Automata

Automata

Alphabate (Σ)

(i) Finite non empty set of symbol.
(ii) Alphabate are atomic.
(iii) 'ϵ' is not used as alphabate symbol.

---

(2) String : Sequence of 0 or more symbol.
(ii) It must have finite length.
(iii) A language can have infinite no. of string of finite length.

---

(3) Concatination

(4) Reversal : $(uv)^R = v^R . u^R$

---

(5) Length of String

$\Sigma = \{a, b, \ldots\}$

String of length 0 : (means no character include)

$|\Sigma|^0 = \{\epsilon\}$

String of length 1 :

$|\Sigma|^1 = \Sigma = (a, b)$

String of length 2 :

$|\Sigma|^2 = \Sigma . \Sigma = (a, b) . (a, b) = \{aa, ab, ba, bb\}$

String of length 3 :

$|\Sigma|^3 = \Sigma . \Sigma . \Sigma = (a, b) . (a, b) . (a, b)$

String of length n :

$|\Sigma|^n = \Sigma . \Sigma . \Sigma \ldots\ldots \Sigma = (a, b) . (a, b) . (a, b) \ldots\ldots (a, b)$

So, Total No. of string upto length n :

$= |\Sigma|^0 + |\Sigma|^1 + |\Sigma|^2 + |\Sigma|^3 + \ldots\ldots + |\Sigma|^n$

$= \dfrac{|\Sigma|^0 (|\Sigma|^{n+1} - 1)}{(|\Sigma| - 1)} = \boxed{\dfrac{(|\Sigma|^{n+1} - 1)}{|\Sigma| - 1}}$ //

$|\Sigma|$ = No. of alphabate

(6) Prefix : (a) $\in$ & w is always prefix of itself.
(b) No. of prefixes in string of length n = (n+1)
(c) No. of proper prefix = n (w not included)

(7) Suffix : (a) $\in$ & w is always suffix of itself. ($\in$ Str $\in$)
(b) No. of suffix of string of length n = (n+1)
(c) No. of proper suffix = n (w not included.

(8) Substring : (a) Consecutive subpart of String.
(b) Every prefix & suffix are string.
(c) Subword in substring excluding "NULL"

(d) No. of Substrings are possible if no repetition is there :

$\Rightarrow$ # Substring of length 0 (include NULL) + # of length 1 + # of length 2 + ------ + # of length n

$\Rightarrow$ 1 + (n + (n-1) + (n-2) + ------ + 1)

$$|S| \Rightarrow 1 + \frac{n(n+1)}{2} \quad \text{(including NULL)}$$

$$|S| = \frac{n(n+1)}{2} \quad \text{(Excluding NULL)}$$

(e) If repetition is there then do as follow :

= # of SS of L0 + # of SS of L1 + - - - - + # of Substring of L n
(unique)

= 1 + (No. of total SS − No. of repeated pair) + - - - -

ex: AXIOMATIZABLE $|S| = 13$

No. of subword ? {A, X, I, O, M, A, T, I, Z, A, B, L, E}

No. of substring of length 1 = 13 − {A repeated 3 times, so keep 1 & remove rest} − {I repeat 2 times, so keep 1 remove rest}

= 13 − 2 − 1 = 10

No. of substring of length 2 = {AX, XI, IO, OM, MA, AT, TI, IZ, ZA, AB, BL, LE}
(No pair is repeating.)

So, if no 2 pairs are repeating it means No pair 3 pairs are repeating and so on.

= (n−1) = 12

No. of substring of l3 : 11

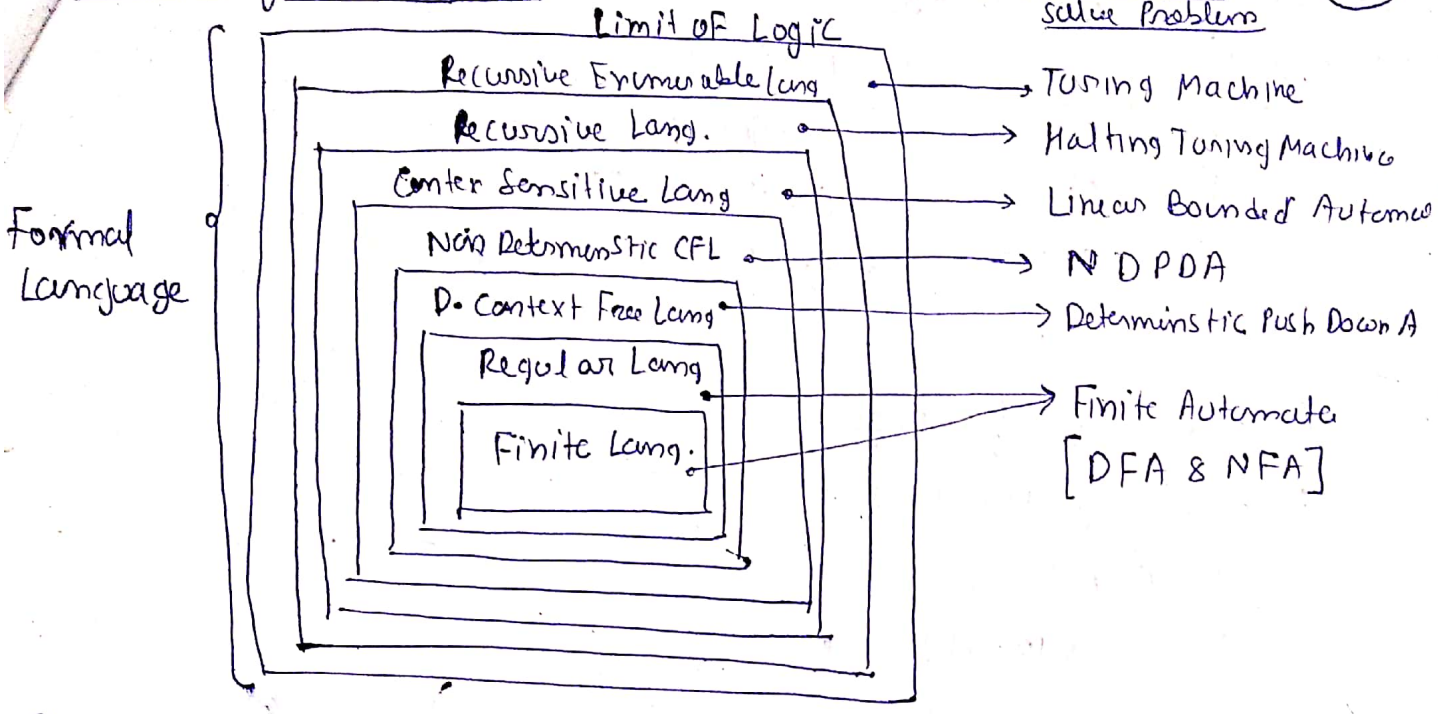
⋮

N : 1

Total = 10 + (12 + 11 + 10 + ---- 1) = 10 + $\frac{12 \times 13}{2}$

Chomsky Hierarchy

Machine To solve Problem



(10) Language representation : (a) Grammer (b) Machine — These two can represent any Language ; (c) Regular Expression — For Finite Automata only.

(11) Operations on Language

(a) Union ($|L_1 \cup L_2| = |L_1| + |L_2| - |L_1 \cap L_2|$)

(b) Intersection ($L_1 \cap L_2 = \{ w_B \in L_1 \text{ && } w \in L_2 \}$)

(c) Complement ($\overline{L_1} = \Sigma^* - L_1$)

(d) Set difference : ($A - B = A \cap \overline{B}$) or ($A - B = A - (A \cap B)$)

(e) XOR : ($L_1 \oplus L_2 = (L_1 - L_2) \cup (L_2 - L_1)$ or $(L_1 \cap \overline{L_2}) \cup (\overline{L_1} \cap L_2)$)

(String not common to L_1 & L_2)

(12) Reversal of Language

$L = (a+b)^* \, ab \, (a+b)^*$

$L^R = (a+b)^* \, ba \, (a+b)^*$

(13) Concatination of Lang : ($L_1 \cdot L_2 = \{L_1\} \times \{L_2\}$)

Eg: $L_1 = \{ a^n, n \geq 0 \}$; $L_2 = \{ b^n, n \geq 0 \}$

here both n are different.

$(\epsilon, a, aa, aaa, \ldots) \times (\epsilon, b, bb, bbb, bbbb, \ldots)$

$= \{ (\epsilon, b, bbb, \ldots) \cup (a, ab, abb \ldots) \cup (aa, aab \ldots) \ldots \}$

$= \{ a^* b^* \}$ or $\{ a^n b^m, n \geq 0, m \geq 0 \}$

4 (14) Power of Language

(a) $L^0 = \epsilon$; $L^1 = L$; $L^2 = L \cdot L$; $L^3 = L \cdot L \cdot L$; ... ; $L^n = L \cdot L \cdot L \cdots$

* $L \cdot \epsilon = L$
* $L \cdot \phi = \phi$
* $L^m \cdot L^n = L^{m+n}$
* $(L^m)^n = L^{mn}$

(b) Cardinality of language = No. of string present in that Language.

