pumping Lemma for Regular language

Note:- pumping lemma is necessary but not ~~necessary~~ sufficient condition.

Let 'L' be a regular language there exists a natural no. 'n' (pumping lemma constant) for every $z \in L$, $|z| \geq n$, for a division of $z = uvw$ satisfying

1) $|v| \geq 1$
2) $|uv| \leq n$
3) $uv^i w \in L$, $\forall i \geq 0$

* pumping lemma is satisfied by every regular language

Eg: $a^*a \Rightarrow \{aa\ aaa\ aaaa\ aaaaa\}$
~~aaaa~~ $i=0$ $i=1$ $i=2$ $i=3$

~~pumping lemma is necessary to satisfy~~

* all regular languages/CFL are satisfies pumping lemma but we can't say ~~whether~~ ~~if~~ CFL/regular if they satisfy the condition (PL), it is not sufficient

* To say ~~the~~ non regularity/CFL we use pumping lemma.

* If a language couldn't satisfy the pumping lemma then it is not regular/CFL.

pumping Lemma CFLs.

* Let 'L' be a CFL then, we can find a natural no. 'n' (PLC) such that every $z \in L$, $|z| \geq n$ can be written as $z = uvwxy$ satisfying

1) $|vx| \geq 1$
2) $|vwx| \leq n$
3) $uv^i wx^i y \in L$, $\forall i \geq 0$

$L_1 = \{a^n b^n c^n d^m / m,n \geq 0\}$ CFL

$S \longrightarrow asd / A$

$A \longrightarrow bAc / \epsilon$

$L_2 = \{a^n b^n c^n d^n$ not CFL because overlap.

$L_2 = \{ww / w \in \{a,b\}^*\}$

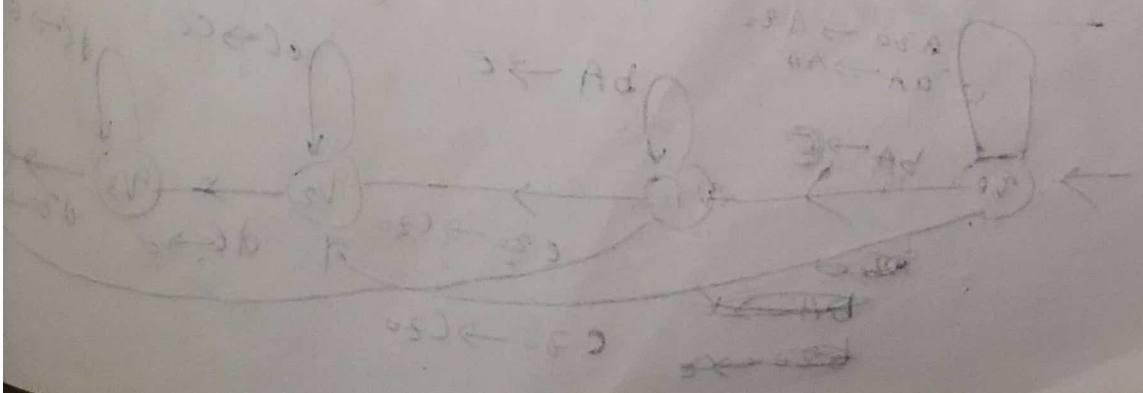
Ex:- abcab not CFL because overlap

if PDA exist CFG and CFG exist then PDA exist

Equivalence b/w CFG and PDA

CFG ⇒ PDA

PDA ⇒ CFG



CFG to PDA

1) Push 'S' on to stack initially

$$\delta(q_0, \epsilon, S) = q_1$$

2) when TOS is NT: & NT's productions is $NT \rightarrow \beta$

$$\delta(q_1, \epsilon, NT) = (q_1, \beta)$$

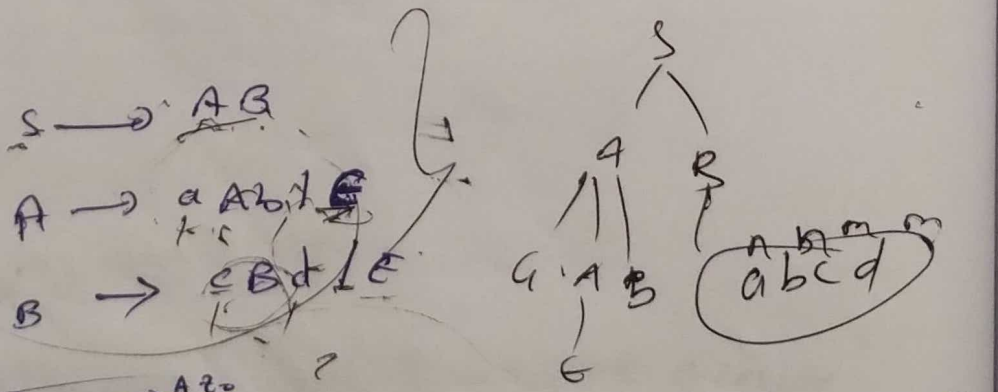
write the above kind of transitions for every NT

