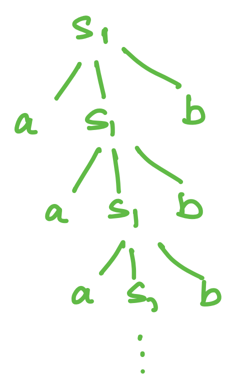
Q: what language does the following unrestricted grammar derive?

$$\left.\begin{array}{l} S \to S_1 B \\ S_1 \to a S_1 b \\ bB \to bbbB \\ aS_1b \to aa \\ B \to \lambda \end{array}\right\} \text{unrestricted grammar}$$



$a^n S_1 b^n$

taking 1 b $\longrightarrow$ generating 2 b's

b's are increasing with a common ratio of 2.

$$S$$

$$S_1 B$$

$$a^n S_1 b^n B$$

$$a^{n-1} \underline{a S_1 b}\, b^{n-1} B$$

$$a^{n-1} aa\, b^{n-1} B$$

$$a^{n-1} a^2 b^{n-1} B$$

$$a^{n+1} b^{n-1} B$$

$$a^{n+1} \cdot b^{n-2} \cdot b \cdot B$$

$B \rightarrow \lambda$

$a^{n+1} \cdot b^{n-2} \cdot b \cdot \lambda$

$\boxed{a^{n+1} \cdot b^{n-1} \cdot \lambda}$

$bB \rightarrow bbbB$

$a^{n+1} b^{n-2} bbbB$

$a^{n+1} b^{n+1} B$

$B \rightarrow \lambda$

$\boxed{a^{n+1} b^{n+1} \lambda}$

$a^{n+1} \cdot b^{n} \cdot \underline{bB}$

$a^{n+1} b^{n} bbbB$

$a^{n+1} b^{n+3} B$

$B \rightarrow \lambda$

$\boxed{a^{n+1} b^{n+3} \lambda}$

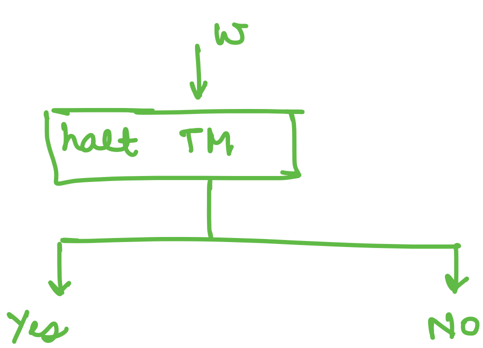
$L = \{ a^{n+1} b^{n+k} \lambda \ ; \ n \geq 1, \ k = -1, 1, 3, 5 \ldots \}$

Membership Algorithm:

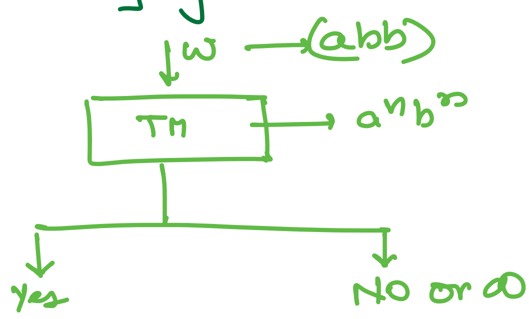
Given a string and a language, tell whether string belongs to the language or not.
Give an answer in Yes / NO.

String: aabb language: $\underline{a^n b^n}$ ↳ TM.

Recursive language



Recursively Enumerable language



($w \in L$)
(halts at a final state)

($w \notin L$)
(halts at a non final State)

($w \in L$)
(halts at a final State)

($w \notin L$)
may halt at a non final State or it can go in an ∞ loop

Membership algo exists

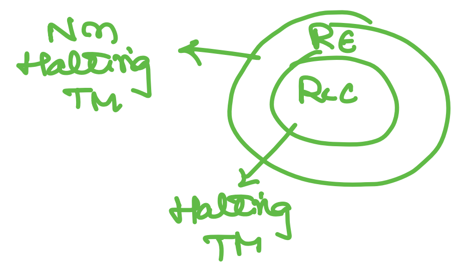
If m/c halts at non final state : NO
If m/c halts at a final state: Yes

membership algo doesnot exist bcz we might go into an ∞ loop, we will keep on waiting we donot get the answer in Yes/NO.