(c) A string ($\omega \in L^n$) if and only if ω can be broken into n pieces & each piece belongs to L;

ex. $L = \{10, 01\}$

* Is 10 01 01 0 $\in L^3$

To check: 01 10 (010) → $\notin \{L\}$ (not possible)

* Is 01 10 01 $\in L^3$?

check: 01 10 01 $\in L$ ✓

(d) $L^* = L^0 \cup L^1 \cup L^2 \cup L^3 \cup \cdots \cup L^n$

$L^+ = L^1 \cup L^2 \cup L^3 \cup \cdots \cup L^n$

$L^+ = L^* - \{\epsilon\}$ {only if 'ϵ' not belongs to language.}

ex: $L = \{\epsilon, 0, 1\}$

$\boxed{L^+ = L^*}$ & $L^+ \neq L^* - \{\epsilon\}$ {because ϵ present in Language}

Grammer: (Set of rules that describe all the members of Language.

(b) It is represented by 4 Tuples: (V, T, P, S)

V: (variables) It is denoted by upper case (A, B, C, ----)

T: (Terminals) Item which Language generate, small case (a, b, c, *, +, ---)

P: (Production) • It is set of rules, & It must not empty.

S: (Start) It is the place from where language start generating.

(c) $S \overset{*}{\Rightarrow} \omega$ (By expanding Start Production any no. of time we can generate string)

(d) Sentential Form: It is in between production of grammar & Viable prefix ex: aaSa, bSa, abAaS, ...

Sentence: Final result of a production after expanding

grammar only have one language, but a language can be more than one Grammar. (3)

(v) Derivation : Expanding Production set starting from S to obtain a string.

(b) Types of Derivation :

* Left Most Derivation (Expand left Production first)
* Right Most Derivation (Expand Right Production first)

(c) Yield of Production : A Leaf node of production or final item.

(d) Ambiguity : (More than one way)

If more than one derivation tree (ie either more than one LMD or RMD) possible than string belonging to that grammar is ambiguous that's make grammar ambiguous.

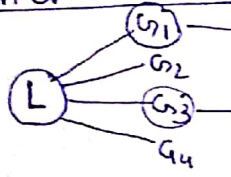
(e) No. of derivation Tree = No. of LMD = No. of RMD (Unambiguous Language)
1 1 1

(vi) Ambiguity of a Language

(a) A Language is ambiguous if every Grammar that generate it is ambiguous.

(b) A Language is unambiguous if at least one grammar that language is unambiguous.

(c) Ambiguity of grammar is removed, if it generate a language which is also generated by a unambiguous grammar.



This ambiguity is removed because, Language corresponding to this grammar have a one unambiguous grammar.

(12) Types of grammar :

(i) Type 0 : (a) It is unrestricted Grammar.

(b) It is RE grammar.

(c) It include all other grammar.

(d) It must have atleast one variable in Left hand Side.

$u \to v$

$aS \to abS$ ✓

$a \to abs$ ✗

No Production here

$u \to (TUV)^* - T^*$

$v \to (TUV)^*$

(6) (ii) Type 1 Grammar:

(a) $u \to v$ [$u = (TUV)^* - T^*$, $v = (TUV)^*$] && [141

(b) Contraction: when n length variable generate production of less than n. ($n > n_p$)

* Contraction is not allowed in Type 1.
  ex: $aS \to a$ X
* Because of contraction production many goes into loop and never halt.
* Non contracting grammar always halt.
* $S \to \epsilon$ is allowed only when, ϵ-production is in language and if $S \to \epsilon$ is allowed then S must not present in RHS of any production. ex: $S \to \epsilon$
  $S \to aSa$ ✓ $S \to aaS$ X
* It is CSL & REC.

(iii) Type 2 Grammar: (a) In it LHS always have 1 variable.

(b) $u \to v$ [$u \to V$ && $v \to (VUT)^*$]

(c) It also Type 0 & Type 1.

(d) No contraction here.

(e) It is Context Free Grammar.

(iv) Type 3 Grammar: (a) In it LHS always have 1 variable.

(b) $[u \to v]$ then $[u \to V$ && $\{u \to T^* + T^*V$ as $u \to T^* + VT^*\}]$

here,

$u \to T^* + T^*V$ (Right Linear)

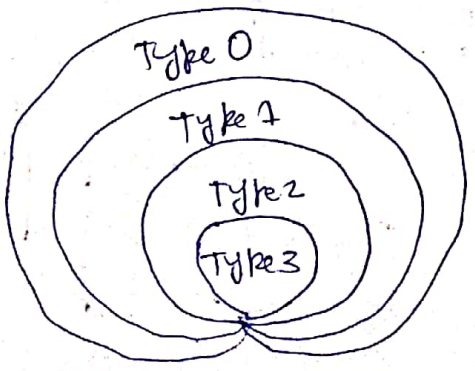
$u \to T^* + VT^*$ (Left Linear)

$u \to T^* + T^*V + VT^*$ (Linear)

(c) It is either Left linear or Right Linear but not both.

(d) It is Regular Grammar.

ex: $S \to aB \mid a$
$B \to aB \mid \epsilon$
} It is regular but not Type 3 because it contains ϵ as well as B present in RHS of grammar



Machine

Melay and Moore

n → output (m) state (7)

Melay → Moore (mn+1)

Moore → Melay (m)

(i) Transducer : It produce output.

(ii) Acceptor : It is use to check wether given string is member or not

(a) Deterministic (No choise)

(b) Non Deterministic (Choise is present with help of copy & replica)

(iii) Types of machine

(a) Finite Automata : ~~xxxxxxxxxxxxxxxxxx~~ . It have only Finite memory. to process only.

* Move one step right only at a time.
* It goes to halt state only when
  - Input ends (DFA)
  - Dead configuration occur (NFA)
* It perform comparison of 1 symbol only.

*In DFA Read Configuration is Trap state.

* Accept as well as reject.
* Power of DFA = Power of NFA
* It can only read.

(b) Push Down Automata :

- It ~~xxxx~~ is Finite Automata + Stack. means move right only one step at a time.
- It have infinite storage, so it can perform one comparison (m==n or m==p) but not two at a time. (m==n & m==p) (Not possible)
- It can only read.
- Power of DPDA ≠ Power of NPDA.

(c) Turing Machine

- It is FA + 2 Stack or PDA + 1 Stack or PDA + PDA
- It can move and (read or write) in both direction i.e left or right.
- It have infinite memory therefore perform any number of comparison.
- Hangs are only allowed in TM, so that Turing machine can reject string.
- Power of DTM = Power of NDTM.

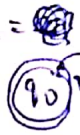
(19) Finite Automata

(a) It define using 5 tuple: $(Q, \Sigma, \delta, q_0, F]$ → Final state (DFA)
- Q → States
- Σ → Input Symbol
- δ → Transition Function
- q_0 → Initial state

(b) It have unique initial state.

(c) If FA is empty then one state is present and Language is also empty. No Final state. But vice versa not true.

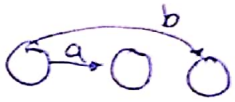
} Final state is empty, so language is also empty.

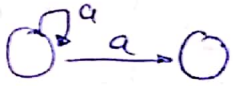


} Language is empty but Final state is there

(d) Transition Function (δ) can be represented using: Function method, Table and State diagram.

Function $\{\delta(q_i, X) = q_j\}$
- q_i → Present state
- X → Input Symbol
- q_j → Next State

(e) DFA → (No edges have NO choices) ex: 

NFA → (edges allow, have choice) ex: 

(f) ∈-Move: It is way to change from one state to other

(g) ∈-NFA: It is NFA with NULL move.

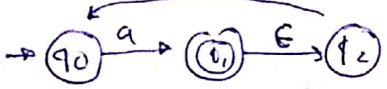
(20) Transition function of Finite Automata

(i) For DFA: $\delta[Q \times \Sigma \to Q]$

(ii) For NFA: $\delta[[Q \times (\Sigma \cup \{\in\})] \to 2^Q]$
It include all states including DC.

(iii) Due to Dead configuration in NFA, we can skip some states. because of this No. of states in NFA is always less than equal to 2^Q.

