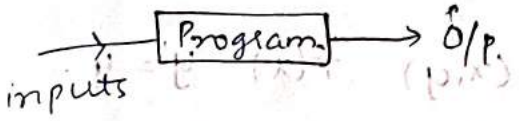
THEORY OF COMPUTATION

→ Theory of "programs".



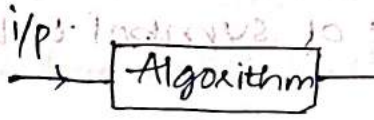
⇒ Theory of "algorithm"

→ Algorithm – Set of instructions that is written in a programming language.

1) Takes input
2) Carries out a set of finite operations on the i/p.
3) Return the o/p.

→ Computer : Reprogrammable machine.

A computer is not a machine.



$f : x \to y$

$x \in D \; ; \; y \in R$.

D: A set of integers R : {Yes, No}

Ex) $x \in I_+ \; (D)$ $y \in \{Yes, No\}$

$$y = f(x) = \begin{cases} Yes, & \text{if } x \text{ is prime} \\ No, & \text{if } x \text{ is not prime.} \end{cases}$$

We can't write algorithm for all functions.

Ex) Domain (D) Range (R)

1) Sentences in English. ⟶ Sentences in French.
   f: Translation

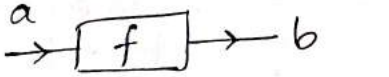
2) Image $\xrightarrow[\text{is car present in the image or not.}]{f: \text{Object detection}}$ {Yes, No}

D: A set of all programs. ⟶ R : {Yes, No}

f: Program will finally stop / continue indefinitely.

→ COMPUTABLE PROBLEM :

Given a function, will that admit an algorithm to solve it ?



$a \in D; f(a) = b; b \in R$

$G(f)$

Graph(f)

$G(f) = \{(a,b) \mid f(a) = b\}$

$(x,y) \quad f(x) = y' = y$.

Given a set S and an element z, check if $z \in S$ is a membership problem.

Membership question:

Given a set S and an element a, does $a \in S$?

Set of "finite string"

String: An ordered sequence of symbol taken from an alphabet.

(Σ^*) Alphabet: A finite set of symbols $\{a,b,c,d\}$
↳ Kleene closure $\{0,1\}$

$(0+1)^*$
$(a+b+c+d)^*$ Σ representation of alphabet.

Σ^* Set of all finite strings that can be composed from symbols $\{0,1\}$

$\Sigma = \{0,1\}$, $\Sigma^* \rightarrow \{\epsilon, 0, 1, 00, 01, 10, 101, \ldots\}$

Symbols $\rightarrow$ Alphabet $\rightarrow$ Strings

$\Sigma \qquad \Sigma^*$

A formal language L is a subset of Σ^*

$\Sigma^* = \Sigma^+ + \epsilon$ $\quad L = \{0,1,00,01,111010\}$.

↓ Positive closure $L_1 = \{x \in \{0,1\}^* \mid x \text{ has odd no. of } 1s\}$

$1001110101 \notin L_1$ (6 1's)

$0001101011 \in L_1$ (5 1's)

1) Finite State Machine (FSM)
Finite State Automata (FSA) → text processing compilers hardware design.

2) Push Down Automata (PDA) → AI applications
3) Turing Machine

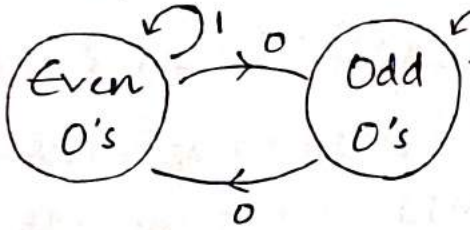
Regular grammars
Context free
Content sensitive
Unrestricted grammars.

Finite State Machines (FSM):

$\Sigma = \{0,1\}$, $L = \{x \in \{0,1\}^* \mid x \text{ contains even 0s}\}$

1 0 0 1 1 0 1 1 0

1) Start from the left end of the string
2) Scan through the symbols in the string one by one in a sequential manner.
3) All the symbols are processed, return the result



$L = \{x \in \{0,1\}^* \mid x \text{ has even 0's}\}$

1 0 1 1 0 1 0 0 1 $\in L$.

1) Set of states Q
2) Set of input alphabet Σ
3) Transition function δ.

$\delta : Q \times \Sigma \rightarrow Q$

4) Initial state q_0
5) Final states / Accepting states (F)

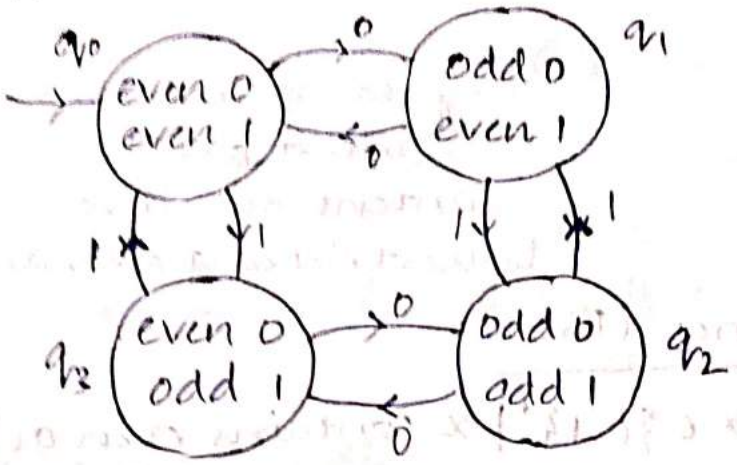
$Q = \{q_0, q_1\}$

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

δ :

Current state	Input symbol	Next state
q_0	0	q_1
q_0	1	q_0
q_1	0	q_0
q_1	1	q_1

$L = \{ x \in \{0,1\}^* \mid x \text{ has even 0's \& even 1's} \}$



even 0, even 1
even 0, odd 1
odd 0, even 1
odd 0, odd 1

$Q = \{q_0, q_1, q_2, q_3\}$

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

$\hat{\delta}, \delta^* \rightarrow$ extended transition function

$\delta(q, a)$

$\hat{\delta}(q, x)$

$\hat{\delta}(q, \epsilon) = q$ → empty string

$x \in \Sigma^*, \; b \in \Sigma$

$\hat{\delta}(q, x) = p$

$\hat{\delta}(q, xb) = \delta(\hat{\delta}(q, x), b)$

$$\boxed{\hat{\delta}(q, xb) = \delta(\hat{\delta}(q, x), b)}$$

M (Machine)

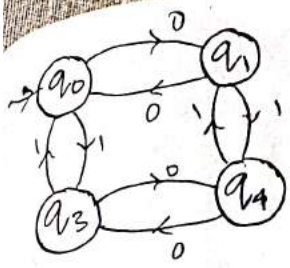
(language) $L(M)$ → set of all strings that will be accepted by M

$L(M) = \{ x \in (0,1)^* \mid \hat{\delta}(q_0, x) \in F \}$

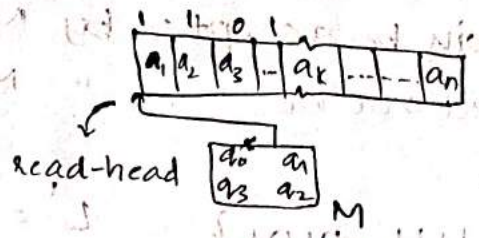
$F = \{q_0\}$

$\delta : Q \times \Sigma \rightarrow Q$

$F \subseteq Q$



Deterministic Finite Acceptors (DFA)



Read-head:
Job is to read symbol and change state accordingly.

Once an input symbol is read, it can't be read again.

$q_0\ q_3\ q_0\ q_1\ q_2$
$1\ 1\ 0\ 1$

$\hat{\delta}(q_0, 1101) = q_{0,4}$

Dealing with two sets:

1) Set of input symbols
2) Set of states machine is passing through.

M, $L(M) = \{ x \in \Sigma^* \mid \hat{\delta}(q_0, x) \in F \}$

1) All strings in L(M) are accepted by M
2) All strings not in L(M) are not accepted by M.

Then M is a machine for this particular lang. L.

Regular Languages:

A language for which a DFA can be constructed.

$L(M) = \{ x \in \Sigma^* \mid \hat{\delta}(q_0, x) \in F \}$

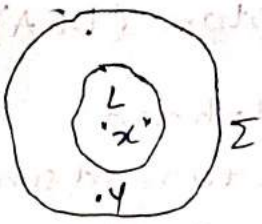
$L_1 = \{ x \in \Sigma^* \mid x \text{ has even 0's \& even 1's} \}$.

$L(M) \subseteq L_1$
$L_1 \subseteq L(M)$
$\Rightarrow L_1 = L(M)$

If machine has DFA $\Rightarrow$ has regular lang.

With every machine M there is a unique lang. L associated with it, and we say this language is accepted by M.

If there is a lang. it is not always true that a machine exists associated with that language.



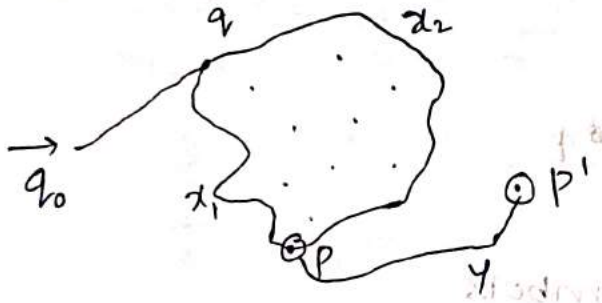
$L_1 = L(M)$

x will be accepted by M

y will not be " " M

$L' = \{00, 1010, 1111, 0101\}$ $\quad L' = L(M)$

$L' \neq L(M)$ (L' is not a language of M)



⊙ – final states.

$x_1, x_2 \in \Sigma^*$

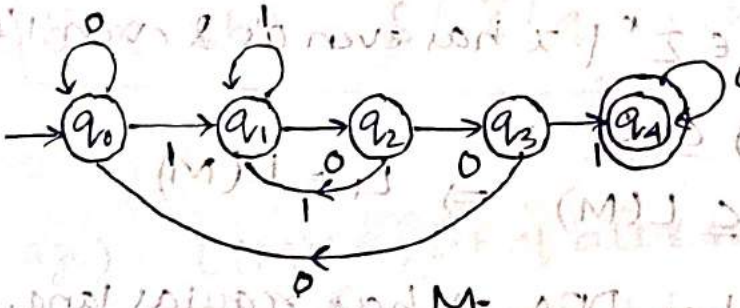
$\hat{\delta}(q, x_1) = \hat{\delta}(q, x_2) = p \in Q$

$y \in \Sigma^* \quad \hat{\delta}(q, x_1, y) = \hat{\delta}(q, x_2, y) = p' \in Q$

→ $L = \{x \in \Sigma^* \mid$ string 1001 is substring of $x\}$

$x = \text{----}1001\text{----}$

1. If 1001 is a substring of x then accept x
2. If 1001 is not a " " then don't accept x.



M

Properties to be checked: Complement
Union
Intersection

$\bar{L}$ → Complement of L.

$\bar{L} = \{x \in \Sigma^* \mid 1001$ is not a substring of $L\}$

L is a language

$\bar{L} = \{x \in \Sigma^* \mid x \notin L\}$

$\bar{L} = \Sigma^* - L$ (or) $\Sigma^* \setminus L$

→ M, L

1) x is accepted by M if $x \in L$
2) x is not accepted by M if $x \notin L$.

→ $\bar{L}$, M_1

1) x is accepted if $x \notin L$
2) x is not " if $x \in L$.

→ To make a complement:

1) Switch the final states to non-final states
2) Switch non-final states to final states.

