theorm statment.

Proof

Let L be Recursive language and M a TM that halts on all i/p and accepts L. Construct M' from M so that if M enters a FS on i/p w, then M' halts with out accepting. If M halts with out accepting, M' enters a FS. So $L(M')$ is the complement of L and the complement of L is a recursive language.





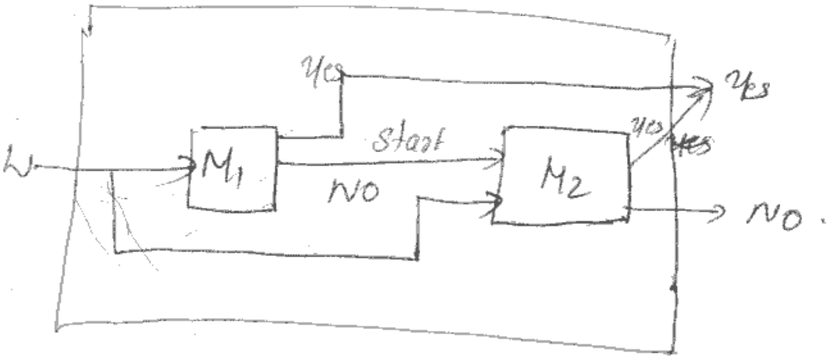
The above construction shows that recursive languages are closed under complementation.

Theorm 2

The union of two recursive language is recursive. The union of two RE language is recursively enumerable.

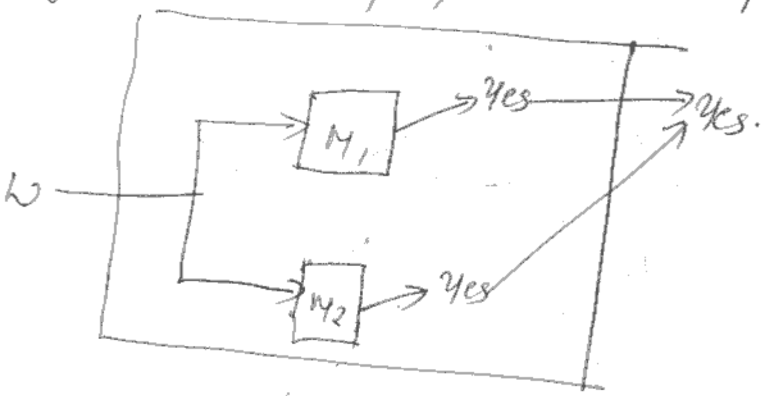
Proof

Let L_1 and L_2 be recursive languages accepted by algorithms / ~~TMs~~ M_1 and M_2. We construct M which first simulates M_1. If M_1 accepts, then M accepts. If M_1 rejects, then M simulates M_2 and accepts iff M_2 accepts. Since both M_1 and M_2 are algms, M is guaranteed to halt. Clearly M accepts $L_1 \cup L_2$.



$M \subseteq L(M) = L(M_1) \cup L(M_2)$

For R.E languages the above construction does not work since M_1 may not halt. Instead M can simultaneously simulate M_1 and M_2 on separate tapes. If either accepts, the M accepts.