(iv) Extended Transition function (δ^*): It tells us for a given string on some state what are the possible states we reach.
Σ^* → It represent string.

ex: 

For DFA, $\delta^* Q \times \Sigma^* \to Q$

For NFA, $\delta^* Q \times (\Sigma \cup \{\in\}) \to 2^Q$

$\delta^*(q_0, \in) = \{q_0\}$

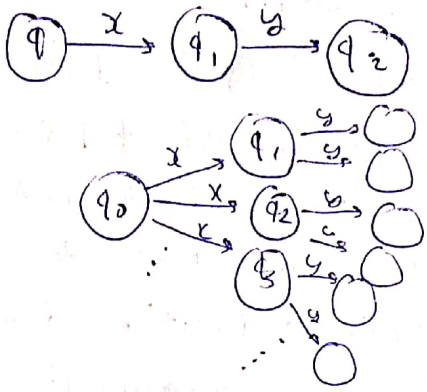
$\delta^*(q_0, a) = \{q_1, q_2, q_0\}$

For DFA, $\delta^*(q, \in)$: It represent null character i/p. $\delta^*(q_1, \in) = \{q_1, q_2, q_0\}$

$\delta(q, xy) \rightarrow$

DFA: $\delta^*(\delta^*(q,x), y)$

NFA: $q \in \delta^*(q,x) \cup \delta^*(q,y)$



(vi) **Right Invariance Rule:**

$xz \equiv yz$

$\delta^*(q, x) = \delta^*(q, y)$

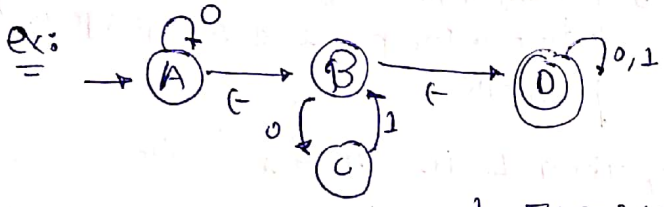
$\delta^*(q, xz) = \delta^*(q, yz)$

(vii) **ϵ-Closure**: what are the possible state reachable.

(b) Structure:

State	ϵ*	Input	ϵ*

ex:



ϵ-closure for D

	ϵ*	0	ϵ*
A	A	A	A
	B	C	B, C
	D	D	D

A → {A, B, C, D}

This technique is use to remove NULL-transition from NFA, after it we get NFA without NULL Move.

(21) **FAutomata Theorems**: * It cannot perform (+, −, /, *).

(i) $L(\bar{M}) = \overline{L(M)}$

For DFA: $L(\bar{M}) = \overline{L(M)}$

For NFA: $L(\bar{M}) \neq \overline{L(M)}$ (Because of ϵ Move & Dead Configuration)

(ii)
- Every Finite Language is Regular because no. of states are known to us.
- Every regular language have regular expression.
- FA & Regular expression have same power.

(iii) **Pumping Lemma For RL**: If a language satisfy pumping lemma for regular, then it can be regular Language, but if it not satisfy then it is surely not regular. (It is negativity test).

(iv) **Myhill Nerode Class**: It is minimum no. of state present in DFA.

1 state is 1 class. | No. of Myhill Nerod class = Minimum state in DFA = Index of L lang.

iff ≡ ⇔ (Bidirectional)

(10) (v) Kleen's Theorem :

(a) L is regular iff ∃ FA that accept L.

(b) L is regular iff ∃ DFA that accept L.

(c) L is regular iff ∃ NFA with one Final state.

- NFA with Single Final state have same power as multiple Final State.
- We can conve Multible Final state to NFA with single Final state.

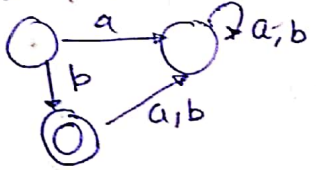
[* L is regular iff ∃ NFA without ε Moves]

(d) L is regular ∃ NFA without DC.

(e) L is regular ∃ NFA without choice.

(f) L is regular iff ∃ regular expression for L.

(g) Presence of loop in FA, not means language is infinite.

ex: 

} Language only accept (b), but there is loop and Language is not infinite.

(h) If in FA no cycle present then L is surely finite.

(i) If L is regular, ∃ regular grammar for it.

(j) If L is regular then there can be non regular grammar as well as Machine that produce & accept L.

(k) If L is regular then ⇔ ∃ right Linar grammar
⇔ ∃ Left " "

(l) If L is regular then ⇒ ∃ Linar grammar.
but not opposite. ex: S → aS | B
B → Ba | a
} Linar grammar but not regular.

(m) Every Left Linear Grammar & Right Linear grammar are interconvertable.

(22) Regular Expression with DFA & NFA

(a) $L(r^*) = (L(r))^*$, Language generated by r^* is equal to Language generated by r, then it kleene *.

(b) Precedence : ( ) > * > · > + ex: $a \overset{3}{\oplus} b \overset{2}{\cdot} c^{\overset{1}{*}}$

(c) In case of Binary alphabate {0,1} : all s & max true 3 bit

* Not Starting with '001'?

$(000 + 001 + 010 + 011 + 100 + 101 + 110 + 111)\cdot(0+1)^*$ + (2bit string) $(0+1)(0+1)$ + (1 bit & 0 bit str) $0 + 1 + \epsilon$

(write all possibility & then remove not required)

$(000 + 010 + 011 + 100 + 101 + 110 + 111)\cdot(0+1)^* + (0+1)(0+1) + 0 + 1 + \epsilon$

write a regular expression:

ex:

(a) $n(a) \bmod 3 == 1$ or $n(a) \bmod 3 == 2$ $\quad \Sigma = \{a, b\}$

$$\left[(b^* a b^* a b^* a b^*)^* + b^* \right] (a b^* + a b^* a b^* a b^*)$$

- $(b^* a b^* a b^* a b^*)^*$: Generate string contain exactly 'a's multiple of 3
- b^* : a's multiple of 3
- $(a b^* + a b^* a b^* a b^*)$: Base case. No. of a's i.e 1 or 2

No. of a's mod 3 == 1 or 2

(b) $\{w \mid |w| \bmod 3 == 0\} \Rightarrow ((a+b)^3)^*$

$\{w \mid |w| \bmod 3 == 1\} \Rightarrow ((a+b)^3)^* (a+b)$

$\{w \mid |w| \bmod 3 == 1\} \Rightarrow ((a+b)^3)^* (a+b)^2$

(c) $\{w \mid |w| \bmod 100 == 23\} \Rightarrow ((a+b)^{100})^* (a+b)^{23}$

$\{w \mid |w| \bmod 100 <= 23\} \Rightarrow ((a+b)^{100})^* (\epsilon + a+b)^{23}$

$\{w \mid |w| \bmod 100 >= 23\}$

$\Rightarrow ((a+b)^{100})^* [(a+b)^{23} + (a+b)^{24} + (a+b)^{25} + \dots + (a+b)^{99}]$

{here, upto 99 because at 100, total length become 200 and its mod with 100 is 0.}

$\Rightarrow ((a+b)^{100})^* [(a+b)^{23} (\epsilon + (a+b) + (a+b)^2 + \dots + (a+b)^{76})]$

$\Rightarrow ((a+b)^{100})^* [(a+b)^{23} (\epsilon + a+b)^{76}]$ //

here,

$(\epsilon + a+b)^{76}$ = because of ϵ, we can make any length of string upto 76.

ex: $(\epsilon + a+b)^{76} = (\epsilon + a+b)(\epsilon + a+b)^{75}$

$= a (\epsilon + a+b)(\epsilon + a+b)^{74}$

$= a \cdot \epsilon \cdot (\epsilon + a+b)(\epsilon + a+b)^{73}$

$= a \cdot \epsilon \cdot \epsilon \cdot \epsilon \cdots \epsilon$ 76 times

$= a$ //

(12)

(v) Design DFA For Decimal equivalent of binary:

(a) $L = \{ w \mid d(w) \bmod 4 == 0 \}$ $\quad \Sigma = \{0, 1, 2\}$

Algo:

(i) Find the mod value and take State 1 less than it, ie $\{0, 1, 2, 3\}$

(ii) Find out no. of input i.e. (0, 1, 2)

(iii) Draw table and write all the state in column one below other.

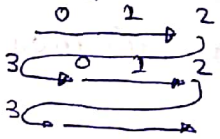
(iv) Write input in one row top of table.

(v) Fill the State value starting from (0,0) by 0, after that 1 and so on in cycle till Final State.

		0	1	2	← Input
Possible State	q_0	0	1	2	
	q_1	3	0	1	
	q_2	2	3	0	
	q_3	1	2	3	

all mod value possible

Filling order



(vi) Finally, Draw its equivalent DFA.