→ L: $M = (Q, \Sigma, \delta, q_0, F)$

$\bar{L}$ $M_1 = (Q, \Sigma, \delta, q_0, Q - F)$

→ $L_1, L_2 \subseteq \Sigma^*$

L_1 and L_2 are regular, then

Is $L_1 \cup L_2$ regular?

$x \in \Sigma^*$; $L = L_1 \cup L_2$

$L_1 \rightarrow M_1$,
$L_2 \rightarrow M_2$

Given x,

1) Scan x and run M_1 on x and note the final state (p)
2) Scan x, run M_2 on x and note the final state (q)
3) Check if $p \in F_1$ or $q \in F_2$

$M_1 = (Q_1, \Sigma, \delta_1, q_{01}, F_1)$

$M_2 = (Q_2, \Sigma, \delta_2, q_{02}, F_2)$

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

Scanning can't be done twice, parallel running is also not possible.

$Q_1 = \{q_{01}, q_{11}, q_{21}\}$

$Q_2 = \{q_{02}, q_{12}\}$

$(q_{01}, q_{02}), a \longrightarrow (q_{11}, q_{02})$

q → States of a Machine M ← P

$Q = Q_1 \times Q_2$

$(p,q) \longrightarrow p \in Q_1, q \in Q_2$

$\delta((p,q),a) = (\delta_1(p,a), \delta_2(q,a))$

$\in Q_1 \quad \in Q_2$

$q_0 = (q_{01}, q_{02})$

$\forall x \quad \hat{\delta}((q_{01}, q_{02}), x) = (\hat{\delta}_1(q_{01}, x), \hat{\delta}_2(q_{02}, x))$

$= (p_1, p_2)$

$p_1 \in Q_1, \; p_2 \in Q_2$

$L = L_1 \cup L_2$

$F = F_1 \times F_2$

$p_1 \in F_1$ or $p_2 \in F_2$

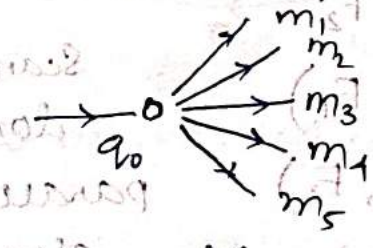
$\Rightarrow x \in L$

$Q_1 \times F_2$, $F_1 \times Q_2$ } F

$\rightarrow L = L_1 \cap L_2$

→ 1) All finite languages are regular.

$L = \{0.0, 10, 11, 1010, 1111\}$



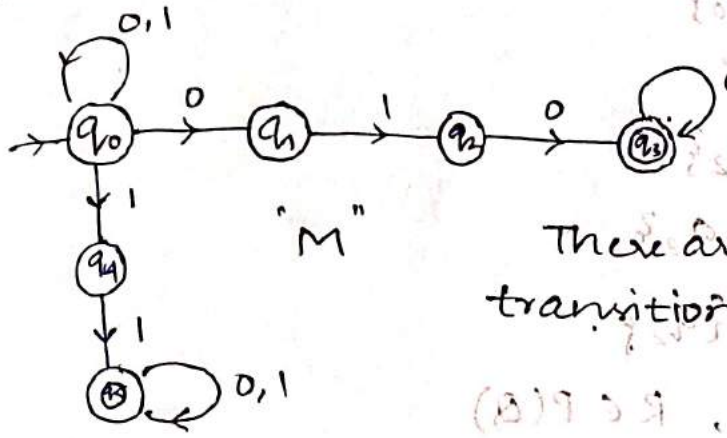
→ Suppose we have a state p and an alphabet a. On applying transition any of these can happen.

i) No transition occurs.

ii) Goes to a unique state q.

iii) Can go to multiple states q_1, q_2.

Here case-(2) is the DFA case.

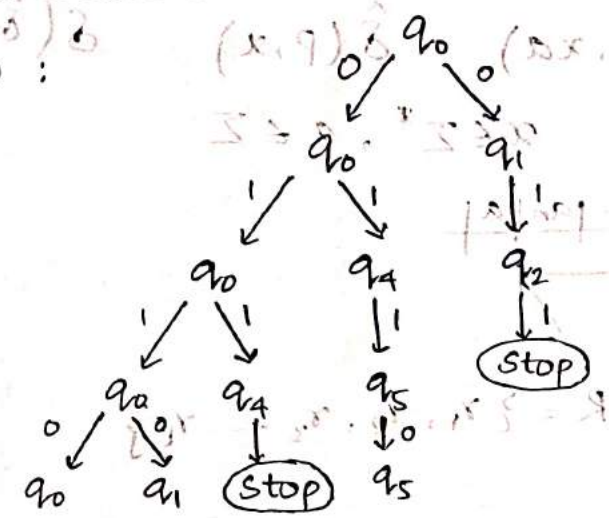


"M"

There are two possible transition states at q_0.

In such cases, we draw a graph to conclude the final status.

0110 :



$$L(M_{DFA}) = \{x \in \Sigma^* \mid \hat{\delta}(q_0, x) \in F\}$$

$$L(M_{NFA}) = \{x \in \Sigma^* \mid x \text{ can take } M \text{ to a final state}\}$$

S: $L(M) = \{x \in \Sigma^* \mid x \text{ has "010" or "11" as substring}\}$

$$L(M) = \{x \in \Sigma^* \mid x \text{ can take } M \text{ to a final state}\}$$

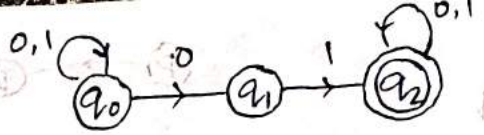
$$\left.\begin{array}{l} S \subseteq L(M) \\ L(M) \subseteq S \end{array}\right\} \Rightarrow L(M) = S.$$

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

- Σ: finite set of symbols
- q_0: initial state
- F: set of final/acc states, $F \subseteq Q$
- Q: finite set of states

$\delta : Q \times \Sigma \longrightarrow P(Q)$ (Power set of Q)

$(q_0, q_1) \Rightarrow P(Q) = \{\emptyset, \{q_0\}, \{q_1\}, \{q_0, q_1\}\}$



$\delta(q_0, 0) = \{q_0, q_1\}$

$\delta(q_0, 1) = \{q_0\}$

$\delta(q_1, 0) = \emptyset$

$\delta(q_1, 1) = \{q_2\}$

$\delta(q_2, 0) = \{q_2\}$

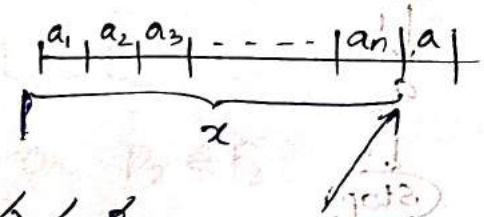
$\delta(q_2, 1) = \{q_2\}$.

$\Rightarrow \delta(p, a) = R \subseteq Q, \quad R \in P(Q)$

Base case: $\hat{\delta}(p, \in) \longrightarrow \{p\}$

Inductive case: $\hat{\delta}(p, xa)$ $\hat{\delta}(p, x)$ $\delta(\hat{\delta}(p, x), a)$

$x \in \Sigma^*, a \in \Sigma$



$\delta(p, a) = R$

$\hat{\delta}(q, x) = R$

$R = \{r_1, r_2, r_3 \ldots r_k\}$

After encountering a, possible final states are

$\delta(r_1, a) \subseteq Q$

$\delta(r_2, a) \subseteq Q$

$\delta(r_3, a) \subseteq Q$

$\vdots$

$\delta(r_k, a) \subseteq Q$.

$$\hat{\delta}(p, xa) = \bigcup_{r \in R} \delta(r_i, a)$$

$$L(M_{NFA}) = \{x \in \Sigma^* \mid \hat{\delta}(q_0, x) \cap F \neq \emptyset\}$$

$$L(M_{DFA}) = \{x \in \Sigma^* \mid \hat{\delta}(q_0, x) \in F\}$$

$N = (Q, \Sigma, \delta, q_0, F)$ $M = (Q', \Sigma, \delta', q_0', F')$

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

(p, q) – Indistinguishable.

$\hat{\delta}(p, x) \in F \Rightarrow \hat{\delta}(q, x) \in F \quad \forall x \in \Sigma^*$

$\hat{\delta}(p, x) \notin F \Rightarrow \hat{\delta}(q, x) \notin F$

– Distinguishable

$\exists x \in \Sigma^*$ s.t $\hat{\delta}(p, x) \in F$ and $\hat{\delta}(q, x) \notin F$

or vice-versa.

Procedure MARK

1) Remove all non-accessible states.

2) For a state pair (p, q) if $p \in F$ and $q \notin F$, (or vice-versa)

(p, q) are distinguishable.

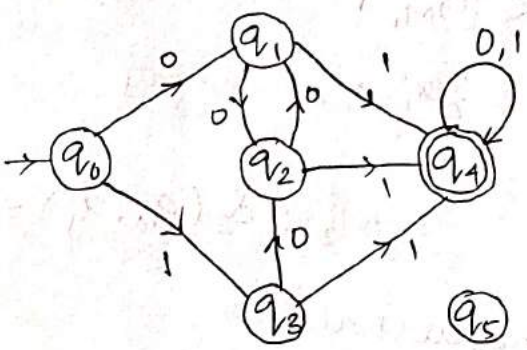
$\hat{\delta}(p, \epsilon) \in F$ & $\hat{\delta}(q, \epsilon) \notin F$.

3) Repeat until no unmarked state pairs are marked.

– for every pair (p, q) and $\forall a \in \Sigma$

$\delta(p, a) = p_a, \quad \delta(q, a) = q_a$

If p_a and q_a are distinguishable, then (p, q) are distinguishable.

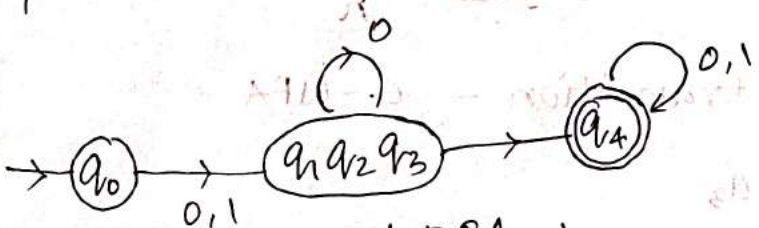


	q_0	q_1	q_2	q_3	q_4
q_0	×	✓	✓	✓	✓
q_1		×	×	×	✓
q_2			×	×	✓
q_3				×	✓
q_4					×

✓ – Distinguishable.
× – Indistinguishable.

Procedure Reduce:

1) Form sets of indistinguishable states.
$\{q_i, q_j \cdots q_k\}$; $\{q_l, q_m \cdots q_n\}$
2) For each set of individual states, form a state of DFA.
3) Initial state: The state that contains q_0.
4) Final states
5) $q_x \in \{q_i, q_j \cdots q_k\}$, $\delta(q_x, a) = q_p$.
$q_p \in \{q_l, q_m \cdots q_n\}$. $\delta(q_{i,j \cdots k}, a) = q_{l,m \cdots n}$.



Reduced DFA.

$M \longrightarrow M_1 = (Q_1, \Sigma, \delta_1, q_{01}, F_1)$

$M_2 \quad |Q_{M_2}| < |Q_{M_1}|$

$\{q_0, q_1, \ldots q_m\}$, $\{x_1, x_2, \ldots x_m\}$

$\hat{\delta}(q_0, x) = q_0$

$M_2 : \hat{\delta}_2(q_0, x_j) = \hat{\delta}_2(q_0, x_k)$

q_j, q_k

↳ distinguishable.