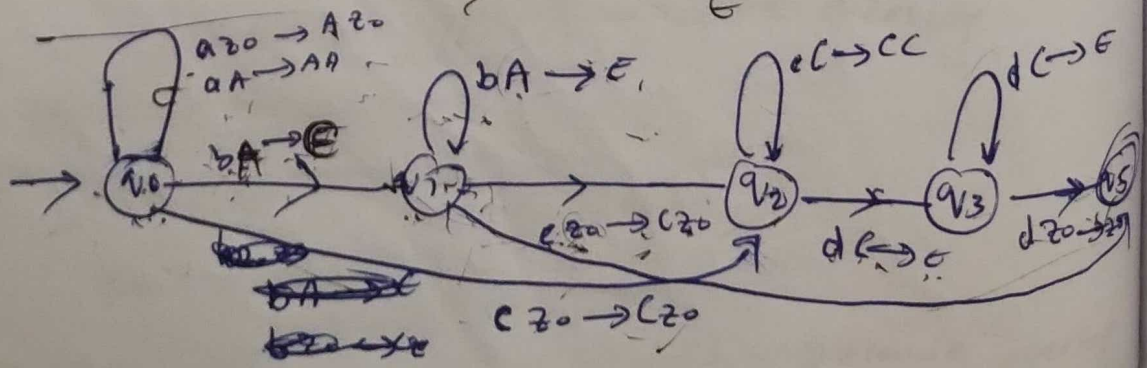
3) when TOS is T: & i/p symbol is same T

$$\delta(q_1, T, T) = (q_1, \epsilon)$$

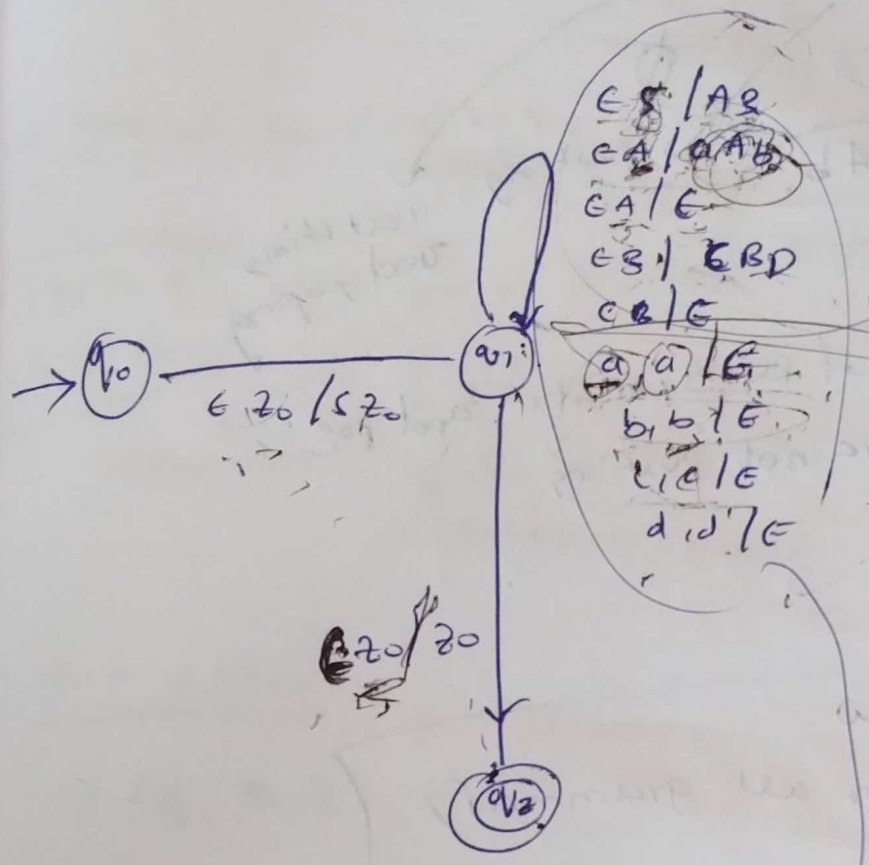
4) $\delta(q_1, \epsilon, Z_0) = (q_2, \epsilon)$