(vi) $(a^* b^* \cap b^* a^*)$

$\Rightarrow (\epsilon + a^* + b^* + a^* b^*) \cap (\epsilon + a^* + b^* + b^* a^*)$

$\Rightarrow (a^* + b^*)$

Write all possibilities that is generated by regular expression. Then perform union or intersection or any set operation.

(ii) $\Pi^* = \epsilon + \Pi\Pi^*$ (For NFA, to convert Regex to NFA)

(iii) Before designing DFA from Regex, check all smallest possible String generated by that regex, after that find the point from where it start looping.

(i) General Form: $L = \{ a^{k_1 n + k_2} ; n \geq 0 \}$

(a) If $k_1 > k_2$: Then it is mod machine & k_2 is its residue

(b) If $k_2 > k_1$: Then it is normal machine. $(a)^{k_2} \cdot ((a)^{k_1})^n$

(c) If $k_1 = k_2$: Then it is normal machine with $(k_1 + 1)$ State.

General Form: $L = \{ a^{kn} ; n \geq 0 \}$

$(a^k)^n$ it have (k) State.

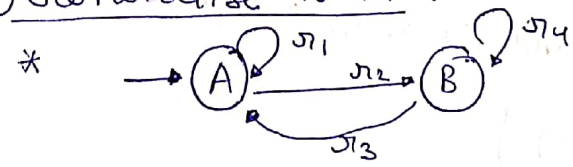
$L = \{ a^{kn} ; n \geq 1 \}$

$(a^k)^n$ it have (k+1) State

Conversion from NFA/DFA to Regular Expression

(i) Remove trap (ii) Remove unreachable state
(iii) For more than one final state write as $\Pi = \Pi_A + \Pi_B + \cdots$
(iv) Remove Dead Configuration state.

(v) **Generalise Form :**



(a) No Final state : $\Pi = \phi$

(b) **Final State A :**

$$\Pi_A = (\Pi_1^* + \Pi_2 \Pi_4^* \Pi_3)^*$$

(c) **Final State B :**

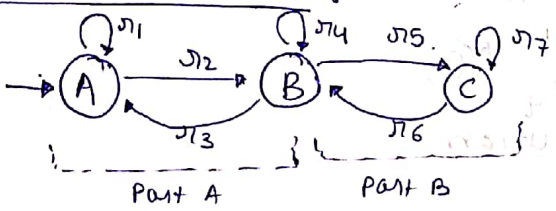
We have two ways :

Left Resolution : $\Pi_B = (\Pi_1 + \Pi_2 \Pi_4^* \Pi_3)^* \Pi_2 \Pi_4^*$

Right Resolution : $\Pi_B = \Pi_1^* \Pi_2 (\Pi_4 + \Pi_3 \Pi_1^* \Pi_2)^*$

---

* **3 State Machine :**



(a) No Final state : $\Pi = \phi$

(b) **A is Final state :**
- Reduce Part First to B.
- Then reduce all to A.

$$\Pi_A = (\Pi_1 + \Pi_2 (\Pi_4 + \Pi_5 \Pi_7^* \Pi_6)^* \Pi_3)^*$$

(c) **B as Final state :** First reduce C to B.

We have two options :

Left Resolution : $X = (\Pi_4 + \Pi_5 \Pi_7^* \Pi_6)^*$

$$\Pi_B = (\Pi_1 + \Pi_2 X^* \Pi_3)^* \Pi_2 X^*$$

Right Resolution : $X = (\Pi_4 + \Pi_5 \Pi_7^* \Pi_6)^*$

$$\Pi_B = \Pi_1^* \Pi_2 (X + \Pi_3 \Pi_1^* \Pi_2)^*$$

(d) **C as Final state :** 4 ways are possible :

(i) In terms of B : $\Pi_C = \Pi_B \Pi_5 \Pi_7^*$

$$\Pi_C = \Pi_1^* \Pi_2 (X + \Pi_3 \Pi_1^* \Pi_2)^* \Pi_5 \Pi_7^*$$

---

(3) Properties of Regular Expression :

(i) **Commutative :**
(a) $\Pi + S = S + \Pi$
(b) $\Pi \cdot S \neq S \cdot \Pi$

(ii) **Associative**
$(\Pi + S) + t = \Pi + (S + t)$
$(\Pi \cdot S) \cdot t = \Pi \cdot (S \cdot t)$

(iii) **Distributive**
$\Pi \cdot (S + t) = \Pi \cdot S + \Pi \cdot t$
$(S + t) \cdot \Pi = S \cdot \Pi + t \cdot \Pi$

(iv) **Identity**
$\Pi + \phi = \phi + \Pi = \Pi$ & $\Pi \cdot \epsilon = \Pi$ { ϕ is for at, ϵ is for . } act as identity element

(14)

(v) $\phi^* = \epsilon$ {Because if we take 0 length string from empty set it is always}

(vi) $r + r = r$

(vii) $r \cdot r = r^2$

(viii) $\epsilon + r r^* = r^*$

(ix) $r^* + r^* = r^*$

(x) $r^* \cdot r^* = r^*$

(xi) $(r^*)^* = r^*$

(xii) $R + \phi = R$

(xiii) $R \cdot \phi = \phi$

(xiv) $R + \Sigma^* = \Sigma^*$

(xv) $\phi^+ = \phi$

(xvi) $\epsilon^* = \epsilon^+ = \epsilon$

(xvii) $\epsilon + r + rrr^* = r^*$

(xviii) $r^* \cdot r^+ = r^+$ $\left((\epsilon + r^+)(r^+) = (r^+ + r^+ r^+) \right)$

(xix) $r^+ \cdot r^* = r^+$ $\left((r + r^+)(\epsilon + r^+) = (r + r^+ r^+ \dots) \right)$

(xx) $r^+ \cdot r^+ = rr r^*$ $\left(r^+ r^+ = r^* - \{\epsilon, r\} = (rr + rrr + rrrr \dots)(r + rr + rrr \dots) = (rrr + rrrr \dots) \right)$

(xxi) $(r^*)^* = r^*$

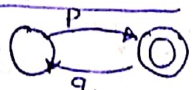
(xxii) $(r^+)^+ = r^+$

(xxiii) $(r^*)^+ = r^*$

---

(24) Major Properties of regular expression

(i) $(pq)^* p = p(qp)^*$



(a) It is like left and right resolution

(b) Proof: $(pq)^n p = p(qp)^n$

For $n = 1$, $(pq)p = p(qp)$ {associativity property}

Now, For $(n+1)$

$$(pq)^{n+1} \cdot p = (pq)^n \cdot (pq) \cdot p = (pq)^n \cdot p\,(qp)$$

$$= p(qp)^n \cdot (qp) = p(qp)^{n+1}$$

$$\boxed{(pq)^{n+1} \cdot p = p(qp)^{n+1}} \quad //$$

(ii) $(r^* + s^*)^* = (r^* s^*)^* = (r^* + s)^* = (r + s^*)^* = (s^* r^*)^* = (r + s)^*$

(a) To prove it we have to try to generate possible strings:

- $(r^* + s^*)^* = \{\epsilon, r, s, rs, sr\} = (r+s)^*$
- $(r^* s^*)^* = \{\epsilon, r, s, rs, sr\} = (r+s)^*$
- $(r^* + s)^* = \{\epsilon, r, s, rr \dots, ss \dots, rs, sr\} = (r+s)^*$
- $(r + s^*)^* = \{\epsilon, r, s, rr \dots ss \dots, rs, sr\} = (r+s)^*$
- $(s^* r^*)^* = \{\epsilon, s, r, ss \dots, rr \dots, rs, sr\} = (r+s)^*$

If s and r are deprecatly generated and it is $(\ )^*$ then it is written as $(r+s)^*$

$(R+S)^* R^* = R^* (R+S)^* = (R+S)^*$

(c) Here, by not considering 'S':

$(R+S)^* R^* = R^* . R^* = R^*$

(iv) if $R_1 \subseteq R_2$, $R_1 + R_2 = R_1$

ex: $(a+b)^* (a+aa)$ } ending with a & aa
$a \subseteq aa$

$\Rightarrow (a+b)^* a$

ex: $(a+b)^* (a + ba + bb)$ } ending with a, ba & bb
$a \subseteq ba$.

$\Rightarrow (a+b)^* (a+bb)$

(v) Arden's Theorem :

(a) $\boxed{R = P + QR}$ (Left recursion) (q must not be NULL)

$\boxed{R = PQ^*}$

Verification

$\boxed{R = P + QR}$

$\boxed{R = Q^* P}$

Proof:

$R = P + (P + RQ) \cdot Q$

$R = P + PQ + RQQ$

$R = P(\epsilon + Q + RQQ) = PQ^*$

$\boxed{R = PQ^*}$ //

(b) $\boxed{R = PR + Q}$ (Right recursion)

$\boxed{R = P^* Q}$ //

(c) * Starting state always have NULL.
* Write equation for every incoming arrow for each state.
* Format : State input ex: $A = \epsilon + Ba$ → 

(d) Arden's Theorm is useful when, we are given a Finite Automaton and ask to check whether language accepted by 2 states are same or not or what

(16) (M)

(25) DCFL and Regular Language are always unambiguous

(a) because they are deterministic but regular grammar or DCFL grammar can be ambiguous.