$\hat{\delta}(q_0, x_j) = q_j \quad \hat{\delta}(q_j, w) \in F$

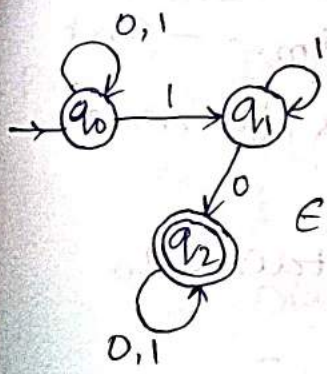
$\hat{\delta}(q_0, x_k) = q_k \quad \hat{\delta}(q_k, w) \notin F.$

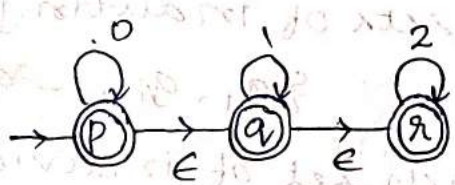
$\hat{\delta}_2(q_0, x_j w) = \hat{\delta}_2(\hat{\delta}_2(q_0, x_j), w)$

$= \hat{\delta}_2(\underbrace{\hat{\delta}_2(q_0, x_k)}_{q_i}, w)$

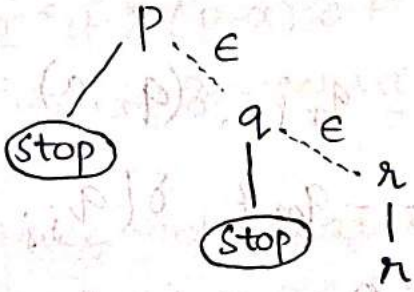
$= \hat{\delta}(q_i, w) \in F$

or $\notin F$.





ϵ-transition $M \quad \Sigma = \{0, 1, 2\}$



$\epsilon \epsilon 2 = 2$.

NFA with ϵ transition — ϵ-NFA.

$x = a_1 a_2 a_3$

$= \underbrace{\epsilon \ldots \epsilon}_{0 \text{ or more } \epsilon\text{'s}} a_1 \epsilon \ldots \epsilon a_2 \epsilon \ldots \epsilon a_3 \epsilon \ldots \epsilon$.

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

$\delta : Q \times \Sigma \cup \{\epsilon\} \longrightarrow P(Q)$

$\hat{\delta} : Q \times \Sigma^* \longrightarrow P(Q)$

$L(M) = \{0^i 1^j 2^k \mid i, j, k \geq 0\}$

Is there any language L accepted by some ϵ-NFA for which there is not DFA?

Claim: A language L is accepted by an ϵ-NFA iff it is accepted by a DFA.

ϵ-Closure(q) = $\{p \in Q \mid p$ can be reached from q by ϵ-transitions$\}$

In the previous example,

ϵ-Closure$(p) = \{p, q, r\}$

ϵ-Closure$(q) = \{q, r\}$

ϵ-Closure$(r) = \{r\}$

$\hat{\delta}(q_0, \epsilon) = \epsilon$-Closure$(q_0)$

$\hat{\delta}(q, \epsilon) = \epsilon C(q) = \{p_1, p_2, p_3 \cdots p_k\}$

$a_1 \{ r_1 \wedge \wedge \wedge \wedge r_m \}$ $\bigcup_{i=1}^{k} \delta(p_i, a_i) = R'$

ϵ-Closure(r_i) $\bigcup_{j=1}^{m} \epsilon$-Closure$(r_j)$

$= \epsilon$-Closure(R')

$N = (Q, \Sigma, \delta, q_0, F)$

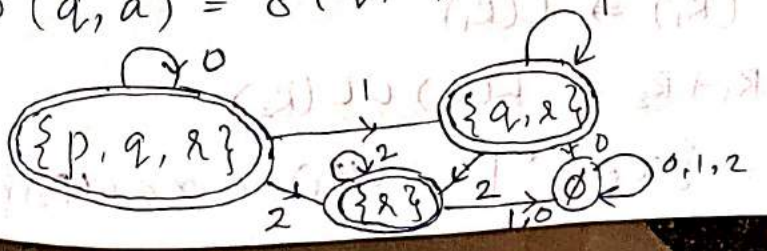
$M = (Q', \Sigma, \delta', q_0', F')$

$Q' = P(Q)$ $\quad \{q, r\}$ o † † $\emptyset$ $\emptyset$ $\{\emptyset\}$.

$q_0' = \epsilon$-Closure(q_0)

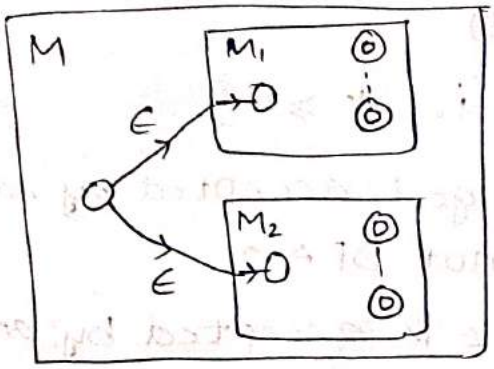
$F' = \{q \in Q' \mid q \cap F \neq \emptyset\}$

$\delta'(q, a) = \hat{\delta}(q, a)$



L_1, L_2 $\quad M_1 \to L_1$

$L_1 \cup L_2$ $\quad M_2 \to L_2$




```
if (condition) {
    expressions;
}
```

A regular expression R over Σ is defined as follows:

$\underline{\emptyset}$, $\underline{\epsilon}$ and $\underline{a}$ are regular expressions.

$a \in \Sigma$

R_1 and R_2 be two r.e's

(R_1), $R_1 + R_2$, $R_1 R_2$, $(R_1)^*$ are r.e.s.

Any r.e R can be written using these expressions finitely many times.

Ex: $\Sigma = \{0, 1\}$: $\underline{0}$, $\underline{1}$

$(\underline{0} + \underline{1})$

$(\underline{0} + \underline{1})^*$

For a r.e R over Σ denotes a language L over Σ

$\underline{\emptyset}$ denotes $L(\emptyset)$

$\underline{\epsilon}$ denotes $L(\epsilon) = \{\epsilon\}$.

$\underline{a}$ denotes $L(a) = \{a\}$

$a \in \Sigma$

If R_1, R_2 $\quad L(R_1), L(R_2)$

$(R_1) \Rightarrow L(R_1)$

$R_1 + R_2 \Rightarrow L(R_1) \cup L(R_2)$

$L(R_1 + R_2) = \{x \in \Sigma^* \mid x \in L(R_1) \text{ or } x \in L(R_2)\}$.

$R_1 . R_2$ → concatenation operator.

$R_1 \cdot R_2 \Rightarrow L(R_1) \cdot L(R_2)$

$L(R_1 \cdot R_2) = \{ xy \in \Sigma^* \mid x, y \in \Sigma^* \; ; \; x \in L(R_1) \; ; \; y \in L(R_2) \}$

$(R_1)^*$ → Closure

Kleene's Closure

x^*	L
$x^0 = \epsilon$	$L^0 = \epsilon$
$x^1 = x$	$L^1 = L$
$x^{n+1} = x x^n$	$L^2 = LL$
	$L^{n+1} = LL^n$

$x^* = \{ y \mid y = x^i, \; i \geq 0 \}$.

$x \in L^n \Rightarrow x = y_1 y_2 y_3 \cdots y_n$

$\in L \; \in L \; \in L \qquad \in L$

$L(R_1^*) = \bigcup_{i \geq 0} L(R_1)^i$

$= \epsilon, L(R_1), L(R_1) \cdot L(R_1), \cdots$

$R = (\underline{0} + \underline{1})^*$

$\underline{0} \longrightarrow \{0\}$

$\underline{1} \longrightarrow \{1\}$

$0 + 1 \longrightarrow \{0, 1\}$.

$(0+1)^* \longrightarrow \{0,1\}^*$ → set of all finite binary strings.

$((10)^* (1)^*)^*$ same as $\{0,1\}^*$.

Order of Precedence:

Top: $*$

$\cdot$

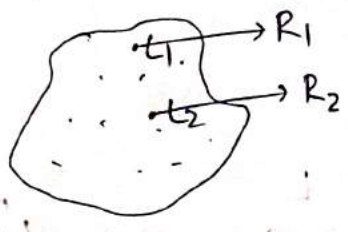
Bottom: $+$

Decreasing priority ↓

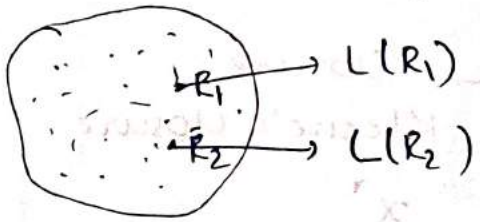
$0^*10^* \Rightarrow L$?

$0(0+1)^*0 + 1(0+1)^*1 + 0 + 1 \Rightarrow L = ?$

precisely.
The r.e's denote the class of regular languages.



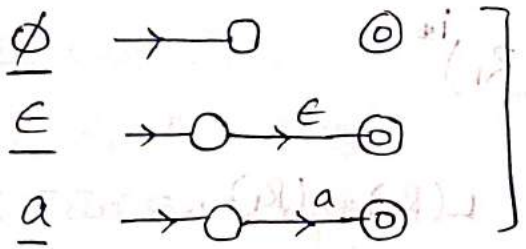
Class Reg. L's



Reg. exp's

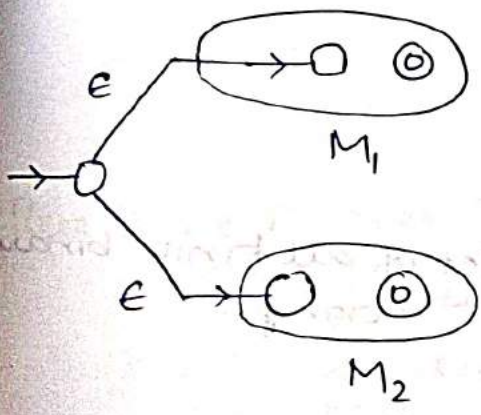
We have to show that

1. Given an R over Σ, there is a regular language $L(R)$
2. Given a language L, there exists a regular expression R such that $L(R) = L$.

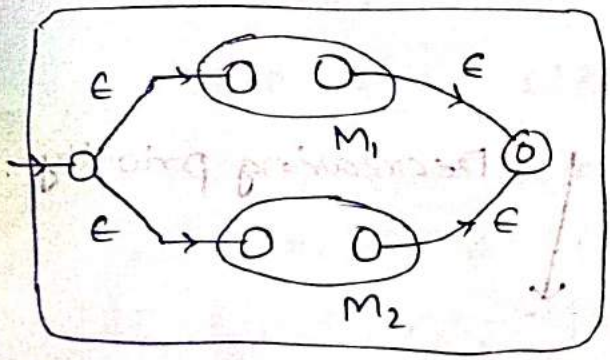


$R_1 + R_2$, $R_1 \cdot R_2$, R_1^*

$L(M_1) = L(R_1)$

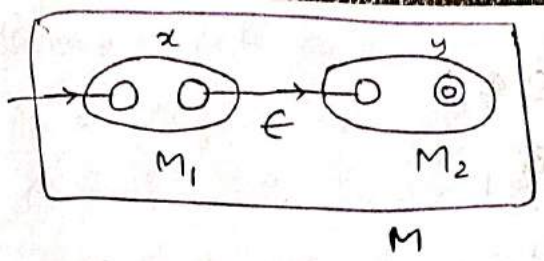


for a unique final state



$M = R_1 + R_2$

$R_1 \cdot R_2$

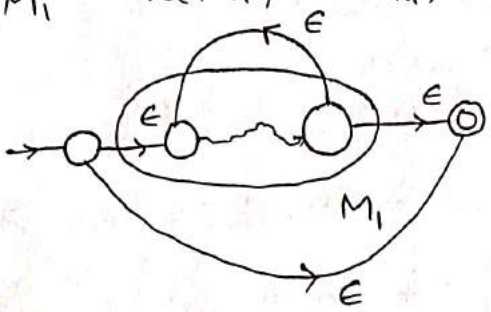


$z = \underset{\epsilon L_1}{x}\ \underset{\epsilon L_2}{y}$

$z = \underset{x_3}{x_1}\ \underset{x_4}{x_2}$

R_1^* M_1 $L(M_1) = L(R_1)$

$L(R_1^*)$



L^*

$L^0 = \epsilon$

$x = x_1 x_2 x_3 \cdots x_n$

$x_1, x_2, \cdots x_n \in L$.

