Definition

① Regular language

A language that can be defined by a regular exp. is called regular language.

eg: $L_1 = \{a, ab, abb, \dots\}$

$\Rightarrow L_1 = L(ab^*)$

② Regular Expression

~~Any language that can be defined by~~

All the symbols of alphabet set Σ can be treated as r.e.

③ If L_1 and L_2 are regular languages, then L_1+L_2, $L_1.L_2$ and L_1^* are also regular languages.

→ L_1+L_2 means lg of all words in either L_1 or L_2.

→ $L_1.L_2$ means lg of all words formed by concatenating a word from L_1 with a word from L_2.

→ L_1^* means strings that are concatenation of arbitrarily many factors from L_1.

Proof: L_1 and L_2 are regular lg, there must be TGs that accept them.

Let TG_1 accept L_1 and TG_2 accept L_2.

Let us further assume that TG_1 and TG_2 each have unique start and unique seperate final state.

① For L_1+L_2

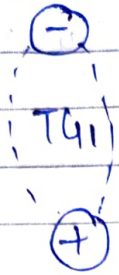
L_1 & L_2

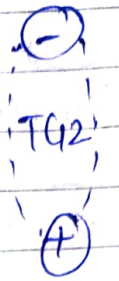
↓ RE ↓ RE

TG_1 TG_2 —Map→ TG_3 —By kleene theorm→ R_3 → RL (L_1+L_2)

$L_1 \rightarrow TG_1$ accepts all words in L_1

$L_2 \rightarrow TG_2$ accepts all words in L_2



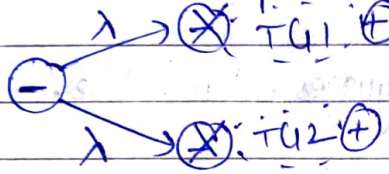


$TG = TG_1 + TG_2$

$\downarrow$ i.e.

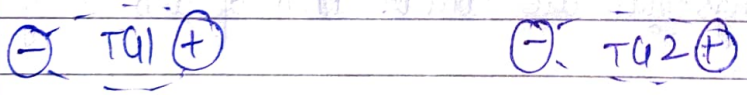
$L_1 + L_2$

→ TG which accepts all words in L_1 or L_2



→ for $L_1 . L_2$

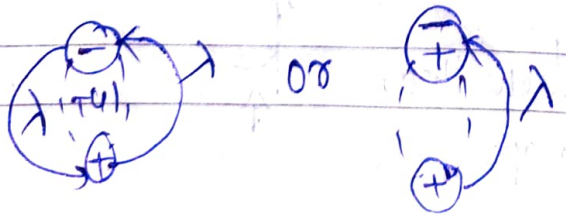
TG_1 TG_2



$TG_1 . TG_2 \backsimeq L_1 . L_2$

→ for $(L_1)^*$

$TG \backsimeq TG_1^* \longleftrightarrow L_1^*$



④ If L is a lg over alphabet Σ, we define it's complement L', to be the lg of all strings of letters from Σ that are not words in L.