(b) Machine to Grammar, in it final state always accept ∈.

(c) For Complex Grammar, to find its language use grammar to machine conversion. Only if Grammar is RLRG.

(d) NFA and DFA does not allow multiple initial state. It is only allow by Transition System. If multiple initial states are present then we make it into one with help of ∈-moves.

(V) Algorithms For Finite Automata

(a) Conversion of ∈-NFA to NFA without ∈-Move

Find ∈-closure → → →

	∈	a	∈
→	A		A
	B		B
	C		C
	D		D

Given $A = \{A, B, C, D\}$ on input a

State | $\in^*$ | Input | $\in^*$

A

- $\in^*$: Possible state which is reachable with ∈-move including current one.
- Input: Possible state which is reachable with input
- $\in^*$: Possible state which is reachable with help of ∈-move

No. of state: $n \leq \quad \leq 2^n$

It gives NFA without ∈-Move

(b) Conversion of NFA with ∈-Move to DFA (Not Minimal DFA)

* Subset Construction algorithm

NFA = $\{ M(Q, \Sigma, \delta, q_0, F) \}$

DFA = $\{ M'(Q', \Sigma', \delta', q_0', F') \}$

$M \equiv M'$

(i) $Q' \subseteq 2^Q$

(ii) $\Sigma' = \Sigma$

(iii) $q_0' = \{q_0\}$

(iv) $F' \subseteq Q'$

(c) Conversion of DFA to Minimal DFA

If $A \equiv B$, then $\forall w \in \Sigma^*$

$\delta^*(A,w)$ & $\delta^*(B,w)$ both accept & goes to final or both reject and on non final state

k-equivalent | $\pi_{k+1} = \pi_k$

$A \equiv_k B$ iff $\forall w \in \Sigma^*$ & $|w| \leq k$

(a) $\delta^*(A,w)$ & $\delta^*(B,w) \subseteq F$

(b) $\delta^*(A,w)$ & $\delta^*(B,w) \not\subseteq F$

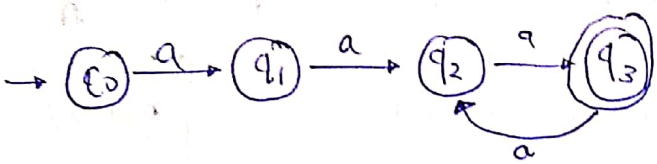
Ambiguous

Pumping lemma For Regular

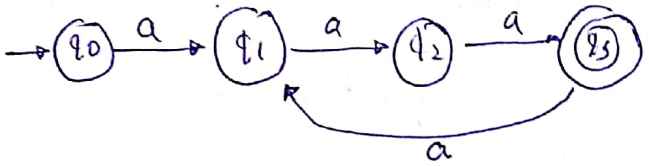
If L is regular ⇒ ∃ FA) with n state such that $w \in L(G)$, and w divide into $x y^i z$.

Ex: Why n & 2n-1? For infinite ($n \le |w| \le 2n-1$)

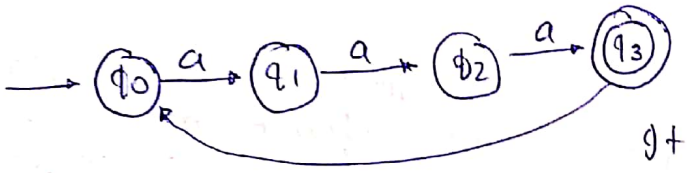
Let, n=4



If |w| = n
(may or may not be accepted)



If |w| = n+a
(may or may not be accepted)



If |w| = 2n-1
It surely have to accept

So, |w| is already greater than n and when (2n-1) it is surely accepted that signify that infinite length string is accepted by FA, so FA can have infinite Language. and we know that string can never be infinite.

(b) Pumping lemma state that there is pattern exist:
$w = x y^i z$ where $\{i \ge 0, 1, 2, \ldots n\}$

(c) If string length is in b/w $|w| \le |w| \le |2n-1|$ then Language may or may not be infinite, but if string length is $|w| = 2n-1$, then language surely is infinite.

---

(27) **Myhill Nerode Theorem**: Myhill Nerode Equivalence classes are regular expression of all state.

If L is regular ⇔ No. of Myhill Nerode equivalence class is finite

(28) **Indistinguishable String**: It is use to check equivalence of 2 string. u R v iff $\delta^*(q_0, u)$ lead to same State as $\delta^*(q_0, v)$ (Final or non Final)

(18) (29) **Closure Properties:**

Regular → Closed under all

DCFL → Closed under $\bar{L}$ & Inverse Homomorphism

CFL → Closed under all except Intersection & $\bar{L}$ (Complement)

CSL → Closed under all

REC → Closed under all

RE → Closed under all except $\bar{L}$.

For These 3, (Transpose as Reversal, Homomorphism and Inv. Homo is not define)

(30) **Homomorphism:** It is a function which is used to convert one form of language to other.

(31) **Closure Property not satisfy by any languages are:**

$\supseteq$, $\subseteq$, infinite union, infinite intersection, infinite difference

ex: $L_1 - L_2 = L_1 \cap \bar{L}_2$, $L_1 \uparrow L_2 = \overline{L_1 \cap L_2} = \bar{L}_1 \cup \bar{L}_2$ (NAND), $L \downarrow L_2 = \bar{L}_1 \cap \bar{L}_2$ (NOR) $\overline{(L_1 \cup L_2)}$

Priority: $() > \bar{} > \cap > \cup > \rightarrow > \leftrightarrow$

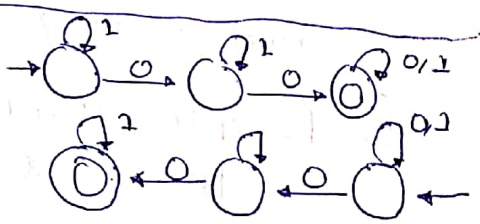
(32) **Reversal of Language:**

(a) Make every final state as initial state

(b) Make every initial state as final state

(c) Reverse all the arrows.

ex:



(33) **Homomorphism and Inverse Homomorphism**

(a) **Homomorphism**

ex: $L = \{aa, ab, b\}$ $h(a) = 001$, $h(b) = 10$ {Closed under regular & CFL}

$L = \{001001, 00110, 10\}$ $\boxed{h(\xi) = \Gamma}$ {It apply on each alphabet (atomic) on string}

(b) **Inverse Homomorphism:** It will cover every possibility, which may result in L.

ex: $h(a) = 1$, $h(b) = 0$, $h(c) = 10$; $L = \{1010, 010\}$

$L = \{\underbrace{abab}_{1010}, abc, cab, cc, \underbrace{bab}_{010}, bc\}$

(c) $|h| \leq |L|$, because h can be ϵ

(d) $|h^{-1}|$ is not comparable to $|L|$, because h^{-1} can be ϕ.

ex: $h(1) = \epsilon$, $h(0) = \epsilon$; $L = \{10, 01\}$; $\{h(a) = 00, h(b) = 100\}$

$h(L) = \{\epsilon\}$

$h^{-1}(L) = \phi$

INIT and QUOTITENT

INIT → Prifix

QUOTITENT → / (Right) , \ (Left)

$\epsilon \in INIT(L)$

$L \subseteq INIT(L)$

$INIT(INIT(L)) = INIT(L)$

$L/\Sigma^* =$ Prefix (L) of INIT(L)

$L\backslash\Sigma^* =$ Suffix(L)

Ex:

L	INIT(L)
$a^* = \{\epsilon, a, aa, aaa \ldots\}$	$\{\epsilon, a, aa, aaa \ldots\} = a^*$
$a^*b^* = \{\epsilon, a, b, aa, bb, aab\}$	$\{\epsilon, a, b, aab, abb \ldots\} = a^*b^*$
$(ab)^* = \{\epsilon, ab, abab, \ldots\}$	$\{\epsilon, a, ab, aba \ldots\} = (ab)^* + \overline{(ab)^*a}$
$(a^n b^n)_{n\geq 0} = \{\epsilon, ab, aabb, \ldots\}$	$\{\epsilon, a, ab, aab, aabb\} = \{a^m b^n \mid m \geq n\}$
$\{w \mid n_a(w) == n_b(w)\}$ $\{\epsilon, ab, ba, aabb, bbaa \ldots\}$	$\{\epsilon, a, b, ab, ba, \ldots\} = \{(a+b)^*\}$

$L_1 = \{100, 01, 00\}$; $L_2 = \{00, 0\}$

$L_1/L_2 =$ 100/00 , 01/00 , 00/00

1 , ϕ , ϵ

$L_1/L_2 =$ 100/0 , 01/0 , 00/0

10 , ϕ , 0

Remove matching Denominator pattern from right side of numerator.