$L \to M$ $\quad M = (Q, \Sigma, \delta, q_i, F)$

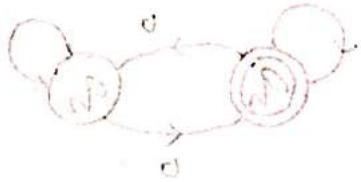
$Q = \{q_1, q_2, q_3 \cdots q_n\}$ $\quad q_i, q_j \in Q$

k is the no. of states to be passed to go from state q_i to state q_j.

$k = 0$: $q_i \longrightarrow q_j$

$k = 1$: $q_i \xrightarrow{q_1} q_j$

$k = 2$: $q_i \xrightarrow{q_1\ q_2} q_j$



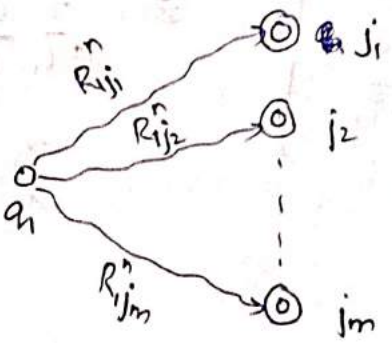
$k = n$.

$L(R_{ij}^k) = \{x \in \Sigma^* \mid \hat{\delta}(q_i, x) = q_j$ such that x takes M from q_i to q_j through a path such that the index of name of the state in the path is $\le k\}$.

$R_{ij}^n - R_{ij_k}^n \quad j_k \in F = \{j_1, j_2, \cdots j_m\}$

$L(M) = L(R_{1j_1}^n) + L(R_{1j_2}^n) + \cdots L(R_{1j_m}^n)$

$= \bigcup_{q_{j_k} \in F} L(R_{1j_k}^n)$



Define R^0_{ij}

Use it to build R^1_{ij}

Use R^1_{ij} to build R^2_{ij}

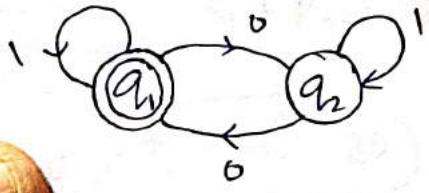
⋮

Use R^{n-1}_{ij} to build R^n_{ij}

R^0_{ij} 

$$L(R^0_{ij}) = \{ a \in \Sigma \mid \delta(q_i, a) = q_j \} \quad i \neq j$$

$$L(R^0_{ij}) = \{ a \in \Sigma \mid \delta(q_i, a) = q_j \} \cup \{ \in \} \quad i = j$$



$$L(M) = L(R^n_{1j_1}) + L(R^n_{1j_2}) + \cdots + L(R^n_{1j_m})$$

$$= \bigcup_{q_{j_k} \in F} L(R^n_{1 q_{j_k}})$$

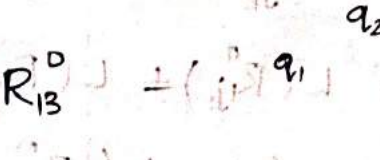
$$R^0_{11} = 1 + \in$$

$$R^0_{12} = 0$$

$$R^0_{21} = 0$$

$$R^0_{22} = 1 + \in$$

$$R^1_{23} = R^0_{21} + R^0_{13}$$



If r.l L, there exists a r.e R such that L(R) = L

$M = (Q, \Sigma, \delta, q, F)$

$Q = \{q_1, q_2 \dots q_n\}$

$F = \{q_{j_1}, q_{j_2}, \dots, q_{j_n}\} \subseteq Q$

$R_{ij}^{k} = \{x \in \Sigma^* \mid \hat{\delta}(q_i, x) = q_j$ any state on the path through which x takes M from q_i to q_j doesnot have index greater than k$\}$.

$R_{ij}^{0} = \delta(q_i, a) = q_j$,

$R_{ij}^{2} = R_{ij}^{0} + R_{ij}^{1} + R_{ij}^{2}$

$R_{ij_1}^{n}, R_{ij_2}^{n}, \dots R_{ij_m}^{n}$

$L(M) = L(R)$

$= L(R_{ij_1}^{n} + R_{ij_2}^{n} + \dots R_{ij_m}^{n})$

$= \bigcup_{q_{j_k} \in F} L(R_{1q_{j_k}})$

Define R_{ij}^{0}

Use R_{ij}^{0} to find R_{ij}^{1}

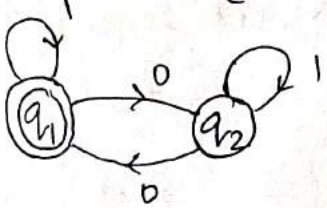
" R_{ij}^{1} " " R_{ij}^{2}

⋮

" R_{ij}^{n-1} " " R_{ij}^{n}

$L(R_{ij}^{0}) = \{a \in \Sigma \mid \delta(q_i, a) = q_j\}$ $i \neq j$

$= \{a \in \Sigma \mid \delta(q_i, a) = q_j\} \cup \{\epsilon\}$ $i = j$



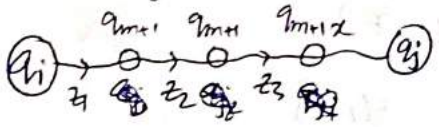
$R_{11}^{0} = 1 + \epsilon$

$R_{12}^{0} = 0$

$R_{21}^{0} = 0$

$R_{22}^{0} = 1 + \epsilon$

We know R_{ij}^{m} $m < n$.



$x = z_1 z_2 z_3 z_4$

1) The path doesn't pass through q_{m+1}

$$R_{ij}^{m+1} = R_{ij}^{m}$$

2) The path passes through q_{m+1}.

$$R_{ij}^{m+1} = R_{i,m+1}^{m} \left(R_{m+1,m+1}^{m}\right)^{*} R_{m+1,j}^{m}$$

$$R_{ij}^{m+1} = R_{ij}^{m} + R_{i,m+1}^{m} \left(R_{m+1,m+1}^{m}\right)^{*} R_{m+1,j}^{m}$$

$$R_{11}^{1} = R_{11}^{0} + R_{11}^{0} \left(R_{11}^{0}\right)^{*} R_{11}^{0}$$

$$= (1+\epsilon) + (1+\epsilon)(1+\epsilon)^{*}(1+\epsilon)$$

$$= (1+\epsilon) + 1^{*}$$

$$= 1^{*}.$$

$$(0+1)^{*} = (0^{*}1^{*})^{*}$$

Grammar:

Sentence → Subject Predicate

Subject → NP

Predicate → VP

NP → <article> <noun>

VP → <verb>

G

UNIT-2 GRAMMAR

$G = (V, T, S, P)$

$V \rightarrow$ Set of variable / non-terminals

$T \rightarrow$ Set of terminals.

$S \rightarrow$ Start Variable, $S \in V$

$P \rightarrow$ Set of production rules

$x \rightarrow y \quad x \in (V \cup T)^+ \quad y \in (V \cup T)^*$

$* \rightarrow 0/1/2 \text{ -- times}$

$+ \rightarrow 1/2/\text{-- "}$

No ϵ in $+$.
↳ epsilon

$w = u\,x\,v$

$z = u\,y\,v$

$w \Rightarrow z$ (w derives z)

$w_1 \Rightarrow w_2 \Rightarrow w_3 \cdots \Rightarrow w_n \Rightarrow w \rightarrow$ Sentence generated by G.

Sentential forms

Derivation

$\Rightarrow w_1 \overset{*}{\Rightarrow} w$

$L(G) = \{ x \in T^* / S \overset{*}{\Rightarrow} x \}$

→ $G = (V, T, S, P)$

$V = \{S, A\}$ → Capitals

$T = \{0, 1\}, \{a, b\}$ → small

$x \rightarrow y$

$x \in (V \cup T)^+$

$y \in (V \cup T)^*$

$P: S \rightarrow A/\epsilon$

$A \rightarrow aAb/\epsilon$

$S \Rightarrow A \Rightarrow aAb \Rightarrow aaAbb \overset{q}{\Rightarrow} \ldots a^q A b^q \overset{A=\epsilon}{\Rightarrow} a^q \epsilon b^q = a^q b^q$

$L(G) = \{ a^q b^q \mid q \geq 0 \}$

→ Right Linear Grammar

Left "

"

R.L.G

$A \rightarrow xB$

$A \rightarrow x$

L.L.G

$A \rightarrow Bx$

$A \rightarrow x$

$x \rightarrow$ string

$x \in T^*$

A, B → Variables $\in V$

R.L.G -

$G_1 = \{\{S\}, \{a,b\}, S, P_1\}$

$P_1 : S \rightarrow \underset{\in V}{ab}\,S / \underset{x}{a}$

L.L.G -

$G_2 = \{\{S_0, S_1, S_2\}, \{a,b\}, S, P_2\}$

$P_2 : S \rightarrow \underset{\in V}{S_1}\,\underset{x}{ab}$

$S_1 \rightarrow S_1 ab \mid S_2$

$S_2 \rightarrow \underset{x}{a}$

If L is a regular language, then there exists a linear grammar G such that $L(G) = L$

The language of a linear grammar G is regular

$V = \{V_0, V_1, \ldots V_m\}$

$T = \{a_1, a_2, \ldots a_n\}$

$S : V_0$

$V_i \rightarrow x_1 V_j$

$V_j \rightarrow x_2 V_k$

$\vdots$

$V_n \rightarrow x_l$

$\Big\} \; x \in L(G)$

$V_0 \Rightarrow x_0 V_i \Rightarrow x_0 x_1 V_j \cdots \Rightarrow x_0 x_1 \cdots x_l = x$

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

$Q = V(G), \quad \Sigma = T(G), \quad q_0 = V_0, \quad F \subseteq Q$

$\hat{\delta}(V_0, x_0) = V_i$

$\hat{\delta}(V_i, x_1) = V_j$

$\vdots$

$\hat{\delta}(V_n, x_1) = V_f \in F$

$V_i \rightarrow x_1 V_j$

$\Rightarrow \hat{\delta}(V_i, x_1) = V_j$

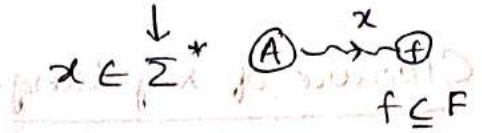
L is a regular language iff ∃ G such that G is right linear and L(G) = L.

$A \to xB$

$A \to x$

$A, B \in V, \quad x \in T^*$

$A, B \in Q \quad x \in \Sigma^*$



L is a regular language iff ∃ G such that G is left linear and L(G) = L.

$A \to Bx$ ⇒ $A \to x^R B$, $A \to x^R$ — right linear

$A \to x$

G → $\hat{G}$

$L(G) = (L(\hat{G}))^R$

$A \to x^R B$

$B \to y^R$

$A \Rightarrow x^R B \Rightarrow x^R y^R$

$A \to Bx$

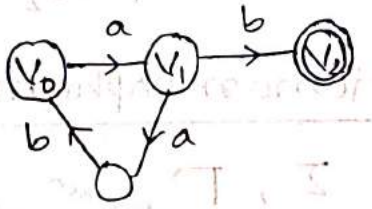
$B \to y$

$A \Rightarrow Bx \Rightarrow yx$.