→ set of reg. lg is closed under complementation

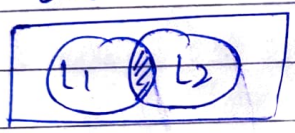
⑤ Intersection

By De-Morgan law of sets

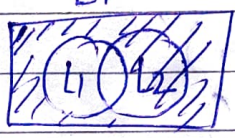
$$L_1 \cap L_2 = (L_1' \cup L_2')'$$

Illustration through venn diagram

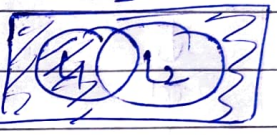
LHS



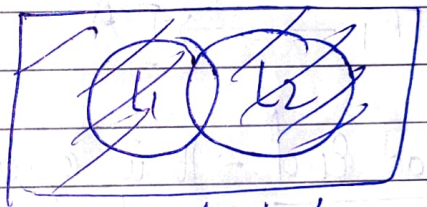
$=$



$+$

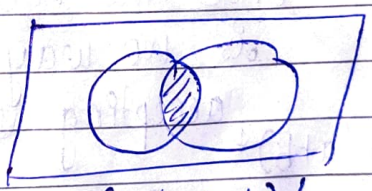


$=$



$L_1' + L_2'$

$=$



$(L_1' + L_2')'$

$L_1 \cap L_2$

⑤ Complement

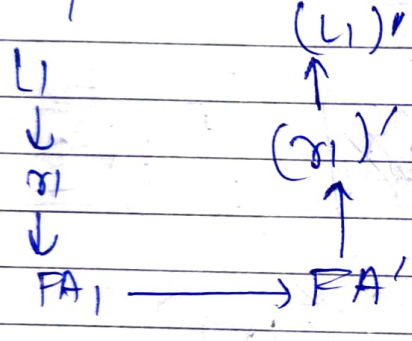
If L_0 is reg. lg, by Kleene's theorem that there is some FA that accepts the lg L. Some of states of this FA are final states and some are not.

Let us reverse the final state of each state, i.e.,

if it was a (+), make it (-) and if it was (-), make it (+).
If input string formerly ended in a non-final state, it now ends in final state and vice-versa.

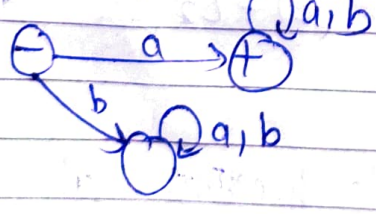
This new m/c we have built accepts all input strings that were not accepted by original FA (all words in L') and rejects all input strings that FA used to accept (words in L)
∴, this m/c accepts exactly lg L'.

So by kleene theorem L' is regular.



* Dead End → It is a rejecting state i.e. essentially a dead end.
Once the m/c enters a dead state, there is no way for it to reach an accepting state.

eg. a(a+b)*



Page No.
Date

* Properties of CFL

① If L_1 is CFL and L_2 is CFL, then $L_1 + L_2$ is also CFL

We will prove above theorem by constructive algorithm which means that we shall show to create a grammar $L_1 + L_2$ out of grammar for L_1 and L_2.

Since, L_1 is CFL → there exist CFG1 for it
L_2 is CFL → " " CFG2 for it

→ let us asume a CFG for CFL, L_1 (words starting with a)
CFG, $S \to aX$
$X \to aX | bX | \lambda$

we will rename non-terminal by adding subscript,

$$\left.\begin{aligned} S_1 &\to aX_1 \\ X_1 &\to aX_1 | bX_1 | \lambda \end{aligned}\right] \text{—} (1)$$

→ let us assume a CFG for CFL, L_2 (word starting with b) which has start symbols S_1

$S \to bX$
$X \to aX | bX | \lambda$

we will rename non-terminal by adding subscript,

$$\left.\begin{aligned} S_2 &\to bX_2 \\ X_2 &\to aX_2 | bX_2 | \lambda \end{aligned}\right] \text{—} (2)$$

→ Build a new CFG with new start symbols S and add a new production of form

$$\left.\begin{aligned} S &\to S_1 | S_2 \\ S_1 &\to aX_1 \\ X_1 &\to aX_1 | bX_1 | \lambda \\ S_2 &\to bX_2 \\ X_2 &\to aX_2 | bX_2 | \lambda \end{aligned}\right] L_1 + L_2$$