$L = \{\epsilon, 0, 1, 10\}$

35) One way theorems for Regular Language { May or May not be }

(i) Regular ∪ Non Regular = Can be Regular

$\{\Sigma^*\} \cup \{a^n b^n\} = \{\Sigma^*\}$

(ii) Non Regular ∪ Non regular = Can be regular

$\{a^n b^n\} \cup \{\overline{a^n b^n}\} = \{\Sigma^*\}$

(20) (iii) Non regular ∩ Regular = can be regular.

$\{a^n b^n\} \cap \{\phi\} = \{\phi\}$

(iv) Non regular ∩ Non regular = can be regular

$\{a^n b^n\} \cap \{a^n b^n\} = \{\phi\}$

(v) Non regular · Regular = can be regular.

$\{a^n b^n\} \cdot \{\phi\} = \{\phi\}$

(vi) Non regular · Non regular = Non regular (always)

(vii) Finite ∪ NR = Non regular.

$\{ab\} \cup \{a^n b^n\} = \{a^n b^n\}$

(Regular)

(viii) (Non regular)* = can be regular

$\{a^{n^2} \mid n \geq 0\} = \{a^*\}$

$\{a^i \mid i \text{ is prime}\} = \{a^* - a\}$

$\{a^2, a^3, a^5, a^7, \ldots\} = \{\in, a^2, a^3, a^4, a^5, a^6, a^7, a^8, \ldots\}$

($a^4 = a^2 \cdot a^2$; $a^6 = a^3 \cdot a^3$; $a^8 = a^2 \cdot a^2 \cdot a^2 \cdot a^2$)

If L^* is non regular then surely L is non regular.

(36) Variation of FA

(a) FA < FA + 1 STACK < FA + n stack
(PDA) ($n \geq 2$) (TM)

(b) Counting Automata : • Only one symbol used
• Used for counting comparison

FA < FA + counter < FA + 1 stack < FA + 2 counter ≡ FA + 2 Stack
(PDA) TM TM

(c) FA + 1 Queue ≡ TM
n^{th}

(d) FA + R/W + L/R ≡ TM

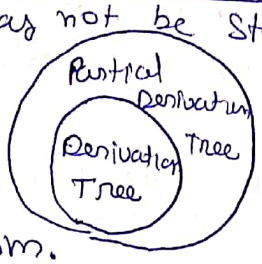
(e) FA + R/W ≡ 1 way TM

(f) FA + L/R ≡ FA ≡ 2 way FA

Partial Derivation Tree

(a) Root may or may not be Start symbol.

(b) Root may or may not in Sentential form.



(38) Inherently Ambiguous Language

* A language is ambiguous if every grammar corresponding to that language is ambiguous. Such language is known as inherently ambiguous language.

(39) Condition for Language to be Ambiguous

(i) Two conditions are in union: $A \cup B$

(ii) Two conditions must not overlap. i.e $A \not\subseteq B$ or $B \not\subseteq A$

ex: $\{a^n b^n \mid n \geq 0\} \cup \{a^{2n} b^{2n} \mid n \geq 0\} = \{a^n b^n \mid n \geq 0\}$ (overlap)

(iii) Two condition must have something common. $A \cap B \neq \emptyset$

(40)

(i) For Every DCFL there exists LR(K) Grammar.

(ii) For Every DCFL LL(K) may or may not exists.

(iii) LR(K) are Fast and unambiguous and also LL(K).

(iv) LL(K) is used for TOP Down compiler & LR(K) is used for Bottom up compiler.

(41) Algorithms in CFG

(i) Removal of NULL Production.

(a) If "NULL" belongs to Language, then Grammar must have ϵ production.

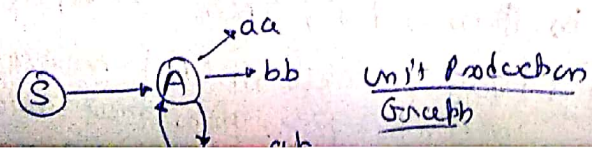
(b) If "ϵ" Present in Grammar then language may or may not have NULL.

(c) Algo: * Identify all NULL variable.
* Write all production without including NULL.
* Write all production generate by putting NULL in variable along with previous one.

(ii) Removal of Unit Production

* Identify all unit production using unit production dependency graph.
* Replace all unit production with there production:

ex: $S \to aC \mid A \mid aA$
$A \to aa \mid bb \mid B$
$B \to ab \mid ba \mid A$



(22) (iii) Removal of Useless Productions

* Create the set of variable set which can directly go to terminal.
* Add other variable in the set which can go to terminal. with help of previous set.
* Remove all variable which is not present in set.
* Draw dependency graph.
* Remove disconnected production.
* Final production is our required grammar.

---

(42) Conversion of Context Free Grammar into Chomsky Normal Formal

Algo

(i) First CFG converted to ε-Free CFG & Unit Production.

(ii) Chomsky Normal Form have production of following form:

$$\begin{bmatrix} S \to VV \\ V \to T \end{bmatrix}$$

(iii) Find out all the productions which are already in CNF.

(iv) Convert remaining production into CNF by introducing new variable.

---

(43) Properties of CNF & GNF

(i) If CFG is ε-Free & Unit Production Free, then $\exists \equiv G'$ in CNF & GNF.

(ii) CNF is used for CYK algorithm.

(iii) Greibach Normal Form is of following type:

$$\begin{bmatrix} V \to TV^* \\ V \to T \end{bmatrix}$$

(iv) GNF is used to convert grammar to PDA.

(v) If G is in CNF, then string of length n have derivation steps: $2n-1$ where $(n-1)$ is for final state & n is for terminal replacement.

ex: $S \to AB$, $A \to a$, $B \to b$

Let, $n=2$, ab

$S \to AB$ ---- (1) Step (Final State)
$S \to aB$ ---- (2) Step (Terminal)
$S \to ab$ ---- (3) Step and Final Step

(vi) CNF gives binary tree; GNF may or may not give BT.

(vii) If G is in GNF, then string of length n have derivation step: n

(viii) **Rank**: Every variable in CNF have unique length, Rank of non terminal is longest path from that variable to other. (Directed Graph)

* Rank of loop is ∞. * If S is finite ⇒ language is finite. * If S is ∞ then anything

PDA accept string using two method :

(a) Final state Method : In it, it is not necessary that stack is empty.

(b) Empty Stack Method : In it stack must be empty, so string is accepted.

*Both have same power

---

(15) CSL is Useful in :

define use

(i) Variable declare before use, $\{ww \mid w \in (0,1)^*\}$

(ii) Matching formal and actual parameter. $\{a^n b^m \quad c^n d^m\}$

ex: f(real, real, int, int, int) → f(π^2 i^3, π^2 i^3) (at time dec) Formal actual (at time calling)

(16) Turing Machine :

(i) Format : $(Q, \Sigma, \Gamma, \delta, q_0, B, F)$ → Tape alphabate

(ii) If a string accepted by turing machine, then it is permanently accepted.

(iii) Input alphabate (Σ) never contain (B) blank symbol.

$$\Sigma \subseteq \Gamma - "B"$$

(iv) $\delta_{Q \times \Gamma} = \delta_{Q \times \Gamma \times \{L,R\}}$

(v) Both deterministic & Non deterministic TM have same power.

(vi) Configuration of TM contain is also known as Instantaneous Description.

ex: If ID is Bq_0aabbB and $\delta(q_0, a) = \delta(q_1, b, R)$

$\delta(q_1, B) = \delta(q_0, a, L)$

State after 2 step:

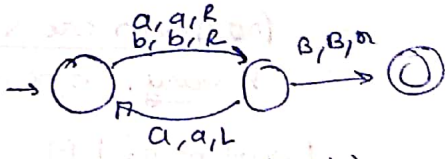
$\Rightarrow$ Bq_0aabbB → Bbq_1abbB → Bq_0babbB

Bq_0babb → Bbq_1abbB → Bq_0babbB

(v) Final state is permanent state, in input after final state is accepted.

ex: q_0 —a,a,R→ (O) } It will accept $a(a+b)^*$ (any)

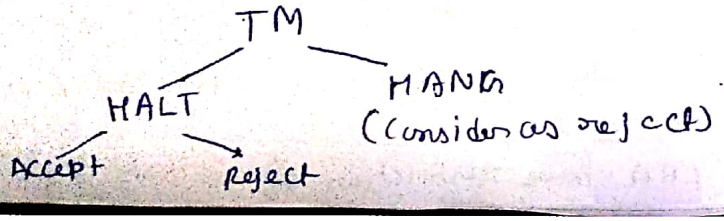
ex: q_0 —a,a,R→ q_1 —B,B,R→ (O) } It will accept only "a".