q_2 is final

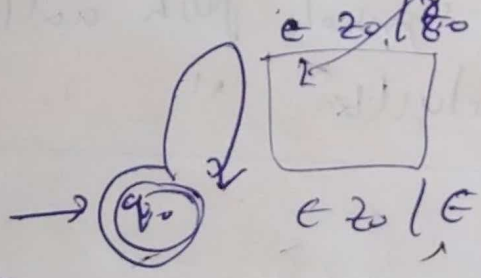


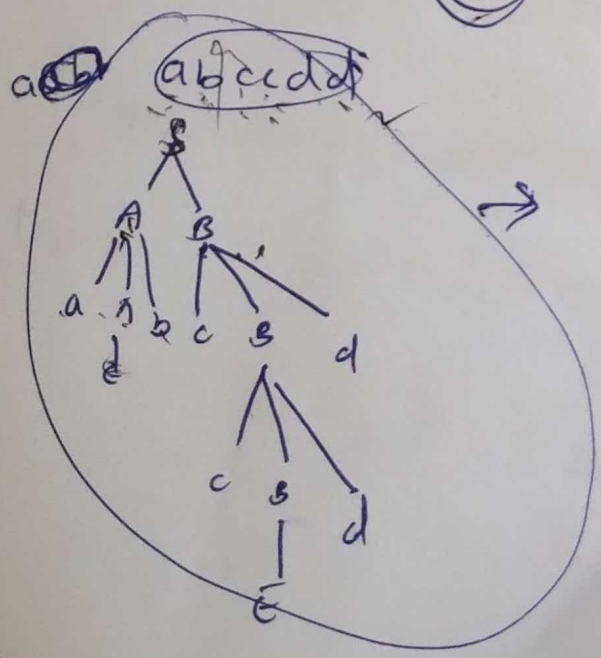


Any CFG is converted into PDA with single state with empty stack acceptance.



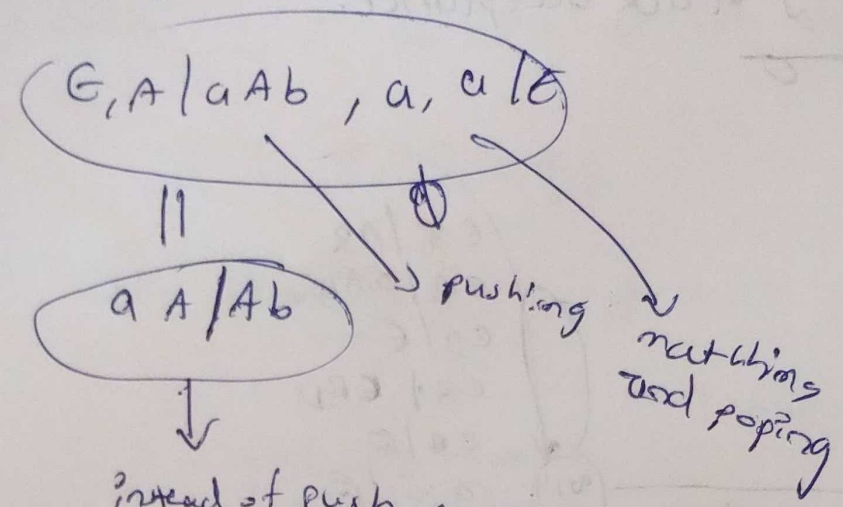
empty stack acceptance





aAb

In GNF grammer



write up (1+/ω

$\epsilon \rightarrow \phi$ (for all gramiers)

---

for all input symbols push action
" " production "