If you are not able to get ans in Yes/No, $\Rightarrow$ membership algo doesnot exist.

Theorem 1:

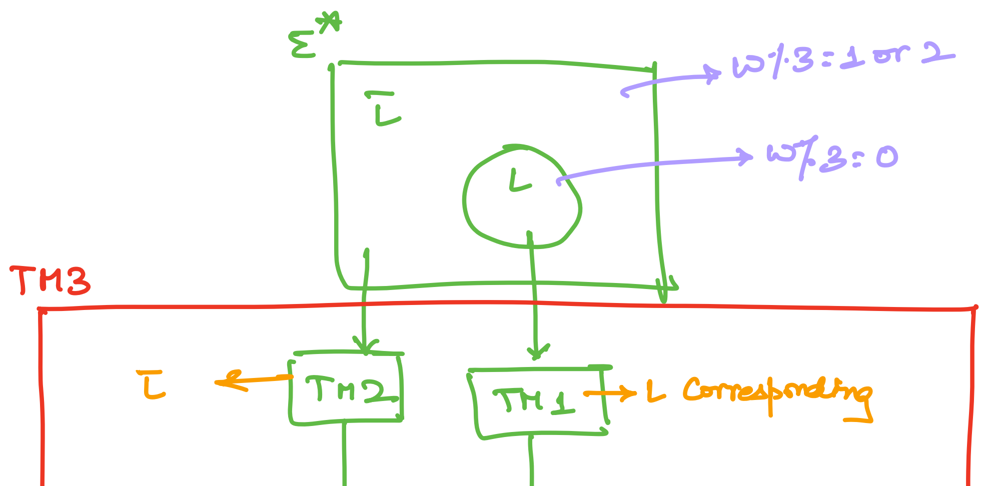
If a language L & its complement $\bar{L}$ both are recursively enumerable then both languages are recursive.

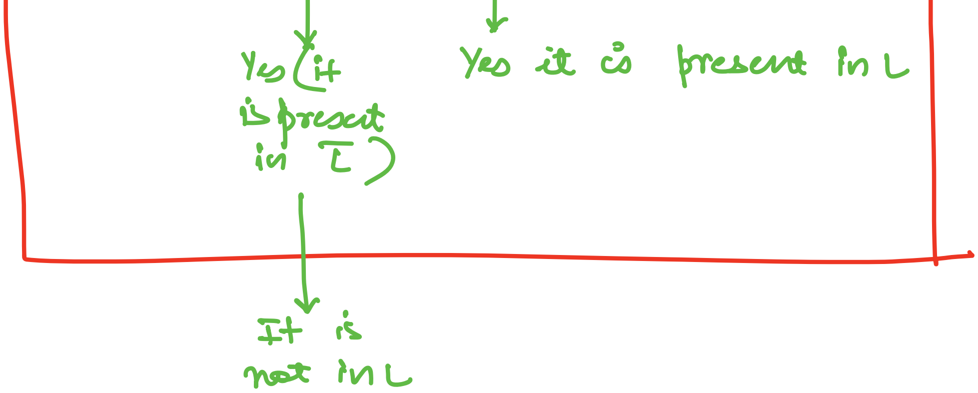


Proof:

$\Sigma = \{0,1\}$

$\Sigma^* =$ Set of all strings





Using TM1 and TM2 create a new TM3

Give strings to both TM1 & TM2

At least one of them will stop & say string is present in L or $\bar{L}$

If TM1 halts at final state : string is present in L

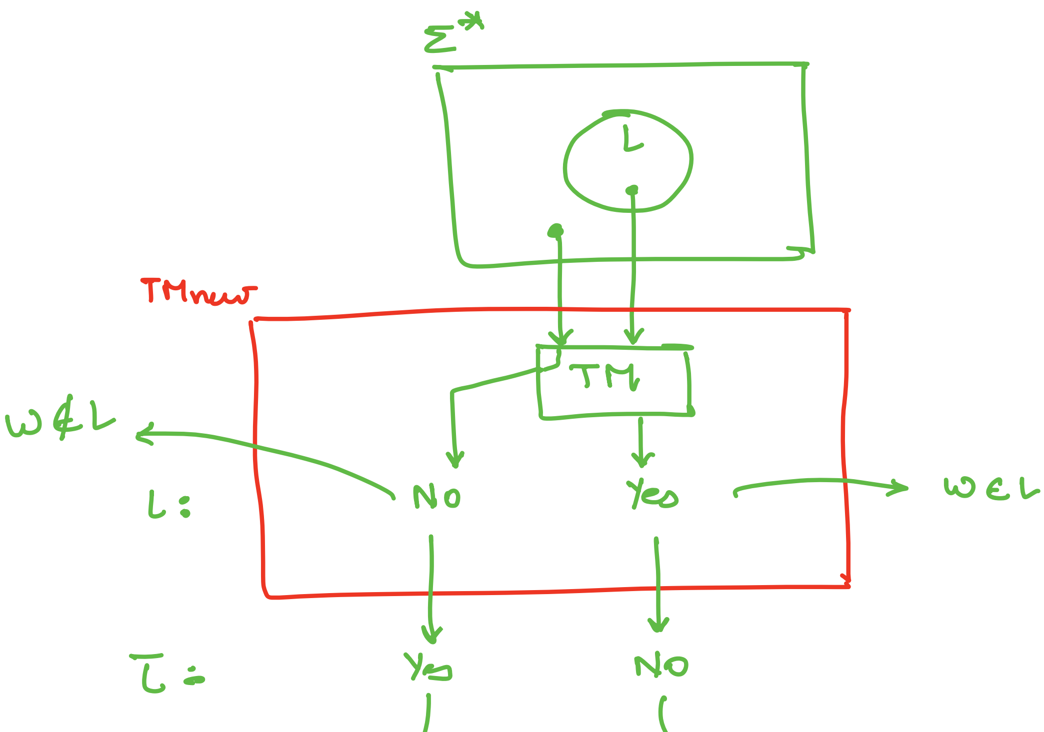
If TM2 ———————— : string is not present in L.

There is a TM3, which will give ans in Yes/NO.

L & $\bar{L}$ are actually recursive.

Theorem 2:

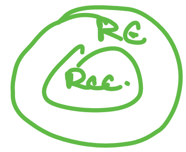
If L is recursive then $\bar{L}$ is also recursive and consequently both are R.E.

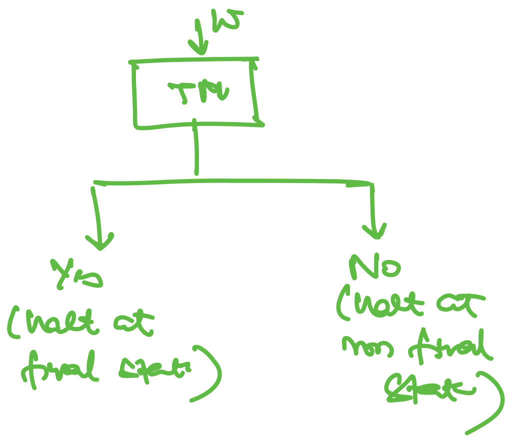
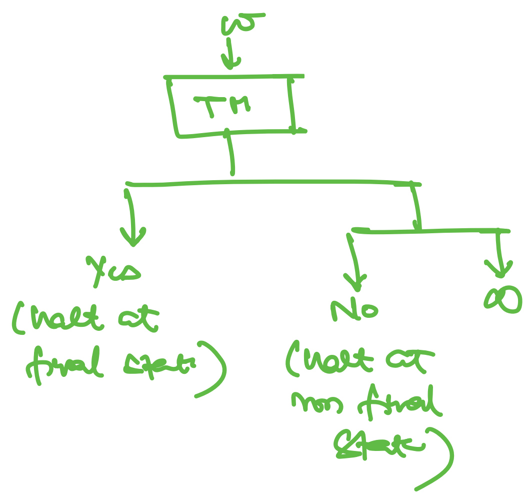