HALT : (a+b)

HANG : $(a+b)a(a+b)^*$

(vi)



(24)

(47) Every Programming language is CSL. but not CFL.

(48) Church Turing Thesis : Every logically computable function is represented by Turing machine.

(49) λ-Calculus Model

(a) It has set of functions which cover all logical property and represented by TM.

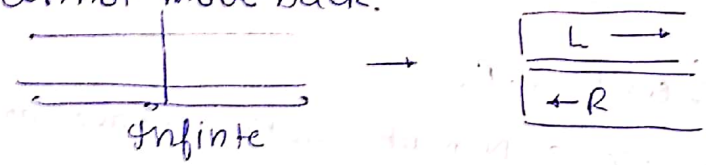
(b) λ-Calculus has same power as Turing Machine.

(50) (a) Every logically recognizable function is turing recognizable.
(b) Every logically computable function is turing computable.

(51) Types of Turing Machine

(i) Turing Machine with Stay option : $\delta_{Q \times \Gamma} \rightarrow Q \times \Gamma \times (L, R, S)$ (S: Stay)

(ii) Turing Machine with semi infinite Tape : After reaching particular point we cannot move back.



(iii) Offline TM : * Input & Output are kept in two different tape.
* Input have restriction but output don't have.

(iv) Multitape TM : It has multiple tape but same as normal TM.
It is best, if we have K-TM, then $O(n)$. other $O(n^2)$

(v) Multiple Dimensional TM : It can move [Right, Left, Up, Down]

(vi) Non deterministic TM

(vii) Universal TM : (a) It is 3 Tape Machine, each have R/W head
(b) It can simulate other TM.

TM Program → It store Program

Co-Tape → Store Input/output

ID Tape → Configuration / Current

Program are stored in form of binary. Encoding require. $(K, \Sigma, \Gamma, \delta, S, H)$

Encoding of TM :

Encod: K, Σ, δ

(52) LBA < REC < RE (LBA have restriction on tape at both end {$ - - $})

① NFA minimization is decidable because its algorithm exists.

② Finiteness of regular, dcfl and CFL is decidable with help of Pumping Lemma.

$\omega \in L(G)$, $n \leq |\omega| \leq (2n-1)$

where, n is number of state

③ In case of regular grammar Ambguity is decidable

④ Checking the ambiguity of CFGs is undecidable.

⑤ Checking whether a grammar is LL(k) or LR(k) is decidable.

⑥ Membership of all language is decidable except Recursive Enumerable (RE).

$\forall \omega \in \Sigma^*$, if $\omega \in L \rightarrow$ YES

$\omega \notin L \rightarrow$ NO

ⓐ Brute Force Algorithm = $O(k^n)$ } NP Problem

ⓑ CYK Algorithm = $O(n^3)$ } P-problem

ⓒ LL(k) & LR(k) = $O(n)$

⑦ Encoding a Turing Machine: ⓐ It is useful in deciding whether turing machine halt or not, which itself is useful to decide Decidability and Undecidability.

ⓑ It means, converting a TM into a binary strings.

ⓒ Tuple of TM: $(Q, \Sigma, \Gamma, \delta, q_0, F)$

$\Sigma \subset \Gamma$, $F \subseteq Q$, $q_0 \in Q$

Need TO Encode:

K : we need to encode state

Γ : we need to encode Take alphabate

δ : we need to encode Transition Function

(d) For any machine number of state is Finite

K → give any arbitrary alphabate to state in our case we give it q.
and for each state represent it using binary bit.
(q_{00}) → inital state and rest as 01, 10, 11, ...

accept state(y) → (y_{10}) state number and y for yes if that state accepting state

Rejecting State(n) → (n_{10}) state number and n for no. if that state is rejecting state.

Γ (take alphabate) → gives any arbitrary alphabate. we consider it as "a".

$\Gamma = (a, b, c, d, \square)$

a_{000}, a_{001}, a_{010}, a_{011}, a_{100}

δ (State Transition) → (state, symbol, state, symbol, move)

→ : Right
← : Left

ex: $\delta(q_{00}, a_{001}, y_{10}, a_{010}, \overrightarrow{\text{or } R})$

ex: $q_0 \xrightarrow{a,b,R} q_1 \xrightarrow{b,c,L} q_2$

encode: $(q_{00}\ a_{000}\ q_{01}\ a_{001}\ R)(q_{01}\ a_{001}\ q_{10}\ a_{010}\ L)$

Complexity Classes :

(a) It contain a set of problems that take similar range of space and time to solve a problem.

(b) Problems are proven to be in a particular complexity class by running the problem on an abstract computational Model. i.e a Turing Machine.

(c) Complexity class helps to describe how many steps it would take a Turing Machine to decide of problem.

(d) Some Complexity classes are subset of other.

Time Complexity:

(a) Numbers of steps it needs to take to solve a problem.

(b) It is also used to describe, how long it takes to verify a problem.

(c) It is also describe as, how many steps a Turing Machine takes when solving a problem.

(vi) Space Complexity:

(a) How much space algorithms needs to solve a problem.

(b) In terms of turing machine, how much take require to solve a problem.

(vii) P-Class (Polynomial)

(a) It contain decision problem. with "yes" or "no" answer. that are solvable in polynomial time by Deterministic Turing Machine (DTM).

(b) These a class of problems are decidable in Polynomial Time by DTM.

(c) Polynomial Time: (n^k), where n is input size.

(viii) NP-Class (Non Polynomial)

(a) It contain decision problem with "yes" or "no" answer that are solvable in non-polynomial time with help of "Non Deterministic Turing Machine (NDTM)."

(b) This include all problems from Polynomial time to Exponential time.

(c) Problems in NP can be verified by Turing Machine in polynomial time. This means if given a "yes" answer to a NP problem, we can check that wethere it is true or not in polynomial time.

Ex: Factoring a large NO: It takes much time to find factor but, if factor is given, it takes less time to verfy it. belongs to factor or not

(d) Proving Problem in NP involves, providing witness string that can be verified in Polynomial time.

$$P \subset NP \quad , \quad P \neq NP$$

(e) Complement NP or CoNP Problem ($\overline{NP}$) :

* It contains problems that have a polynomial time verification for "no".
* If given a solution that does not solve the problem it is easy to verify if that solution does not work.

ex: Checking a Number is PRIME.

(f) NP Problems are also turing decidable.

(g) ex: Graph Isomorphism, TSP, Graph Coloring, Factoring.

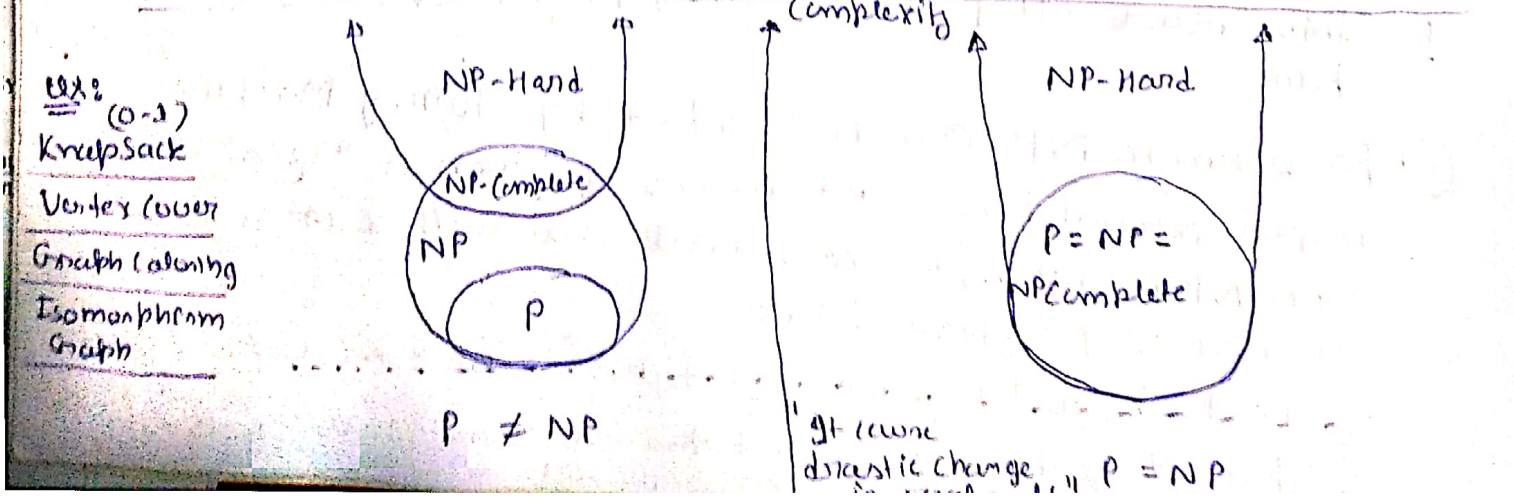
(h) NP-Complete Problem (NP-C)

(a) A problem is NP-C if it is both NP and NP-hard.