PDA to CFG

$(Q, \Sigma, \Gamma, \delta, q_0, z_0)$ → PDA

↓

(V, Σ, D, S) ⇒ CFG

procedure :-

(1) non terminal production :-

$q_0, q_1 \in Q$

$S \to [q_0\ z_0\ q_1]$

$q \in Q$

$S \to [q_0\ z_0\ q_2]$

$[q_0\ z_0\ q_1]$

$[q_0\ z_0\ q_2]$

initial state

initial stack symbol

$[q\ z\ p]$

(2) pop action

$\delta(q, a, z) \to (p, \epsilon)$ // push down automata

production : $[q, z, p] \to a$

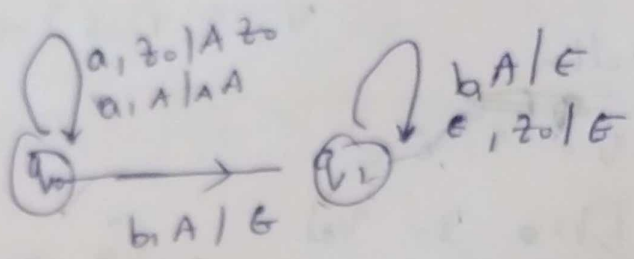
(3) push action

$\delta(q, a, z) \to (p, z_1 z_2 z_3)$

$[q\ z\ \_] \to a\,[p\ z_1\ \_]\,[\_\ z_2\ \_]\,[\_\ z_3\ \_]$

problem :-

$L\{a^m b^n \mid m \geq 1\}$



1) $S \rightarrow [q_0\ z_0\ q_0]$ ✗

$S \rightarrow [q_0\ z_0\ q_1]$

2) $\delta(q_0, b, A) \rightarrow (q_1, \in)$

$[q_0\ A\ q_1] \rightarrow b$

$\delta(q_1, b, A) \rightarrow (q_1, \in)$

$[q_1, A, q_1] \rightarrow b$

$\delta(q_1, \in, z_0) \rightarrow (q_0, \in)$

$[q_1, z_0, q_1] \rightarrow \in$

3) (I) $\delta(q_0, a, z_0) \rightarrow (q_0, A z_0)$

(II) $\delta(q_0, a, A) \rightarrow (q_0, AA)$

(I) $[q_0\ z_0] \rightarrow a\,[q_0\ A]\,[A\ z_0]$

$[q_0 \; z_0 \; q_0] \rightarrow a\,[q_0 \; A \; q_0]\,[q_0 \; z_0 \; q_0]$

$[q_0 \; z_0 \; q_0] \rightarrow a\,[q_0 \; A \; q_1]\,[q_1 \; z_0 \; q_0]$

$[q_0 \; z_0 \; q_1] \rightarrow a\,[q_0 \; A \; q_0]\,[q_0 \; z_0 \; q_1]$

$[q_0 \; z_0 \; q_1] \rightarrow a\,[q_0 \; A \; q_1]\,[q_1 \; z_0 \; q_1]$

(ii) $[q_0 \; A \; q_0] \rightarrow a\,[q_0 \; A \; q_0]\,[\quad A \quad]$

$[q_0 \; A \; q_0] \rightarrow a\,[q_0 \; A \; q_0]\,[q_0 \; A \; q_0]$

$[q_0 \; A \; q_0] \rightarrow a\,[q_0 \; A \; q_1]\,[q_1 \; A \; q_0]$

$[q_0 \; A \; q_1] \rightarrow a\,[q_0 \; A \; q_0]\,[q_0 \; A \; q_1]$

$[q_0 \; A \; q_1] \rightarrow a\,[q_0 \; A \; q_1]\,[q_1 \; A \; q_1]$

$S \rightarrow [q_0 \; z_0 \; q_1]$

$[q_0 \; z_0 \; q_1] \rightarrow a\,[q_0 \; A \; q_1]\,[q_1 \; z_0 \; q_1]$

$[q_0 \; A \; q_1] \rightarrow a\,[q_0 \; A \; q_1]\,[q_1 \; A \; q_1]$

$[q_0 \; A \; q_1] \rightarrow b$

$[q_1 \; A \; q_1] \rightarrow b$

$[q_1 \; z_0 \; q_1] \rightarrow G$

$S \rightarrow A$

$A \rightarrow aBC$

$B \rightarrow aBD \mid b$

$D \rightarrow b$

$C \rightarrow \in$

⇒

$S \rightarrow A$

$A \rightarrow aB$

$B \rightarrow aBb \mid b$

Turing machine

$Q, \Sigma, \tau, \delta, q_0, B, q_{accept}, q_{reject}$

Q : finite set of states

Σ : input alphabet

τ : Tape symbol

δ : Transition function

$\delta : Q \times \{\Sigma \cup \tau\} \rightarrow Q \times \tau \times [L, R]$

(anything → symbol replaces the symbol on tape (may or may not replace) (optional))

q_0 = initial state

B = Blank symbol

q_{accept} : Acceptance state

q_{reject} : reject state.

* can move both directions
* can read/write

advantages

* tape is infinite / memory is large
* head can read and write
* can move in both directions i.e forward and backward and can replace the symbols of tape if we want

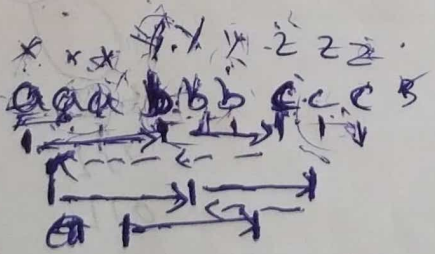
* other than input stream all are filled with blank symbols to identify starting and ending of tape.