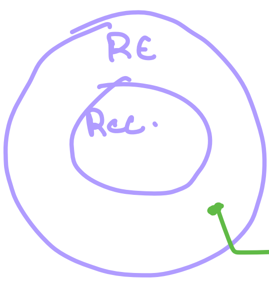
$w \in L$ $\qquad$ $w \notin L$

L is Recursive $\Rightarrow$ Halting TM for L.

Recursive is a subset of RE. Hence, everything that is Recursive is also RE.



Recursive	Recursively Enumerable
→ Halting TM (TM which halts)	→ TM (may halt or may not halt)
	
membership algo exists	no membership algo.



Recursive languages are a proper subset of RE language.

→ this has already been proven that there exist at least 1 language which is RE but not recursive.

Decidability :

Problem   Ans: Yes/NO

→ If there exist an algo to solve this problem then you can say problem is decidable.

eg: number 'n' is prime or not ?

↳ Algo do exist

↳ Decidable.

Halting problem of TM is undecidable.

TM   String w   no algo exists.

There is no algo which can tell us whether our turing m/c will halt or not when string w is provided to it.

Proof by Contradiction :

Assume: Halting Problem is Decidable

If Halting problem of TM is decidable it means there exist an algo which can tell if M will halt or not.

(TM, w) → Algo → Yes → w → TM → Yes or NO

Yes → TM will halt

Algo ↓ NO

TM will not halt

↳ w ∉ L

Yes → string is accepted w ∈ L

NO ↳ String is not accepted w ∉ L

RE there exist a membership algo.

All RE languages are recursive

but this contradicts our fact that there exist atleast 1 language which is RE but not recursive.

Hence, our initial assumption is wrong and Halting problem of TM is undecidable.

Reducability:

$$P_1 \xrightarrow{\text{Algo}} P_2$$

- If P2 is having an algo (P_2 is decidable) it means P_1 will also have an algo (P_1 will be decidable)

  If P2 is decidable then P_1 is also decidable

  Algo for P1: $\underbrace{\text{Convert } P1 \to P2}_{\text{Algo}} + \underbrace{\text{Solve } P2}_{\text{Algo exist}}$

- If it is already proven that P_1 is undecidable then definitely P2 will be undecidable.

State Entry Problem of TM is undecidable.

→ Given a TM, a state $q \in Q$ and $w \in \Sigma^*$

Decide whether or not state 'q' is ever entered when 'w' is given to TM.

TM: $\rightarrow q_0 \xrightarrow{(a,a,R)} q_1 \xrightarrow{(b,b,L)} q_2$

q: q_1 , aba w

Reduction

P1 $\xrightarrow{\text{(algo)}}$ P2

if undecidable then undecidable

Halting problem of TM $\xrightarrow{\text{algo?}}$ State Entry Problem

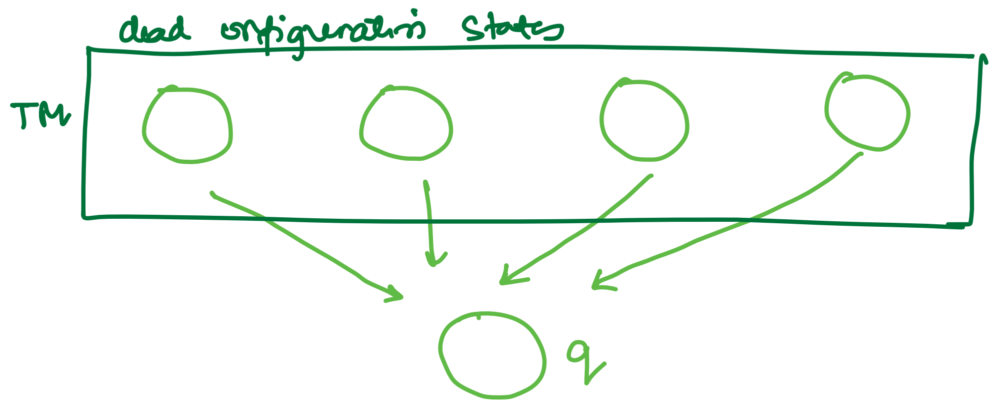
(already proven to be undecidable) (this will also be undecidable)