② Concatenation

Rules ① let $L_1 = a(a+b)^*$

CFG1
$S \to aX$
$X \to aX \mid bX \mid \lambda$

$L_2 = b(a+b)^*$

CFG2
$S \to bX$
$X \to aX \mid bX \mid \lambda$

Rule 1 Rename non-terminal of CFG1 & CFG2

CFG1
$S_1 \to aX_1$
$X_1 \to aX_1 \mid bX_1 \mid \lambda$

CFG2
$S_2 \to bX_2$
$X_2 \to aX_2 \mid bX_2 \mid \lambda$

Rule ② α law, CFG3 $\to L_1 . L_2$

Rule 2. add new rule with start symbol
$S \to S_1 . S_2$

New, CFG3

$S \to S_1 . S_2$
S_1
X_1
S_2
X_2

} $L_1 . L_2$

③ Kleene closure

let $L_1 = a(a+b)^*$

CFG1
$S \to aX$
$X \to aX \mid bX \mid \lambda$

① Rename N.T of CFG1
$S_1 \to aX_1$
$X_1 \to aX_1 \mid bX_1 \mid \lambda$

Page No.
Date

② Add a new rule with start symbol S

s.t. $S \to S_1 S \mid \lambda$

also, new CFG for $(L_1)^*$

$S \to SS_1 \mid \lambda$

S_1

X_1

④ Intersection

Case 1: $L_1 \cap L_2$ is a CFL

If L_1 & L_2 are 2 CFLs and L_2 is contained in L_1, then intersection of 2 lgs is L_2 is again a CFL

for eg:

L_1 = Palindrom excluding λ L_2 = string of all a's excluding λ

$S_1 \to aS_1a \mid bS_1b \mid a \mid b \mid aa \mid bb$ $S_2 \to aS_2 \mid a$

L_2 is contained in L_1

$L_1 \cap L_2 = L_2$ which is again a CFL

Case 2: may not be a CFL

$L_1 = \{a^n b^n c^i ; n, i \geq 0\}$ $L_2 = \{a^i b^n c^n ; n, i \geq 0\}$

$S_1 \to X_1 Y_1$
$X_1 \to aX_1b \mid ab$
$Y_1 \to cY_1 \mid c$

$S_2 \to X_2 Y_2$
$X_2 \to aX_2 \mid a$
$Y_2 \to bY_2c \mid bc$

$L_1 \cap L_2 = \{a^n b^n c^n\} \notin$ CFL

(5) CFL closed under complement

Let us assume CFL_1 & CFL_2 are closed under complementation — (1)

If L_1 & L_2 are CFL

L_1', L_2' — CFL (by (1))

$L_1' + L_2'$ — CFL (union)

$(L_1' + L_2')'$ — CFL (1)

$= L_1 \cap L_2$ (CFL not closed under intersection)

Thus, assumption 1 failed.

∴ CFL are not closed under complement.

* Turing Machine

A TM is a quintuple $(K, \Sigma, \delta, s, H)$ where

(i) K is a finite set of states,
(ii) Σ is an alphabet, containing $\sqcup$ and $\triangleright$ but not containing symbols $\leftarrow$ and $\rightarrow$,
(iii) $s \in K$ is initial state,
(iv) $H \subseteq K$ is set of halting states,
(v) δ is the transition function,

$$\delta(\text{current state}, \text{symbol read}) = (\text{next state}, \text{Action})$$

there is no algorithm that correctly determines whether arbitrary program eventually halt when run

Page No.
Date

* Halting Problem → It is a type of problem in which you don't know when TM will stop or not.

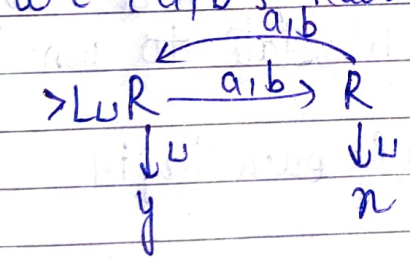
It is a unsolvable problem, which means there does not exist any such algorithm.

* Recursive language

Language L is called recursive if there is a TM that decides L.