L → Represents left.

R → " Right

* Church turing thesis :- if a problem is solvable on computer (i.e algorithm exist) then it is also solvable on turing machine and vice versa

Ex — $a^n b^m c^m$



(first a with first b with first c then next move backward and find second a, second b second c ----.

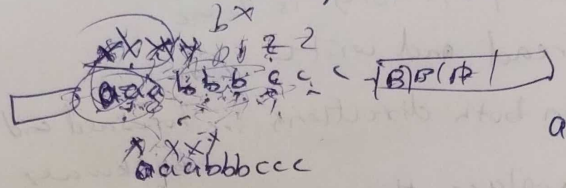
matched symbols are replaced with another symbol.

Ex

a a a b b b c c c
X a a Y b b Z c c
X X a Y Y b Z Z c
X X X Y Y Y Z Z Z

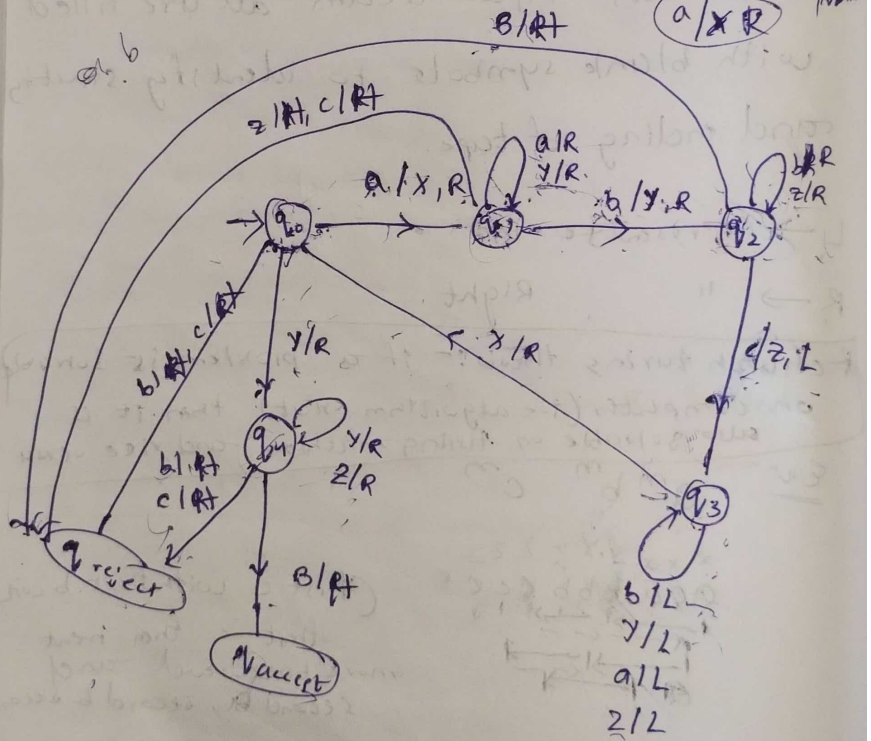
Eg $a^n b^n c^n d^n$

→ can't write c!=b but turing machine is possible.