Convert/Reduce Halting TM problem to State Entry Problem ?

TM halts when it reaches a dead configuration.

m/c is at some state, you have some input symbol, and no transition for state & input symbol.

All the final states of TM are dead configurations; bcz once u reach final state and whatever symbol you look at, no transition is defined.



for every state where there is a dead configuration we give a transition to q.

Actual halting problem is now reduced to halting in state q.

algo?

Halting problem of TM $\longrightarrow$ State Entry Problem

(already proven to be undecidable)

(this will also be undecidable)

Almost all the problems related to RE languages is undecidable.

Reduction

If 2 parse trees exist

Halting TM Problem $\longrightarrow$ Post Correspondence Problem (PCP) $\longrightarrow$ Ambiguity (CFG)

in this entire chain, all problems are undecidable

Post Correspondence Problem

Given 2 sequences of n strings on some alphabet Σ say $A = w_1 w_2 w_3 \cdots w_n$ and $B = v_1 v_2 \cdots v_n$, we say that there exist a PC solution for pair (A, B) if there is a non empty sequence of integers i, j, k such that

$$w_i w_j \cdots w_k = v_i v_j \cdots v_k$$

$n = 3$

A

w_1	w_2	w_3
a	ab	bba

B

v_1	v_2	v_3
baa	aa	bb

You need to find out some sequence of integers in such a way that $w_i w_j \cdots w_k = v_i v_j \cdots v_k$

Sequence: 3 2 3 1 PC Solution

$w_3 \; w_2 \; w_3 \; w_1 \quad = \quad v_3 \; v_2 \; v_3 \; v_1$

bba ab bba a = bb aa bb baa

If you are able to find a sequence then it is called as PC Solution.

PCP is to device an algorithm that will tell us for any (A, B) whether or not there exist a PC Solution.

$w_3 \; w_2 \; w_3 \; w_1 \quad = \quad v_3 \; v_2 \; v_3 \; v_1$

bba ab bba a = bb aa bb baa

Relate it to ambiguity problems in CFG.

You can derive this string in 2 ways from start symbol in such a way that final string is same but intermediate / steps are different.

PC problem is converted to ambiguity problem and PCP is undecidable, so, ambiguity problem will also be undecidable.

Modified PCP

First string from A and first string from B has to be present at starting of solution.

$$w_1 w_i w_j \cdots w_k = v_1 v_i v_j \cdots v_k$$

DECIDABILITY TABLE:

Problem	RL	DCFL	CFL	CSL	Recursive Language	REL
1. Does $w \in L$? (Membership Problem)	D	D	D	D	D	UD

2. Is $L = \phi$? (emptiness problem)	D	D	D	UD	UD	UD
3. Is $L = \Sigma^*$? (completeness problem)	D	UD	UD	UD	UD	UD
4. Is $L_1 = L_2$? (equality problem)	D	UD	UD	UD	UD	UD
5. Is $L_1 \subseteq L_2$? (subset problem)	D	UD	UD	UD	UD	UD
6. Is $L_1 \cap L_2 = \phi$	D	UD	UD	UD	UD	UD
7. Is L finite or not ? (finiteness)	D	D	D	UD	UD	UD
8. Is complement of L a language of same type or not ?	D	D	UD	D	D	UD
9. Is intersection of two languages of same type	D	UD	UD	UD	UD	UD
10. Is L regular language.	D	D	UD	UD	UD	UD