(b) NP-Hard

* A problem is at least as hard as the hardest problem in NP, then it is NP hard.
* A Problem A is NP-hard if for every Problem L in NP, there is a polynomial time reduction from L to A. ($\leq_P$)
* NP-hard Problem, need not be in NP and need not be Decidable.

(c) NP-Complete problems are special because any problem in NP class can be transformed or reduce into NP-Complete.



: @ If there exists a halting Turing machine (HTM) that accepts and halt for every $\underbrace{w}_{\text{word or string}} \in \Sigma^*$ (every thing).

REC is also known as Recursive and Turning decidable or Decidable Language.'

RE : @ A language is RE or Recursive Enumerable, if ∃ a turing machine which accept and Halt for some $w \in L$. not every or all.

ⓑ RE is also known as Turing Recognizable or Turing acceptable or semidecidable.

Decidability : Problem for which HTM exsists.

Semi-decidability : Problem for which TM exsists and allow to go in Hang state.

Undecidable : Problem for which no TM exsits.

* For a language if finite then decidable the infinite ones also decid.
* If a problem is decidable then its complement is also decidable.
* Every Undecidable problem is NOT REC but may or may not be RE.
* x NOT RE is not even semidecidable.

Enumeration Procedure

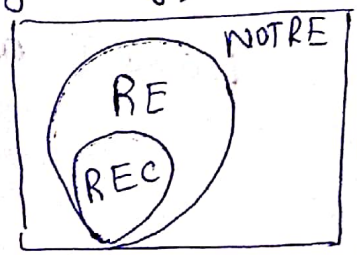
ⓐ It Generate all possible string that belongs to Language. in lexiographical order.

ⓑ RE → T-Enumeration Procedure (Some time it may hang) (TEP)

ⓒ REC → Turing Enumeration Procedure (TEP)

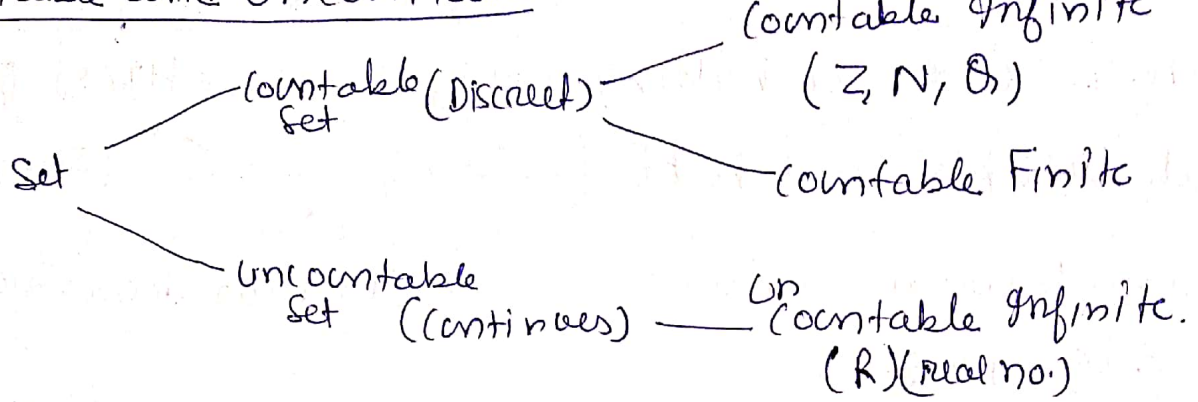
ⓓ If L is REC, then $\bar{L}$ is also REC

ⓔ If L is RE, then $\bar{L}$ may or may not RE.
If $\bar{L}$ is RE, then L is REC
$\bar{L}$ is NOT RE, then L is NOT RE



ENUM
TEP (RE) — FP (NOT RE)

Countable and Uncountable



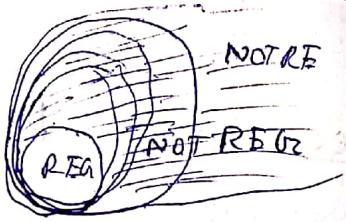
Properties:

(a) Σ^* is countable, because of it every language is countable.

(b) L_{reg}, L_{CFL}, L_{CSL}, L_{REC}, L_{RE} is countable.

(c) NOT RE is uncountable Infinite.

(d) NOT REG, NOT CFL, NOT CSL, NOT REC, NOT RE all are uncountable Infinite because all of there include NOT RE.



(e) Cantor Theorem: If S is CI, then 2^S is UCI.
So, Σ^* is CI then 2^{Σ^*} UCI.

Membership: L is REC iff L has membership Algorithm.

Lexicographical: REC alway generate Lex order.

L, $\bar{L}$ Theorem:

(i) If L is RE but not REC, then $\bar{L}$ is not RE.

(ii) If L is not RE, then $\bar{L}$ may or may not RE.

(iii) If L is RE, then $\bar{L}$ can be REC or NOT RE.

(iv) If L is REC, then $\bar{L}$ is REC.

(v) If L is RE & $\bar{L}$ is RE, then L must REC.

(vi) If L is NOT DCF, then $\bar{L}$ surely NOT DCFL.

L and $\bar{L}$ always fall in same set. except: RE.

Algorithm ($\leq_p$)

- $P_1 \leq_p P_2$, if P_1 is reducible to P_2 in polynomial time.
- If P_1 is undecidable than P_2 is also undecidable.
- If P_1 is decidable than P_2 may or may not decidable
- If P_2 is decidable than P_1 is decidable.
- If P_2 is undecidable then P_1 may or may not be undecidd.

List of Decidable and Undecidable Problems

1. Machine that accepts its own code (encoded): RE but NOT REC (Semidecidable)
2. Machine which can list the machine who rejects there own code : NOT RE (Undecidable)
3. Checking ambiguity of CFG : Undecidable
4. Checking ambiguity of RG : Decidable (Time approx Bounded)
5. Halting Problem : Undecidable

Let a machine H, we take string as input and process it, and gives output "Stuck" if its Hang, otherwise give output not stuck. if its not Hang.

Let, a machine U, which take machine H as input along with it, it also take string as input, then it runs it on itself. If machine H produce output "stuck" then U gives output "Not stuck" and if it produce output "Not stuck" then, U give output "stuck".

Now, we convert U into binary string using encoding and input it to U itself along with string input.

So, when input U says "Not Stuck" we get output "STUCK" and when input U says "Stuck" we get output "NOT STUCK". which is

(6) Blank Tape Halting Problem : Undecidable

* We can reduce Blank Tape Halting problem to Halting Problem and Halting Problem is undecidable.

(7) State Entry Problem : Undecidable

* We can reduce it to Halting Problem, to check whether it reach particular state or not. But we know halting Problem is undecidable.

(8) Post Correspondance Problem : Undecidable

$A = \{\underset{1}{0}, \underset{2}{10}, \underset{3}{11}, \underset{4}{1110}\}$ $B = \{\underset{1}{10}, \underset{2}{11}, \underset{3}{0}, \underset{4}{111}\}$

Pick string from same position in both set and try to make them equal.

(9) Modify Post Correspondance Problem : Undecidable

* In it we fix that 1st position from both set are included.

(10) Printing Specific Character : Undecidable

* We can reduce this problem to Halting Problem or State Entry Problem.

(11) Printing (Arbitrary or Any) Character : Decidable

(12) Finite Step Halting Problem : Decidable

* Run encoded machine along with input for k steps on Universal Turing machine.

L(G)	Membership $w \in L^*$	Emptiness $L = \phi$	Finiteness L=Finite	Equivalence $L_1 = L_2$	Regularity L is regular	Completeness $L = \Sigma^*$	Disjoint $L_1 \cap L_2 = \phi$	Ambiguity	Subset $L_1 \subseteq L_2$
REG	D	D	D	D	D	D	D	D	D
DCFL	D	D	D	D	D	D	UD	UD	UD
CFL	D	D	D	UD	UD	UD	UD	UD	UD
CSL	D	UD	UD	UD	UD	UD	UD	UD	UD
REC	D	UD	UD	UD	UD	UD	UD	UD	UD
RE	UD	UD	UD	UD	UD	UD	UD	UD	UD