aaabbbccc

Q) $\{a^n b^n c^n \mid n \geq 1\}$

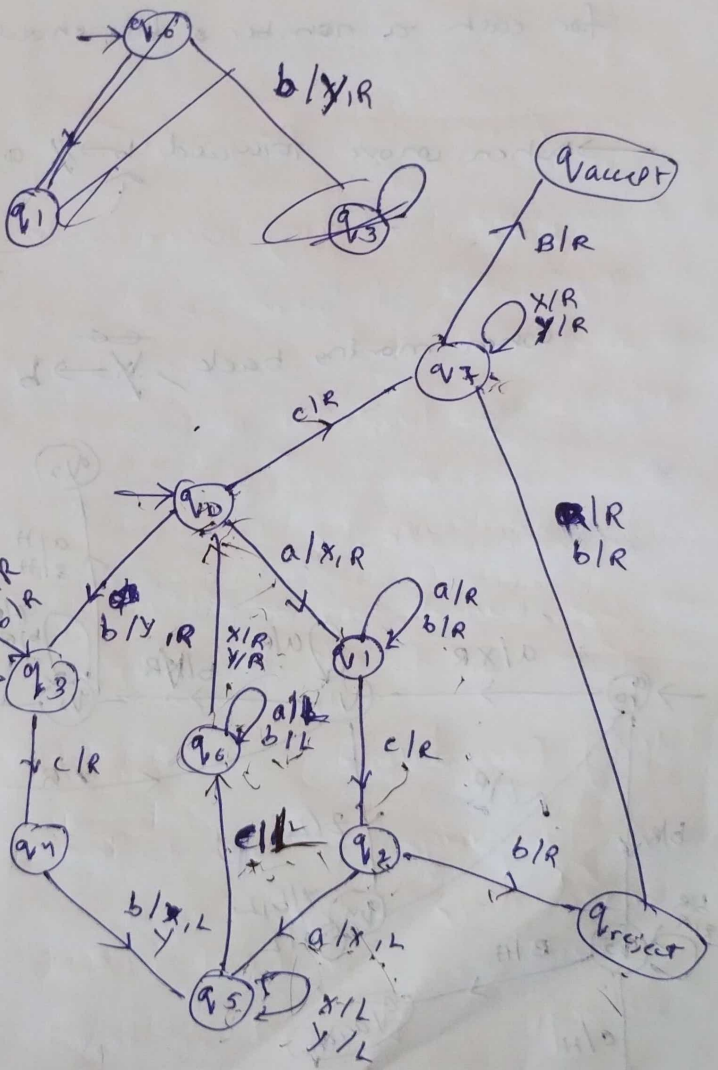


8) $w c w$

aabcaba c aabaaba

aba c aba

ab



6) $\{a^m b^n c^{m \times n} / m, n \geq 1\}$

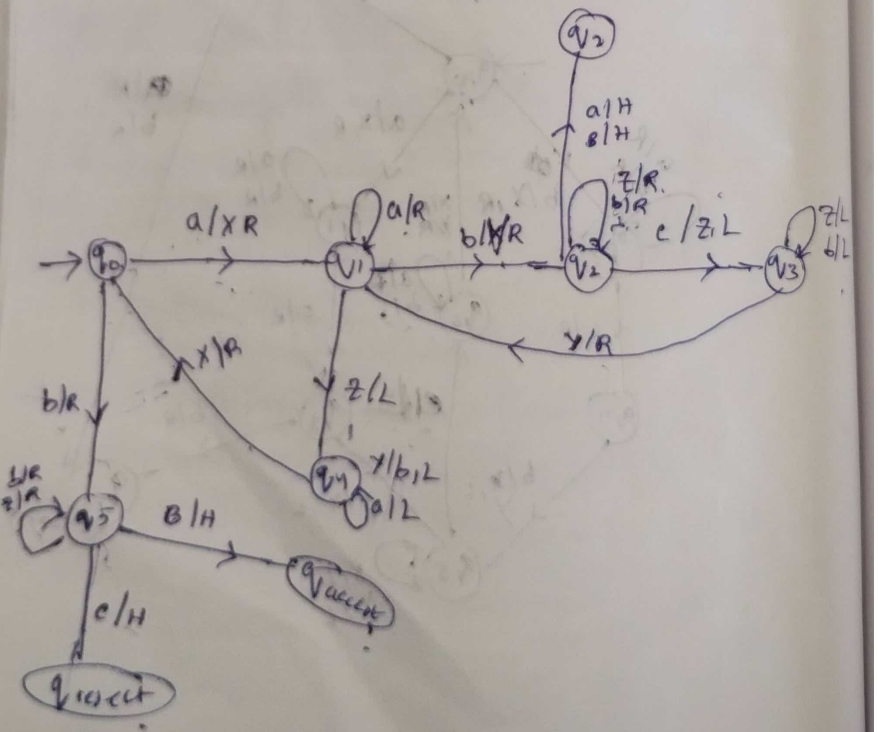
aa bbb ccccc

~~for each a each b should have a c~~

for each a number of b's should have c's

→ when move forward b → Y and c → Z

when moving back, Y → b



a) 0^{2^n}

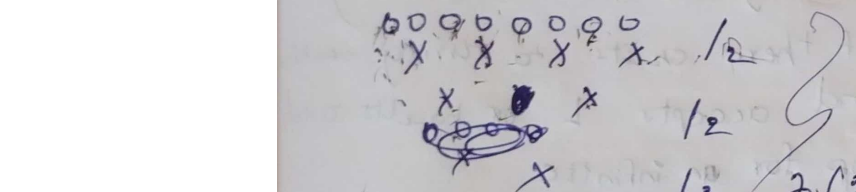
if n = even,
even number of 0's

if n = odd, odd no. of 0's

0 0 0 0 0 0 0
x x x

{00, 0000, 00000000, ---}

logic : prime factorisation



∃ (if odd reveals, then reject)