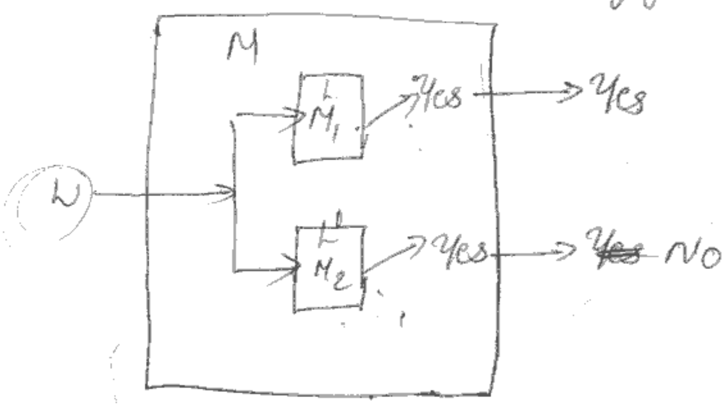


Theorm 3

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

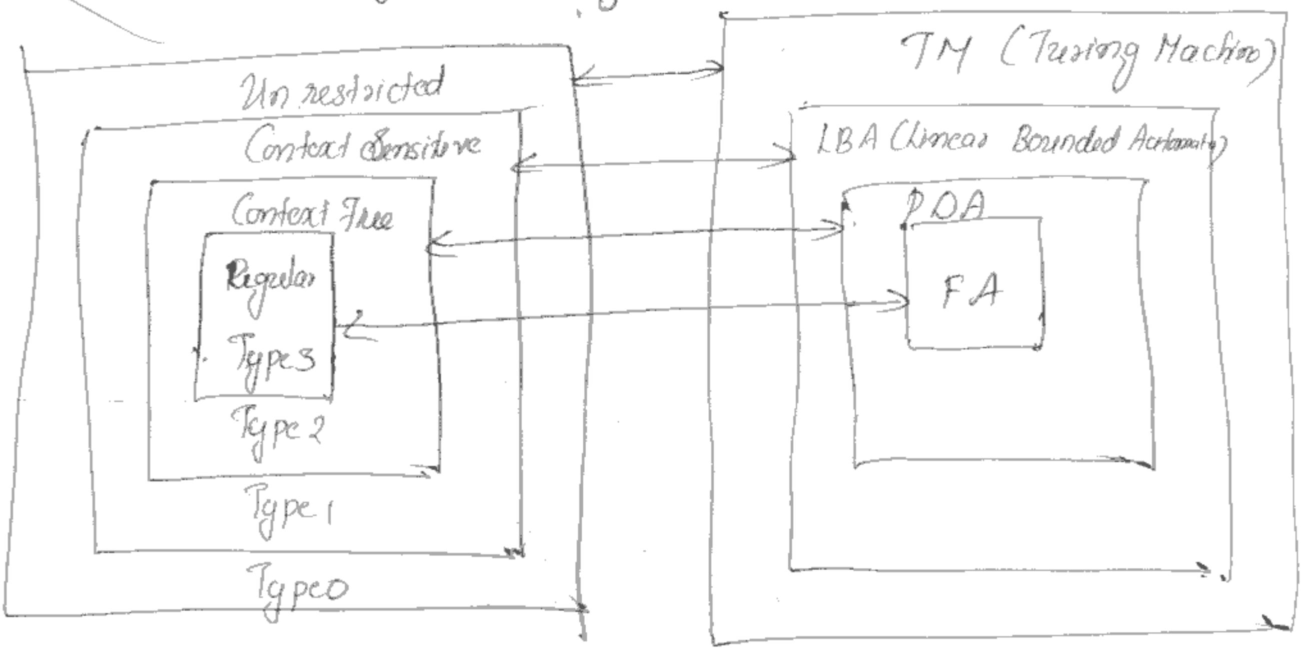
Proof.

Let M_1 and M_2 accept L and $\bar{L}$ respectively. Construct M as in the fig below



to simulate simultaneously M_1 and M_2. M accepts w if M_1 accepts w and rejects w if M_2 accepts w. Since w is in either L or $\bar{L}$, (either of M_1 or M_2 will accept). Thus M will always say either "yes" or "no", but will never say both. Since M is an algm that accepts L, it follows that L is recursive.

Chomsky Hierarchy



The 4 classes of languages are often called the Chomsky hierarchy. This was introduced by Noam Chomsky. He classified the grammars in to 4 classes viz, Type 0, Type 1, Type 2 and Type 3 which are defined as follows

Type 0 or Unrestricted Grammar

A grammar, $G = (V, T, S, P)$ is called Type 0, if all the productions are the form

$u \to v$, where $u \in (V \cup T)^{+}$ and $v \in (V \cup T)^{*}$

The language generated by the unrestricted grammar is called recursively enumerable language.