Let $M = (k, \Sigma, \delta, s, H)$ be a TM
Let $\Sigma_0 \subseteq \Sigma = \{\sqcup, \triangleright\}$ be a alphabet called the I/p alphabet of M,
We say that M decides a lg. $L \subseteq \Sigma_0^*$, if for any string $w \in \Sigma_0^*$, the following is true :
① If $w \in L$, then M accepts w
② If $w \notin L$, then M rejects w

eg. $w \in \{a, b\}$ having even length



* Recursively enumerable lg

A lg L is RE if and only if there is TM 'M' that semidecides L

Let $M = \{k, s, \delta, H, \Sigma\}$ be a TM,
Let $\Sigma_0 \subseteq \Sigma - \{\sqcup, \triangleright\}$ be an alphabet,

Let, $L \subseteq \Sigma_0^*$ be a lg.

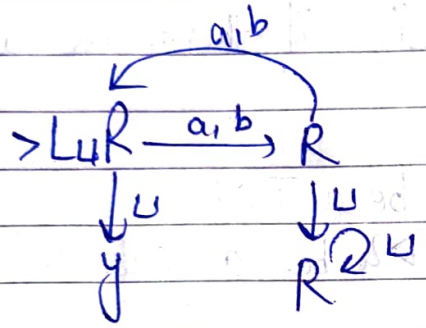
We say that,

M semidecides L, if for any string $w \in \Sigma_0^*$ the following is true:

$w \in L$ if only if M halts on input w.

$w \notin L$, M fails to halt and goes forever to blanks.

eg:

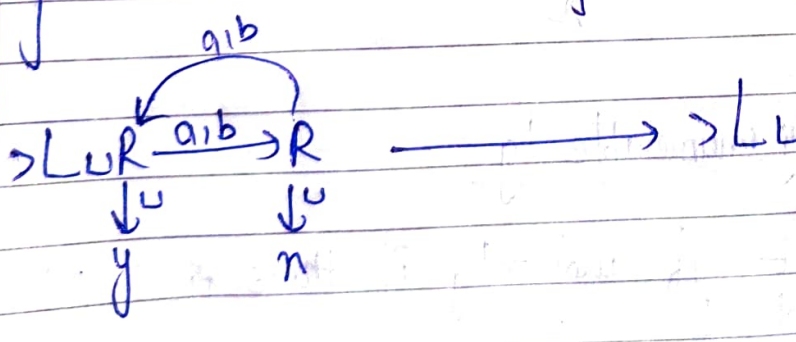


* If a lg is recursive, then it is recursively enumerable.

Every recursive L is also recursively enumerable

→ it means if TM decides L, then it also semidecides
for this, make all 'n' state to non-halting state

eg. $w \in \{a,b\}$'s having even length



TM decides L

TM semidecides L

* If L is Recursive lg, then its Complement $\bar{L}$ is also

Proof If L is decided by TM, $M = \{K, \Sigma, \delta, s, \{y, n\}\}$, then L is decided by TM, $M' = \{K, \Sigma, \delta', s, \{y, n\}\}$ which is identical to M except that it reverses the roles of two special halting states y and n.

ie., δ' is defined as follows:

$$\delta'(q,a) = \begin{cases} n & \text{if } (q,a) = y, \\ y & \text{if } (q,a) = n, \\ \delta(q,a) & \text{otherwise} \end{cases}$$

$M'(w) = y$ if and only if $M(w) = n$ and

∴ M' decides $\bar{L}$.

* Church Turing thesis

TM that halts on all inputs is a precise formal notion corresponding to the intuitive notion of an algorithm.

Nothing will be considered an algorithm if it cannot be rendered as TM that is guaranteed to halt on all inputs and all such m/c will be rightfully called algo.

Acc. to CTT, computational tasks that cannot be performed by TM are impossible, hopeless, undecidable (no algo. exist)

Page No.
Date

* Universal Turing M/C

UTC stimulates a TM, UTC can be considered as a subset of all TMs, it can match or surpass other TMs including itself

UTC is like sing TM that has a soln to all problems that is computable.
It contains a TM ~~operate~~ description as input along with an input string, run TM on input and returns a result

Programmable TM is called UTC