1) Turing → confident → accept
Acceptable / Recognizable language /
(recursive enumerable language)

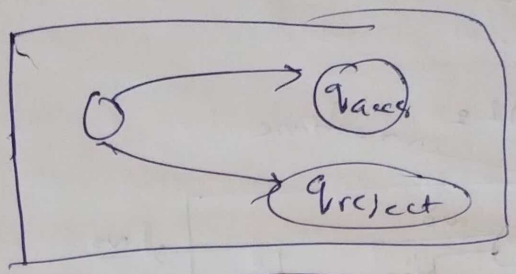
2) Turing decidable language
(Recursive languages)

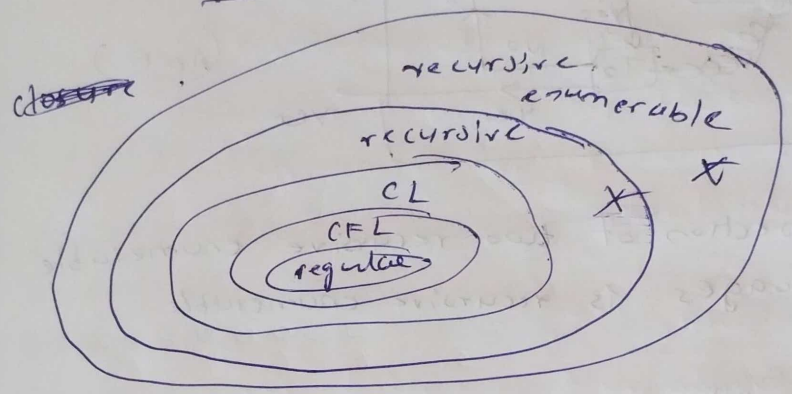
Turing Acceptable Language

* A language is turing acceptable (or) turing recognizable if there exists a turing machine that halts and accepts L or halts and rejects or loop for an infinite
→ we can take decision only if they go to accept state.
→ we cannot say whether it goes to reject state or to infinite loop.

Turing Decidable

* A language is said to be turing decidable or recursive if there exist a turing machine m such that on input x, m accepts if x belongs to L and m rejects otherwise.

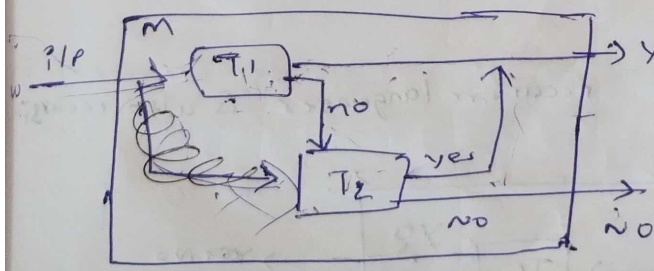




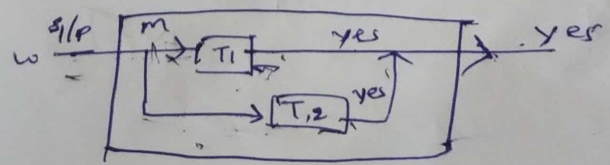
closure properties

* Union of two recursive language is recursive.

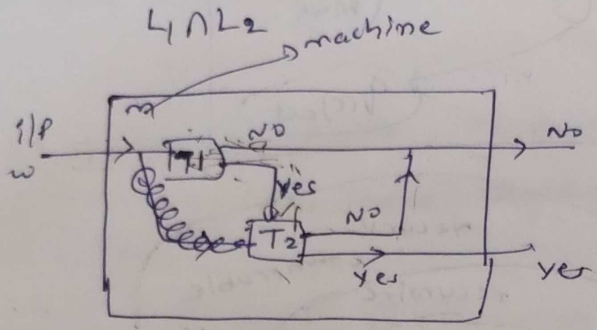
$L_1 \cup L_2$



* Union of two recursive enumerable language is recursive enumerable

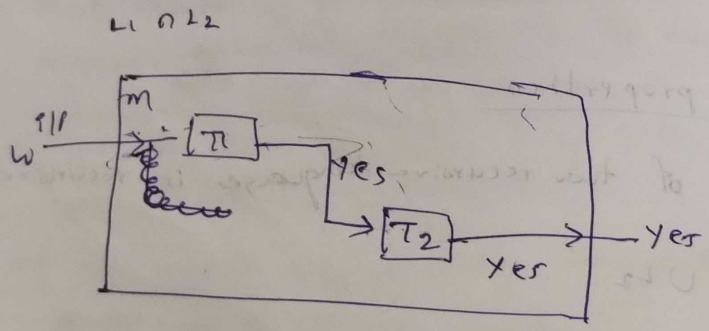


* Intersection of two recursive languages is recursive.