Type 1 or Context Sensitive Grammar

A grammar is said to be Type1, if all the productions are of the form

$x \to y$ where $x, y \in (V \cup T)^{+}$ and $|x| \leq |y|$

Type 2 or Context Free Grammar

A grammar is said to be Type 2, if its production are given as

$A \to \alpha$, where $A \in V$, $\alpha \in (V \cup T)^{*}$

Type 3 or Regular Grammar

A grammar is said to be regular grammar, if its production are given as

$A \to xB$, $A \in V$, $B \in (V \cup \epsilon)$, $x \in \Sigma^{*}$

eg: $S \to 0A \mid \epsilon$

$A \to S10 \mid 0$

* If the pdns are of the form $\left.\begin{array}{l} A \to wB \\ A \to w, \end{array}\right\}$ or

Where A, B are variables, w is a (possibly empty) string of terminals, then we say the grammar is rt linear. If all productions are of the form $A \to Bw$ or $A \to w$, we call it left linear. A rt or lt linear grammar is called regular grammar.

Total recursive function.

This ~~is~~ function corresponds to the recursive languages, since they are computed by TM's that always halt. All common arithmetic function on integers, such as multiplication, $n!$, $\lceil \log_2 n \rceil$ and 2^{2^n} are total recursive functions.

Partial Recursive Function

These are analogous to the Recursively Enumerable Languages, since they are computed by Turing machines that may or may not halt on a given i/p.

Techniques for Turing Machine Construction

(1) Storage in the finite control

The finite control can be used to hold a finite amount of info. To do so, the state is written as a pair of elements, one excercising control (state) and other storing a symbol. No modification in the defn of TM has been made.

(for egs ~~see~~ example refer text)

(2) Multiple Tracks

We can imagine that the tape of the TM is divided in to k tracks, for any finite k. This arrangement is as follows, with k = 3. The symbols on the tape are considered k-tuples, one component for each track.

¢	1	0	1	1	1	\$	B	
B	B	B	B	1	0	1	B	....
B	1	0	0	1	0	1	B	

[A 3-track Turing Machine]

(3) Checking off symbols

It is a useful method for visualizing how a

TM recognizes languages defined by repeated strings, such as
$\{ww \mid w \text{ in } \Sigma^*\}$, $\{wcy \mid w \text{ and } y \text{ in } \Sigma^*, w \neq y\}$ or $\{ww^R \mid w \text{ in } \Sigma^*\}$.

It is also useful when lengths of substrings must be compared, such as in the languages:

$$\{a^i b^i \mid i \geq 1\} \text{ or } \{a^i b^j c^k \mid i \neq j \text{ or } j \neq k\}$$

We introduce an extra track on the tape that holds a blank or ✓. The ✓ appears when the symbol below it has been considered by the TM in one ~~an~~ of its comparisons.

⑤ <u>Shifting over</u>

A TM can make space on its tape by shifting all non blank symbols a finite no. of cells to the right. To do so, the tape head makes an excursion to the rt, repeatedly storing the symbols read in its finite control and replacing them with symbols read from cells to the left. The TM can then return to the vacated cells and print symbols of its choosing. If ~~spacim~~ space is available, it can push blocks of symbols left in a similar manner

(5) Subroutines

A TM can simulate any type of subroutine found in programming languages. The general idea is to write part of TM program to serve as a subroutine; it will have a designated initial state and return state which temporarily has no move and which will be used to effect a return to the calling routine. To design a TM that calls a subroutine, a new set of states for subroutine is made, and move from the return state is specified. The call is effected by entering the initial state and return is effected by move from the return state.

eg: refer text book.

Decidability / Non Decidability

Decidable problems are those problems with answer Yes or No.

eg: Does Machine M have 3 states?
Is string w a binary no.?
Does DFA M accepts any i/p?,

A problem is decidable if some TM decides (solves) the problem. TM answers YES or NO for each instance of the problem.