$G = \{\{V_0, V_1\}, \{a, b\}, V_0, P\}$

P : $V_0 \to aV_1$

$V_1 \to abV_0 \mid b$



L, G, M

↳ right linear

M' such that $L(M') = L^R$.

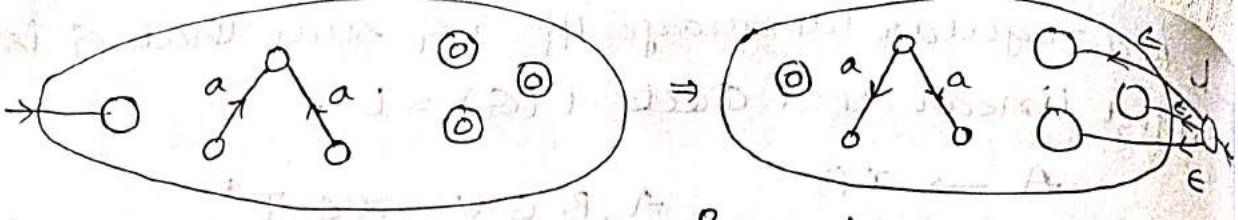
G' from M'

↳ right linear grammar

$L(G') = L^R$

G' to G'' by reversing the r.h.s of the production rules.

G'' → Left linear grammar. $L(G'') = L$.



Reversing.

Closure of reg. lang.:

1. Union
2. Intersection
3. Concatenation
4. Reversal
5. Complementation
6. Kleene Closure
7. Set Difference
8. Homomorphism.

7. Set Difference:

L_1, L_2

$L = L_1 - L_2$

$= L_1 \cap \overline{L_2}$

We should prove that L is a regular lang.

8. Homomorphism:

h → homomorphism function.

Σ, Γ

$h : a \to x$ (a: Domain, x: Range)

$a \in \Sigma$

$x \in \Gamma^*$

Ex:

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

$\Gamma = \{a, b, c\}$

$h : a \to ab$

$b \to bcc$

$w = a\ b\ a$

$h(w) = h(a)\ h(b)\ h(a)$

$= ab\ bcc\ ab$

Homomorphic image of language L

$h(L) = \{ h(x) : x \in L \}$.

If reg. lang. L is defined over Σ and h is on Σ, then $h(L)$ is also regular.

$R \quad L(R) = L$

$h(R) = a b^* a$

$= h(a)(h(b))^* h(a)$

$$\boxed{L(h(R)) = h(L(R))}$$

For a regular language L over Σ, if homomorphism h is defined on Σ, then the $h(L)$ is also regular.

$h : \Sigma \longrightarrow \Sigma^*$

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

$h = \{h(a), h(b)\}$

$= \{ab, ccb\}$

$x = aba$

$h(x) = h(aba)$

Homomorphism on strings

$= h(a) \cdot h(b) \cdot h(a)$

$= abccbab$

$h(L) = \{h(x) : x \in L\}$

$h(R) = a b^* a$

Homomorphism on reg. exprns.

$= h(a)(h(b))^* h(a)$

$= ab(ccb)^* ab$

If L, $\exists$ R s.t $L(R) = L$

$h(L(R)) = L(h(R))$

BASIS of any reg. exprns — $\phi, \epsilon, a \;.\; a \in \Sigma$.

$L(\phi) = \phi$	$L(\epsilon) = \{\epsilon\}$
equal $\begin{cases} h(L(\phi)) = \phi \\ L(h(\phi)) = \phi \end{cases}$	$\begin{cases} h(L(\epsilon)) = \{\epsilon\} \\ h(\epsilon) = \{\epsilon\} \end{cases}$ equal.

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

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

If R, E a subexpression of R.

$E = F + G$, $E = F \cdot G$, $E = F^*$

$L(h(R)) = h(L(R))$

$E = F + G$

$L(E) = L(F+G) = L(F) \cup L(G)$

$L(F), L(G)$ - sets

$h(L(E)) = h(L(F) \cup L(G))$

$= h(L(F)) \cup h(L(G))$

$L(h(E)) = L(h(F+G))$

$= L(\underbrace{h(F)}_{R_1} + \underbrace{h(G)}_{R_2})$

$= L(h(F)) \cup L(h(G))$

$= h(L(F)) \cup h(L(G))$

$L = F \cdot G$

$h(L) = h(F \cdot G)$

$= h(F) \cdot h(G)$

$w_1 \in L(F)$

$w_2 \in L(G)$

$w = w_1 \cdot w_2$

$h(w) = h(w_1) \cdot h(w_2)$

$L(h(E)) = L((h(F)) \cdot h(G))$

$= L(h(F)) \cdot L(h(G))$

$L = \{ x \in \{0,1\}^* \mid x = 0^n 1^n \text{ when } n \geq 0 \}$

$x = \overset{q_0}{\uparrow} a_1 \overset{q_{s_1}}{\uparrow} a_2 \overset{q_{s_2}}{\uparrow} \cdots a_n \overset{q_{s_n}}{\uparrow}$

K states in the DFA.

$|x| = 2k$ | state seq. | $= 2k+1$

$x \in L$ $x = uvw$

$uvw \in L$

$uvvw \in L$

$\vdots$

~~the~~ $uv^i w \in L$

$x \in L$, L is regular

uv^*w

$x = u\ v\ w$

$q_0 \to q$ $q \to q$ $q \to q_f \in F$

Pumping Lemma:

If L is a reg. lang., then there exists a constant k, which is dependent on L such that for all $x \in L$ where $|x| \geq k$, there exists strings u, v, w such that

1. uvw is actually x
2. $|uv| \leq k$ and $|v| > 0$
3. for all $i \geq 0$, $uv^i w \in L$.

$$L = \{ x \in \{0,1\}^* \mid x = 0^n 1^n \quad n \geq 0 \}$$

$L = \{ xx^R : x \in \Sigma^* \}$. Is L regular?

$$L^R = \{ x \in \Sigma^* \mid x^R \in L \}$$

$x_1^R \cdot x_2^R \in L^R$ if $x_1, x_2 \in L$

$L \cdot L^R$ So $L \cdot L^R \neq xx^R$.

✓ $x_1 x_1^R$ $x_1 x_2^R$ ✗

✓ $x_2 x_2^R$ $x_2 x_1^R$ ✗

- Adversary gives a number k, then

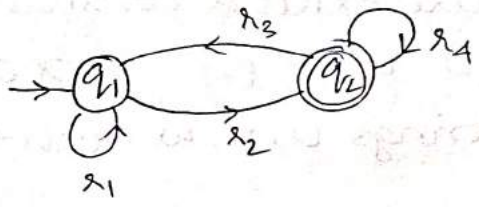
$x \in L$, $|x| \geq k$

$$x = \underbrace{aaa \cdots aaa}_{k} \underbrace{aa \cdots aa}_{k}$$

Adversary decomposes x into u, v, w s.t

$|uv| \leq k$, $|v| > 0$

Algebraic Laws of regular expressions:



$R_{11}^{*} = r_1 + \epsilon$

$R_{12}^{0} = r_2$

$R_{21}^{0} = r_3$

$R_{22}^{0} = r_4 + \epsilon$.

$$R_{11}^{1} = R_{11}^{0} + R_{11}^{0}(R_{11}^{0})^{*} R_{11}^{0}$$

$$= (r_1+\epsilon) + (r_1+\epsilon)(r_1+\epsilon)^{*}(r_1+\epsilon)$$

$$= r_1^{*}$$

$$R_{12}^{1} = R_{12}^{0} + R_{11}^{0}(R_{11}^{0})^{*} R_{12}^{0}$$

$$= r_2 + (r_1+\epsilon)(r_1+\epsilon)^{*} r_2$$

$$= r_1^{*} r_2$$

$$R_{21}^{1} = R_{21}^{0} + R_{21}^{0}(R_{11}^{0})^{*} R_{11}$$

$$= r_3 + r_3 r_1^{*}$$

$$= r_3 r_1^{*}$$

$$R_{22}^{1} = R_{22}^{0} + R_{21}^{*}(R_{11}^{0})^{*} R_{12}$$

$$= (r_4+\epsilon) + r_3(r_1+\epsilon)^{*} r_2$$

$$= r_4 + r_3 r_1^{*} r_2 + \epsilon$$

$$R_{12}^{2} = R_{12}^{1} + R_{12}^{1}(R_{22})^{*} R_{22}$$

$$= r_1^{*} r_2 + r_1^{*} r_2 (r_4 + r_3 r_1^{*} r_2 + \epsilon)^{*}(r_4 + r_3 r_1^{*} r_2 + \epsilon)$$

$$= r_1^{*} r_2 (r_4 + r_3 r_1^{*} r_2)^{*}$$

Decision Problems on r.L:

1) Set membership problem.
2) Check if a reg. lang. is empty, finite or infinite
3) Check if the r.L's L_1 & L_2 are equal.
   ↳ Try using set diff.

Context-free languages:

$G = (V, T, S, P)$ T as Σ

$P: x \to y$

$x \in (V \cup T)^+$

$y \in (V \cup T)^*$

$A \to xB \mid A \to Bx$

$A \to x \mid A \to x$.

$V = \{S, A, B\}$

$T = \{a, b\}$

$P: S \to aB$

$S \to bA$

$A \to aS$

$A \to bAA$

$A \to a$

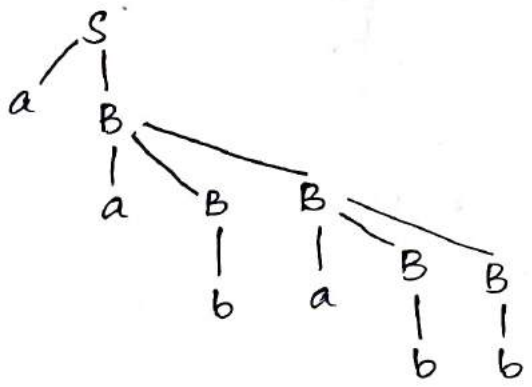
$B \to bS$

$B \to aBB$

$B \to b$.

$S \Rightarrow aB \Rightarrow aaBB \Rightarrow$
$aaBaBB \Rightarrow aaBaBB \Rightarrow$
$aaBabb \Rightarrow aababb$.

Parse tree -



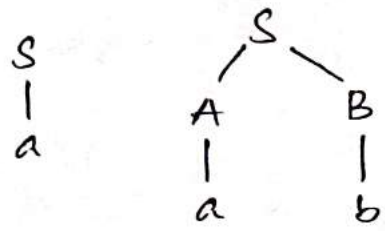
CNF

1) $S \to a$

2) $S \to AB$

$A \to a$

$B \to b$



3) $S \to A_1B_1$

$A_1 \to A_2A_3$

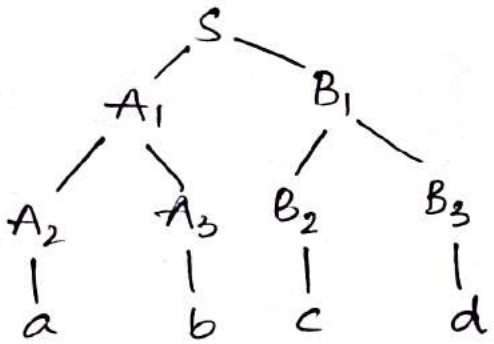
$B_1 \to B_2B_3$

$A_2 \to a$

$A_3 \to b$

$B_2 \to c$

$B_3 \to d$



Case-1) Path length = 1

Variables / Non-terminals = 1

String length = 1.