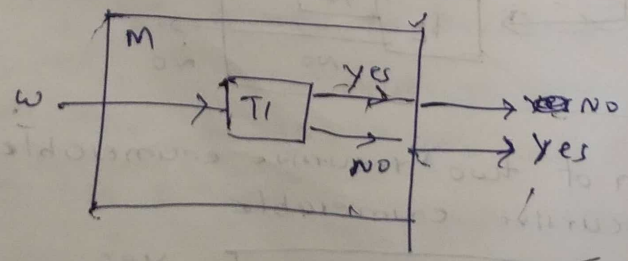


(AND)

* Intersection of two recursive enumerable languages is recursive enumerable



* compliment of recursive languages is also recursive



NOT

* compliment of recursive enumerable is neither recursive nor recursive enumerable.
  → (turing machine does not exist).
  ⊙ because we cannot say Y because we don't know NO, so we cant say compliment.

* concatination of recursive enumerable languages is recursive enumerable.
* concatination of two recursive languages is recursive.
* kleeny closure of recursive enumerable is recursive enumerable →
* kleeny closure of recursive language is recursive language.
* Intersection of recursive and recursive enumerable language is recursive enumerable.
* recursive languages are closed under

* recursive enumerable languages are closed under

decidable (or) solvable

* a language L is decidable if there exist a turing machine M such that M accepts and halts on every string in L; rejects and halts on every string in L compliment.
* computational problem is known as decidable or solvable if the corresponding language is decidable

Recognizable

* A language L is called recognizable if there exist a turing machine M such that M accepts every string in L; either rejects or fails to hault on every string in L compliment.

Undecidable

* An Undecidable language has no decider, there doesnot exist any turing machine which accepts the language and makes a decision by haulting for every input stream.
  → whether a CFG is ambiguous or not is undecidable because we have to check for every input string whether it is ambiguous or not but it is not possible (there is no step by step procedure

* Universal turing machine :- it is a machine that can simulate any turing machine (i.e can behave like an every turing machine).

* whether a turing machine can hault or not is also undecidable.

(PCP is also undecidable)

Post's Correspondable Problem (PCP)

* a instance of PCP consists of two lists of strings over some alphabet sigma (Σ) which are of equal length let the two lists be A, B written as

$A = w_1, w_2, w_3 \dots w_k$ and $B = x_1, x_2, x_3 \dots x_k$

for some integer k. For each i the pair w_i, x_i is said to be corresponding pairs.

* we say this instance of PCP has a solution, if there is a sequence of one or more integers $i_1, i_2, i_3 \dots i_n$, that when interpreted as indexes for strings A and B lists yield the same string i.e $w_{i_1} w_{i_2} w_{i_3} \dots w_{i_m} = x_{i_1} x_{i_2} x_{i_3} \dots x_{i_m}$ we say the sequence i_1 to i_m is the solution to this instance of PCP.

problem

① find out such a sequence of indexes for which both the strings form the same sequence

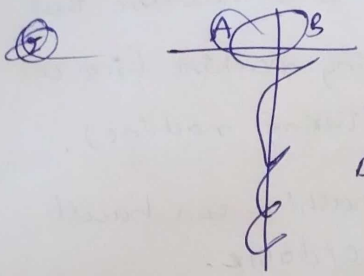
	A	B
1	10	101
2	11	011
3	110	11

sequence 3 2 1 (132)

A 110 11 10

B 110 11 10

(1)



A [11 2] [11] [10]
B [11] [011] [10]

3 2 1

(2)

i	A: w_i	B: x_i
1	110	110 110
2	0011	00 —
3	0110	110 —

1 1 1 1 — —

A [110] [110] — — —
B [110 110] [110 110]

infinitely repeats

2 3

A [0011] [0110] [0110] [—] rejected
B [00] [110] [110]

2 3 1

A [0011] [0110] [110] [ε] → accepted.
B [00] [110] [110 110]

~~* PCP is also undecidability of~~

* PCP is also undecidable.

* PCP is useful for showing undecidability of many problems by means of reducability.

reducability :-

* a reduction is a process of converting one problem into another solved problem in such a way that the solution of the second problem can be used to solve the first problem.

→ let the two problems be A and B where B is a solved problem.

(i) when A is reduced to B, solving A cannot be harder than solving B.

~~(ii) if B is solvable~~

(ii) if A is reducable to B and B is a decidable problem then A is also a decidable problem.

(iii) if A is undecidable problem and reducable to B then B is also undecidable.