The machine that decides (solves) a problem

> If the answer is YES then halts in a YES state

> If the answer is NO then halts in a no state

These states may not be final states.

Undecidable

Some problems are undecidable which means, there is no TM that solves all instances of the problem.

A simple undecidable problem

eg: A membership problem

The Membership Problem

I/p :- * TM, M

* String, w

Problem : Does M accepts w?

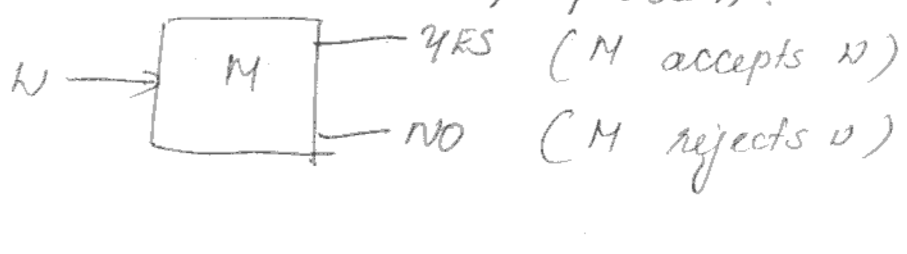
$w \in L(M)$?

Theorm

The membership problem is undecidable (there ar M and w for which we can't decide whether $w \in L(M)$).

Proof

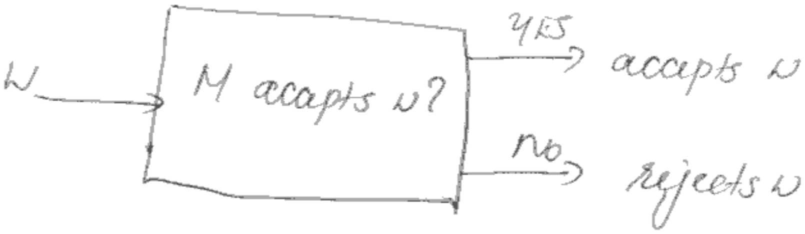
Assume for contradiction that the membership problem is decidable. Thus there exists a TM M that solves the membership problem.



Let L be recursively enumerable language &
Let M be the TM that accepts L.

We will prove that L is also recursive:
We will describe a TM that accepts L and halts on any i/p.

TM accepts that accepts L and halts on any i/p.



Therefore L is recursive.

Since L is chosen arbitrarily, every R.E language is also recursive.

But there are R.E. language which are not recursive.

Contradiction.

Therfore the membership pbm is undecidable

(2) Another famous undecidable problem

The halting problem

Input : TM , M

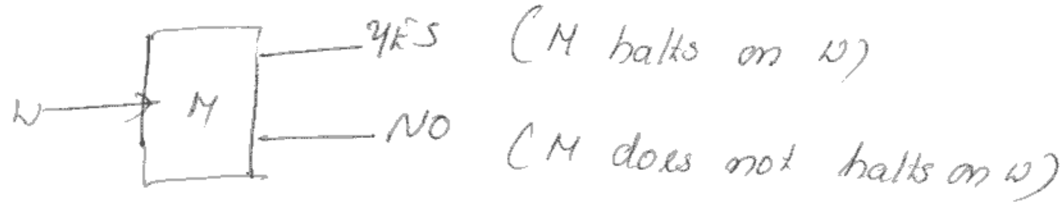
: String w

Quest : Does M halts on i/p w?

Theorem : The halting problem is undecidable.

(there are M and w for which we can't decide whether M halts on i/p w), If halting pbm was decidable then even R.E language is recursive.

Proof : Assume for contradiction that halting pbm is decidable. Thus there exist TM, M that solves the halting problem.