Case-2) Path length = 2

N.t = 2

String length = 2

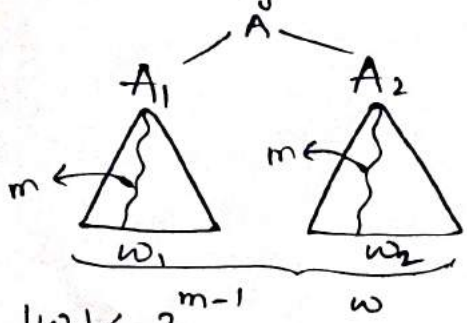
Case-3) Path length = 3

N.t = 3

String length = 4.

$\Rightarrow$ P.L = N.t = n.

Str leng $\leq 2^{n-1}$



$|w_1| \leq 2^{m-1}$

$|w_2| \leq 2^{m-1}$

Δ – Tree

Δ → Max. path length.

$|w| \leq 2 \cdot 2^{m-1}$

$|w| \leq 2^m$

L_R - Regular lang.

L_{CF} - Context Free Lang.

$n \leftarrow$ pumping lemma constant.

We can't use pumping lemma to prove one lang. is regular. What we can say is that if a lang. is regular, there exists n.

For L_R, n - no. of states in the hypothetical FSM

For L_{CF}, n - no. of non-terminals in the corresponding grammar.

Let L be a CFL. Then there is a constant n that depends on L such that $\forall z \in L$, with $|z| \geq n$, there exist u, v, w, x, y satisfying

1) $z = uvwxy$ - decomposition of z.

2) $|vx| \neq 0$, $|vwx| \leq n$

3) $\forall i \geq 0$, uv^iwx^iy is also in L.

$vx \neq \in$

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

$L = \{a^n b^n c^n \mid n \geq 0\}$.

Let K be the pumping lemma.

1) $z = uvwxy$

$|vx| \neq 0$, $|vwx| \leq k$

$\forall i$, $uv^iwx^iy \in L$.

$z = a^k b^k c^k$

$vwx \rightarrow$ all a

" b

" c

a and b

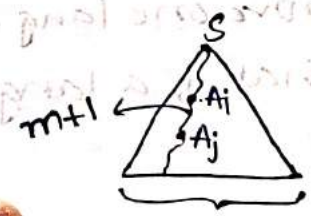
b and c

Consider these cases!

vx contains a finite no. of a's which is $\leq k$.

In such case, pumping lemma fails.

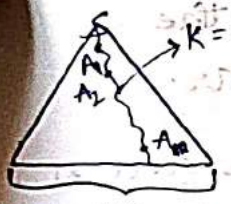
Suppose ∃ a lang. L, grammar G with m no. of non-terminals



$n = 2^m$

$|z| \geq 2^m$

$\Rightarrow$ For equality, max. p.l = m+1



$|w| \leq 2^{k-1}$

L $G = (V, T, S, P)$

$|V| = m$

n - pumping constant of L.

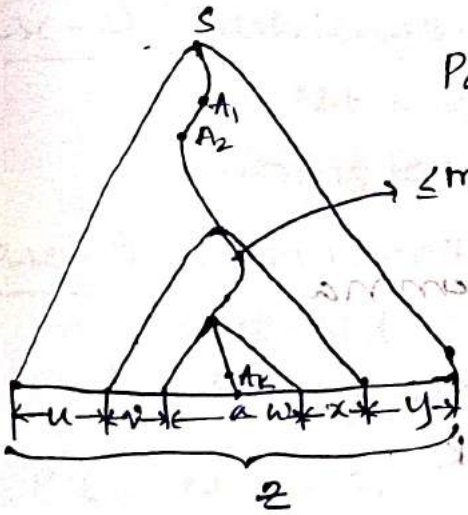
$\boxed{n = 2^m}$

L → Grammar → FSM

$n = 2^m$ → n

$\Rightarrow |w| \geq 2^m$

$\Rightarrow$ Path length $\geq m+1$



Path length $\geq m+1$

$|V| = m$

$v x \neq \epsilon$

$|vwx| \leq n$

$A \overset{*}{\Rightarrow} vAx$

$A \overset{*}{\Rightarrow} vwx$

$A \overset{*}{\Rightarrow} w$

$A \overset{*}{\Rightarrow} AX$

$S \overset{*}{\Rightarrow} uvwxy$

$S \overset{*}{\Rightarrow} uAy$

$S \overset{*}{\Rightarrow} uvAxy$

$S \overset{*}{\Rightarrow} uvvAxxy$

$S \overset{*}{\Rightarrow} uv^iAx^iy$

$\boxed{S \overset{*}{\Rightarrow} uv^iwx^iy}$

$L = \{ a^i b^j c^k d^l \mid \text{either } i=0 \text{ or } j=k=l \}$.

$i=0$

$w = b^n c^m d^p$

$|vwx| \le n$ (p.l const.)

$j=k=l$

$w = a^n b^m c^m d^m$

$|vwx| \le n$

The class of CFLs have some closure properties.

Closed under:
1) Union
2) Kleene's closure
3) Concatenation.

Not closed under:
1) Intersection
2) Complementation.

$L_1 \cdot L_2 \Rightarrow L_1 = L(G_1)$, $L_2 = L(G_2)$

$G_1 = (V_1, T, S_1, P_1)$

$G_2 = (V_2, T, S_2, P_2)$

$V_1 \cap V_2 = \phi$ ↳ else contradictions occur.

$L = L_1 \cup L_2$

$L(G') = L_1 \cup L_2$.

~~$G = (\{S\} \cup V_1$~~

$G = (\{S\} \cup V_1 \cup V_2, T, S, P)$

$P = P_1 \cup P_2 \cup \{S \to S_1 \mid S_2\}$

$S \to S_1 \mid S_2$

$L = \{ a^i b^j c^k d^l \mid \text{either } i=0 \text{ or } j=k=l \}$

$i=0$

$w = b^i c^k d^l$

$|vwx| \le n$

$j=k=l$

$w = a^i b^i c^i d^i \quad i>0$

$|vwx| \le n$

Steps of procedure:
1) Choice of pumping length n ← done by adversary
2) Choice of string w ← Done by you.

3) Choice of decomposition ← Done by adversary

4) Show if pumping can be done ← By you.

Adversary tries to prove that given lang. is CFL. So, when given a string, adversary always tries to decompose it such that pumping lemma is satisfied. (Even if there are other decompositions which prove that pumping lemma fails.)

→ Incase of non-regular lang., we can always find a string such that no possible decomposition will satisfy the pumping lemma. But this is not the case with CFL ⇒ Drawback of this version of P.L.

Closure Properties of CFL:

2) Concatenation:

$L = L_1 L_2$

$L(G') = L$

$G' = (V_1 \cup V_2, T, S', P')$

3) Kleen's Closure

L^* : $L_1 = (V_1, T_1, S_1, P_1)$

$L = L^*$

$L = (V_1, T_1, S, P)$

$P = P_1 \cup \{S \to SS\}$

4) Intersection:

$L_1 = \{a^i b^j c^j \mid i, j \geq 1\}$

$L_2 = \{a^j b^j c^k \mid j, k \geq 1\}$

$L_1 \cap L_2 = \{a^k b^k c^k \mid k \geq 1\}$ ⇒ Not CF

$L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$

1. Not CF

which is not true.

If L is CF, let's assume $\bar{L}$ is CF

$\Rightarrow L_1, L_2 \to CF \Rightarrow \bar{L_1}, \bar{L_2} \to CF$

$\Rightarrow \bar{L_1} \cup \bar{L_2} \to CF$

$\Rightarrow \overline{\bar{L_1} \cup \bar{L_2}} \not\to CF$

→ Hence our assumption is wrong.

5) Complementation
- is not closed.

L_R, w

Membership problem: ? $w \in L_R$?

Pushdown Automata: (PDA)

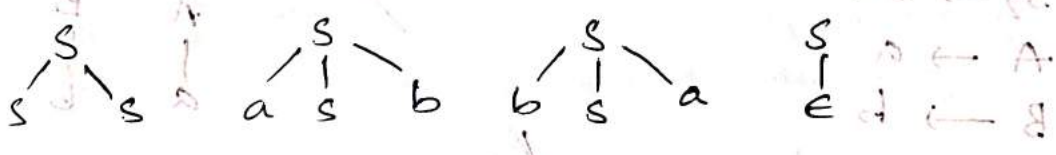
L_{CF}, w

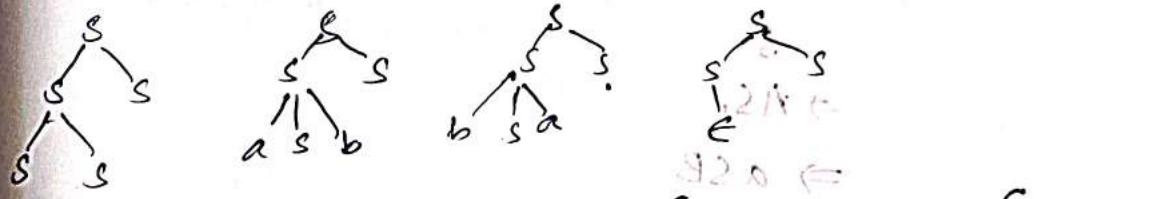
1) Parse trees

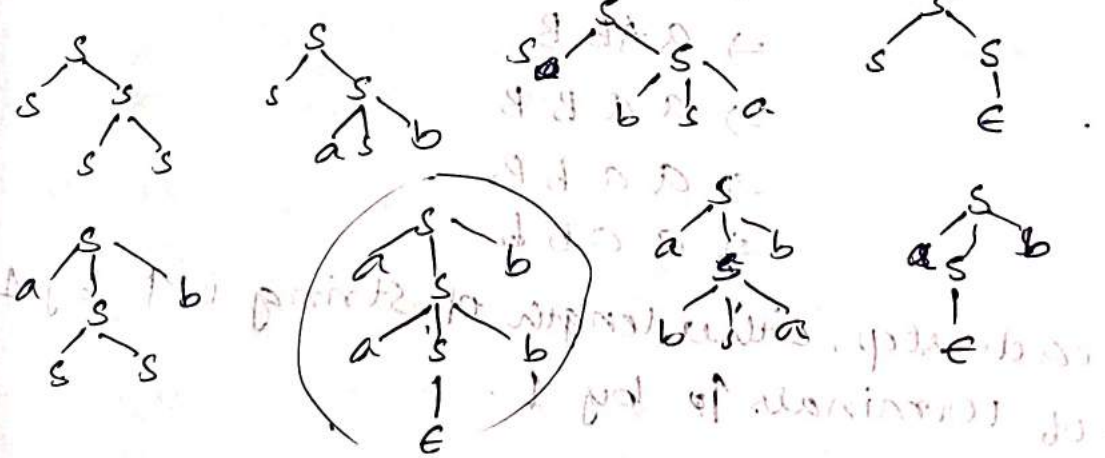
$G_{CF} = (V, T, S, P)$

$S \to SS \mid aSb \mid bSa \mid \epsilon$

$w = aabb$

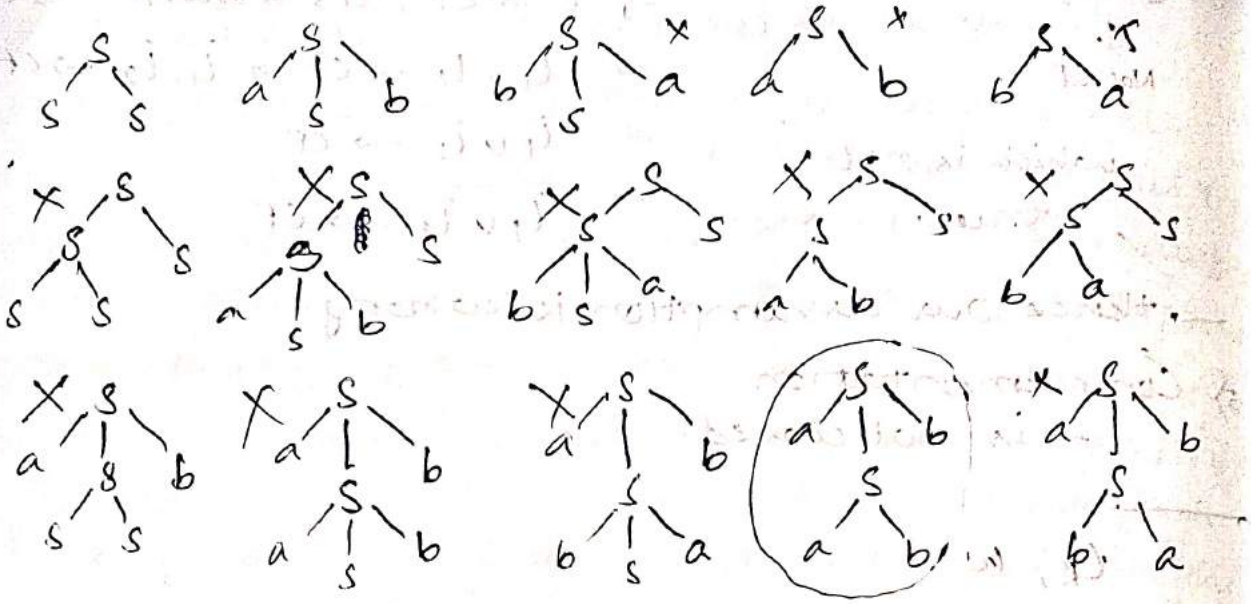






aab

$S \rightarrow SS \mid asb \mid bsa \mid ab \mid ba$ $\quad w = aabb$



$S \rightarrow SS \mid ASB \mid BSA \mid AB \mid BA$

$A \rightarrow a$

$B \rightarrow b$

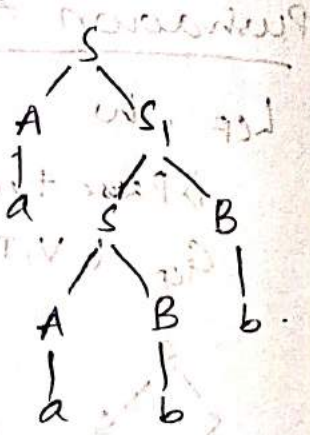
$S \rightarrow SS \mid AS_1 \mid BS_2 \mid AB \mid BA$

$S_1 \rightarrow SB$

$S_2 \rightarrow SA$

$A \rightarrow a$

$B \rightarrow b$



S

$\Rightarrow AS_1$

$\Rightarrow aSB$

$\Rightarrow aABB$

$\Rightarrow aaBB$

$\Rightarrow aabB$

$\Rightarrow aabb$

At each step, either length of string is ↑ by 1 or no. of terminals ↑ by 1.

20-3-23

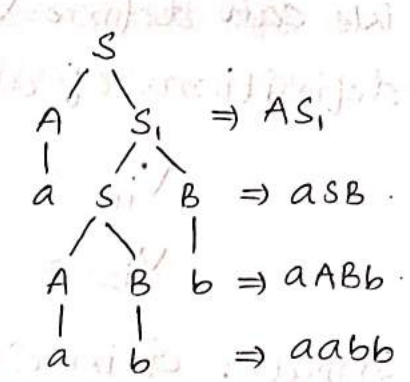
$S \to SS \mid AS_1 \mid BS_2 \mid AB \mid BA$

$S_1 \to SB$

$S_2 \to SA$ $\quad\quad x = aabb$

$A \to a$ $\quad\quad |x| = n$

$B \to b$



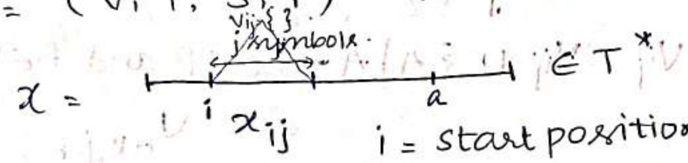
$S \to AS_1 \to aS_1 \to aSB \to aABB \to aaBB \to aabB \to aabb$

For every step, either length of sentential form is increasing by 1, or no. of terminals is increasing by 1.

When given a string, to find whether it is present in the lang. or not -

Cocke - Younger - Kasami (CYK) algorithm:

$G = (V, T, S, P)$ G in CNF. $\quad |x| = n$



i = start position of the string

j = Length of the substring.

Indexing of symbols starts from 1.

$\Rightarrow x \equiv x_{1n}$.

i^{th} symbol $\equiv x_{i1}$

$V_{ij} = \{ A \mid A \overset{*}{\Rightarrow} x_{ij} \}$.

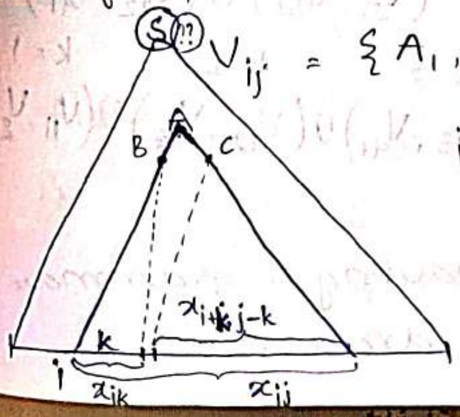
V_{ij} is the set of non-terminals which generate the substring x_{ij}.

If $A_1 \to a$, $A_2 \to a$ ... $A_k \to a$, then

$V_{ij} = \{A_1, A_2, A_3 \ldots A_k\}$ for $x_{i1} = a$.

index of symbol after x_{ik} is "$i+k$".

$A \to BC$



We can define V_{ij} provided we already have the definitions of all V_{lk} where $l < j$ for all $l, k \geq 1$

$$V_{i1} = x$$

$$V_{i2} = x - 1$$

Input : G in CNF, x ($|x| = n$)

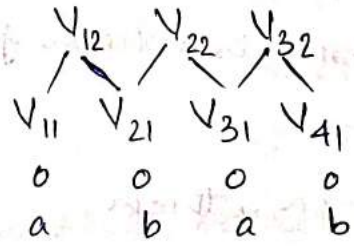
Output : Determine V_{1n}. Return "Yes" if $S \in V_{1n}$.

Steps :

1. for j = 1 to n
2. $V_{j1} = \{ A \mid A \to a$ is in P, "a" the i^{th} symbol of $x \}$
3. for j = 2 to n {
4. for i = 1 to n − j + 1 {
5. $V_{ij} = \phi$
6. for k = 1 to j − 1 {
7. $V_{ij} = V_{ij} \cup \{ A \mid A \to BC \in P$ and $B \in V_{ik}$ and $C \in V_{i+k, j-k} \}$
8. }
9. }
10. }

V_{14} check if $S \in V_{14}$??

V_{13} V_{23}



$$V_{21} = (V_{11}, V_{21}) \quad (k = 1)$$

$$V_{13} = (V_{11}, V_{22}) \cup (V_{12}, V_{31}) \quad (k = 1, k = 2)$$

$$V_{23} = (V_{21}, V_{32}) \cup (V_{22}, V_{41}) \quad (k = 1, k = 2)$$

$$V_{14} = (V_{13}, V_{41}) \cup (V_{12}, V_{32}) \cup (V_{11}, V_{23}) \quad (k = 3, k = 2, k = 1)$$

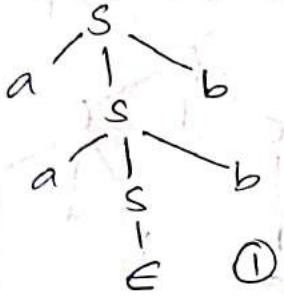
Unambiguous grammar :

For every string s in the lang. L having a grammar G, a unique parse tree can be drawn.

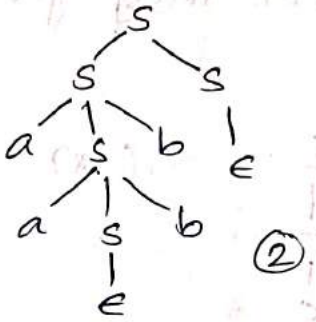
But if there is atleast 1 string in L s.t multiple parse trees are possible, it is called "ambiguous grammar".

Ex: $S \rightarrow SS \mid asb \mid \epsilon$

aabb.



①



②

Def: A CFG G is said to be ambiguous if there exists some $w \in L(G)$ that has atleast two distinct parse trees.

There can be any possible no. of grammars for a particular language. If all these grammars of a language are ambiguous, then the language is called "inherently ambiguous".

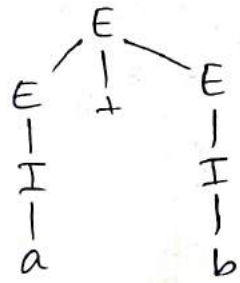
$E \rightarrow E+E \mid E^* E \mid (E) \mid I$

$I \rightarrow a \mid b \mid c$

How do you parse a+b ?

1) $E \Rightarrow E+E \Rightarrow E+I \Rightarrow E+b \Rightarrow I+b \Rightarrow a+b$.

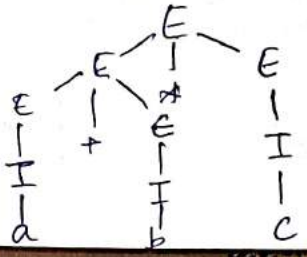
2) $E \Rightarrow E+E \Rightarrow I+E \Rightarrow a+E \Rightarrow a+I \Rightarrow a+b$.



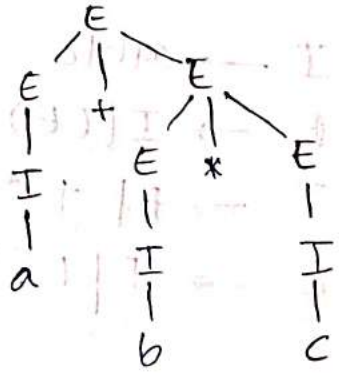
There is no difference between the derivations of 1 & 2.

How to parse $a+b^*c \Rightarrow$

1)



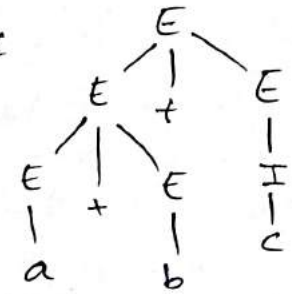
2)



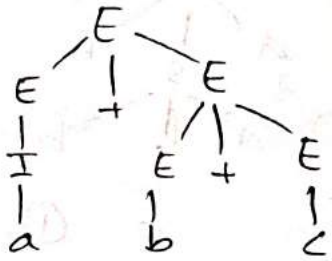
(OR)

Here, both the parse trees are different. Hence the grammar is ambiguous. (Just for us, parse tree 1 is right bcoz it carries out multiplication first then addition) But we can't find a way to ensure that for now. So both trees are right since they generate the expression.

a+b+c



(or)



Prblm: Doesn't give proper order of execution.

Things to be addressed:

1) Precedence relation
2) Ordering / Direction of approach.

1) Factors:

Entities which can't be broken at any point.

a) Identifiers - a+, a*
b) Paranthesis

2) Term: T

Expressions that ~~can~~ cannot be broken by +.

a) Factor.
b) Expression with * operator.

3) Expression: E

That can be broken by + or *.