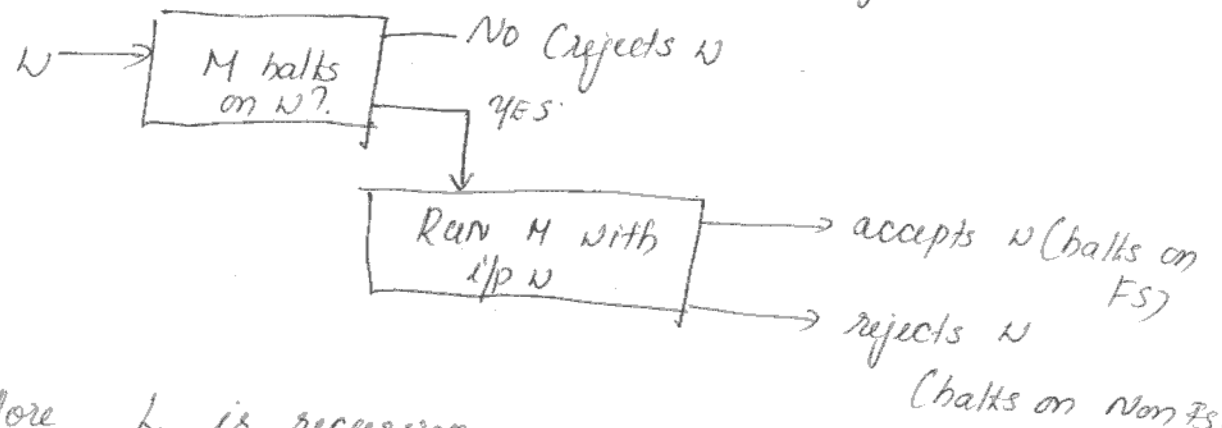


Let L be a R.E language and let M be the TM that accepts L.

We will prove that L is also recursive:

We will describe a TM that accepts L and halts on any i/p.

TM that accepts L and halts on any i/p.



Therefore L is recursive

Since L is chosen arbitrarily, every RE language is also recursive.

But there are RE languages ~~that~~ which are not recursive.

Contradiction!!!

Therefore the halting problem is undecidable.

The Church Hypothesis or The Church Turing Thesis

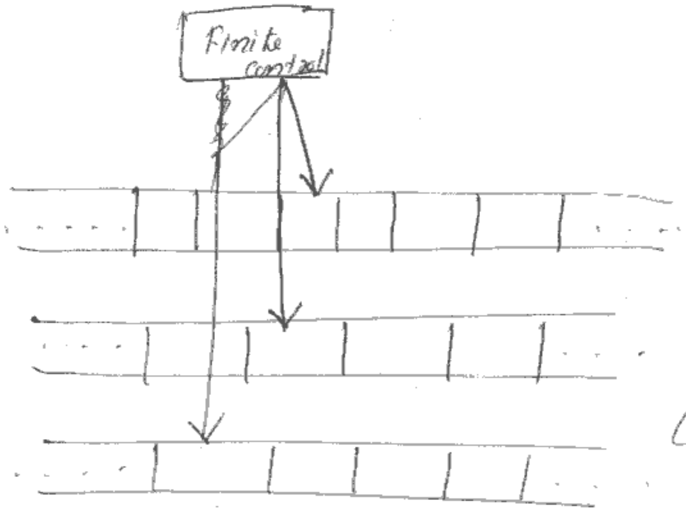
The mathematician and logician, Alonzo Church proposed an alternative formalization for the notion of algm in 1936, known as Church-Turing thesis. This conjecture is stated in number of ways as follows.

1. Any computation that can be carried out by mechanical means can be performed by some TM
2. Anything that is intuitively computable can be computed by a TM
3. The TM that halts on all inputs is the precise formal notion corresponding to the intuitive notion of an algm.
4. Given any problem which can be solved with an effective algorithm, there is a TM that can solve this problem.

The word thesis is used instead of the word "theorem" as it is not a mathematical result. It is based on the intuitive notion of what "mechanical computations" are and equates it with a mathematical idea. ie 'algm'.

Theorm

If a language L is accepted by a multitape TM, it is accepted by a single-tape TM.



(Multitape TM)

Simulation of 3 tapes by one