$I \rightarrow a|b|c$

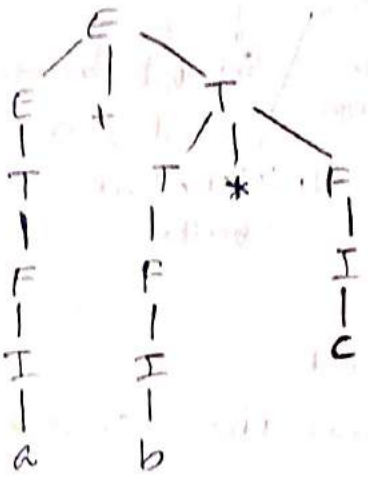
$F \rightarrow I|(E)$

$T \rightarrow F|T*F$

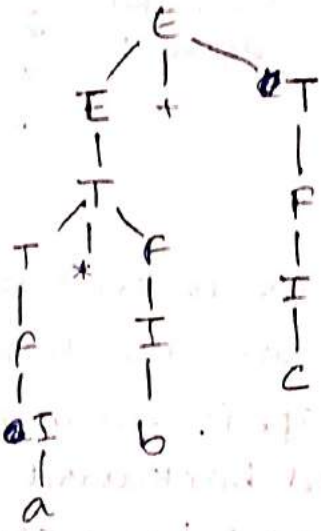
$E \rightarrow T|E+T$

These can be used to remove ambiguities

Ex: a + b * c



a * b + c



$E \rightarrow E+E \mid E*E \mid (E) \mid I$ } - Ambiguous

$I \rightarrow a \mid b \mid c$

$I \rightarrow a \mid b \mid c$
$F \rightarrow I \mid (E)$
$T \rightarrow F \mid T*F$
$E \rightarrow T \mid E+T$
} Un-ambiguous... (Ambiguity Removed)

Push Down Automata (PDA)

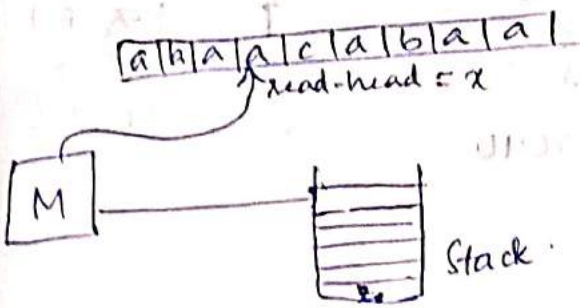
$S \rightarrow asa \mid bsb \mid c$

abaacaaba

Lang. is not regular

Grammar is context-free.

PDA has stack.

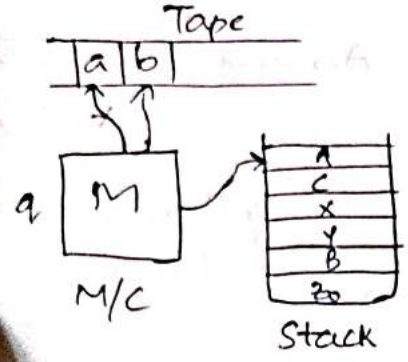



```
while (x != 'c')
    push(x);
After x == 'c'.
while(x)
    if (x == top)
        pop;
```

$M = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$

- Q: States
- Σ: Tape alphabet / alphabet
- Γ: Stack symbols
- q_0: Initial state of the machine
- F: Set of final states, $F \subseteq Q$
- δ: Transition function
- z_0: Initial stack symbol.

1) The tape is read-only.
2) All writes are done on the stack.
3) The tape is scanned in one direction, and never traced backward.
4) The stack head always operates on the stack top.



q → current state of PDA.

$(q, a, A) \rightarrow (p, \gamma)$

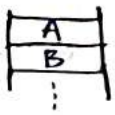
$p \rightarrow$ next state.

$\gamma \in \Gamma^*$

$\Gamma = \{A, B\}$.

$\Gamma^* = \{\epsilon, A, B, AA, AB, \ldots\}$

Stack Operation: Can pop one element (the top element) of the stack and then push γ into stack from right to left. i.e.,

If $\gamma = AB \Rightarrow$ Stack: 

1) When you have multiple transition options,

$(q, a, A) \rightarrow \{(p_1, \gamma_1), (p_2, \gamma_2), \ldots (p_k, \gamma_k)\}$

2) When you have ϵ-transitions

$(q, \epsilon, A) \rightarrow \{(p_1, \gamma_1), (p_2, \gamma_2) \ldots (p_m, \gamma_m)\}$.

$\delta : Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma \rightarrow P(Q \times \Gamma^*)$

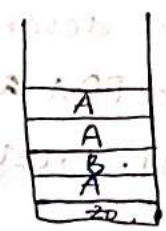
$L = \{ww^R \mid w \in \{0,1\}^*\}$.

$\Gamma = \{A, B\}$

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

Let's check for ~~00~~01000010.

Phase-1:



$\delta(q_0, 0, z_0) = \{(q_0, A, z_0)\}$

$\delta(q_0, 1, z_0) = \{(q_0, B, z_0)\}$.

$\delta(q_0, 0, A) = \{(q_0, AA)\}$.

$\delta(q_0, 1, A) = \{(q_0, BA)\}$.

$\delta(q_0, 0, B) = \{(q_0, AB)\}$

$\delta(q_0, 1, B) = \{(q_0, BB)\}$.

$\delta(q_0, \epsilon, A) = \{(q_1, A)\}$

$\delta(q_0, \epsilon, B) = \{(q_1, B)\}$

$\delta(q_0, \epsilon, z_0) = \{(q_1, z_0)\}$.

Phase-2:

$\delta(q_1, 0, A) = \{(q_1, \epsilon)\}$.

$\delta(q_1, 1, B) = \{(q_1, \epsilon)\}$.

$\delta(q_1, \epsilon, z_0) = \{(q_2, z_0)\}$.

24/3/25

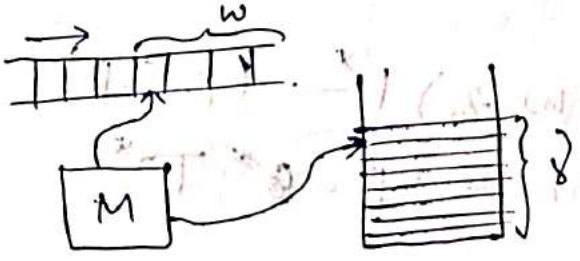
A PDA accepts

1) on final states

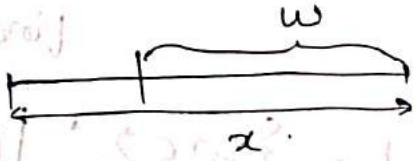
2) on empty stack.

Instantaneous description of a PDA (ID):

$M = (Q, \Sigma, T, \delta, q_0, z_0, F)$



(q, w, γ)



q - current state of M

γ - current stack content

w - portion of input string that is yet to be traversed.

The description which captures the overall state of the PDA at a certain instance is called the instantaneous description (ID) of the PDA.

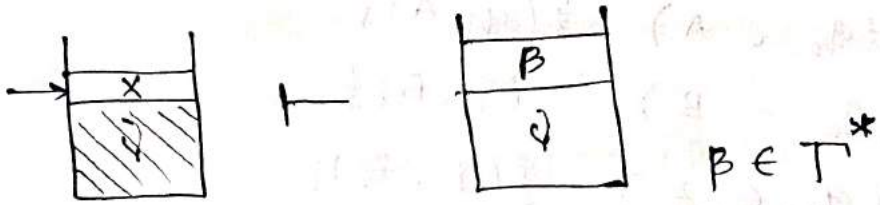
(q_0, x, z_0) - Description of the very initial state of PDA

$\vdash$ → "derives in one step".

↳ Operator.

$(q, aw, X\delta) \vdash (p, w, \alpha\delta)$ ID_2

ID_1 $(p, \alpha) \in \delta(q, a, X)$



$\beta \in \Gamma^*$

$\vdash^*$

$L = \{ww^R \dots\}$

0 1 0 0 0 0 1 0

$(q_0, \downarrow, z_0) \vdash (q_0, 1000010, Az_0) \vdash$

$(q_0, 000010, BAz_0) \vdash (q_0, 00010, ABAz_0) \vdash$

$(q_0, 0010, AABAz_0) \vdash (q_1, 0010, AABAz_0) \vdash^*$

$(q_1, \epsilon, z_0) \vdash (q_2, \epsilon, z_0)$

↑ final state

$L = \{w \in \Sigma^* \mid (q_0, w, z_0) \vdash^* (p, \epsilon, \beta)\}$.

$p \in F$ and $\beta \in \Gamma^*$.

Acceptance by empty stack:

$L = \{0^n 1^n \mid n \geq 0\}$

$(q, 0011, Z_0) \vdash (q, 011, AZ_0) \vdash (q, 11, AAZ_0) \vdash$

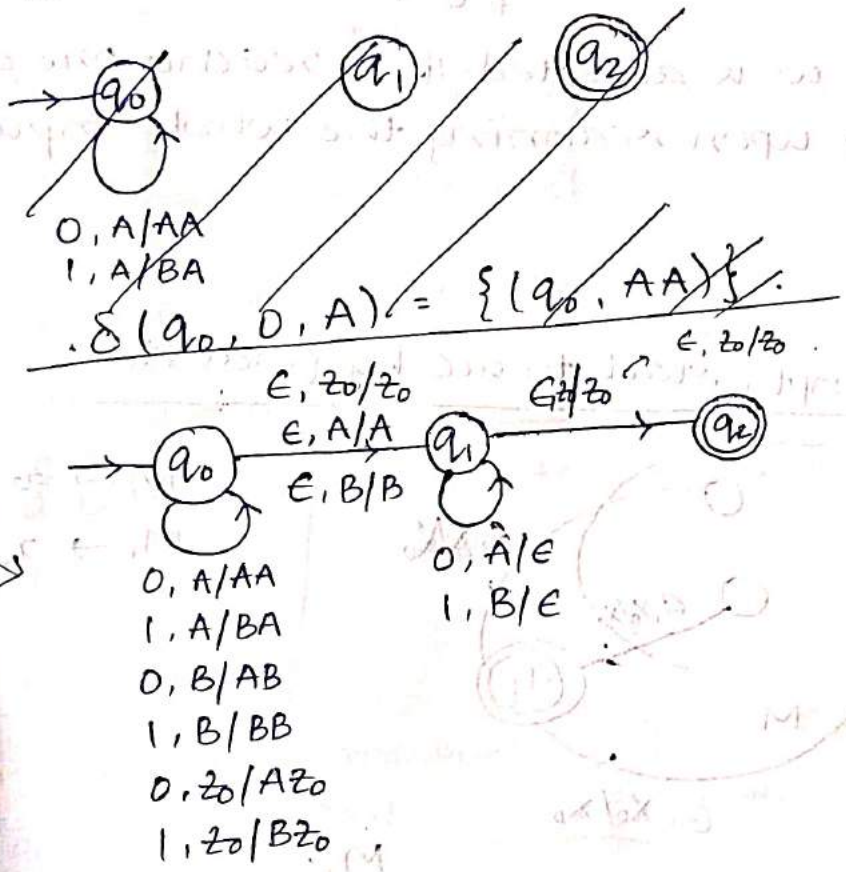
$(q, 1, AZ_0) \vdash (q, \epsilon, Z_0) \vdash (q_2, \epsilon, \epsilon)$

$L \vdash (q, \epsilon, AZ_0)$

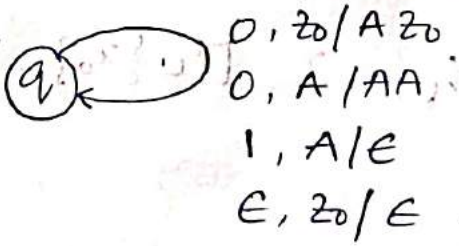
$L = \{w \in \Sigma^* \mid (q_0, w, Z_0) \vdash^* (q, \epsilon, \epsilon)\}$.

$q \in Q$.

$L = \{ww^r \mid w \in \Sigma^*\}$.

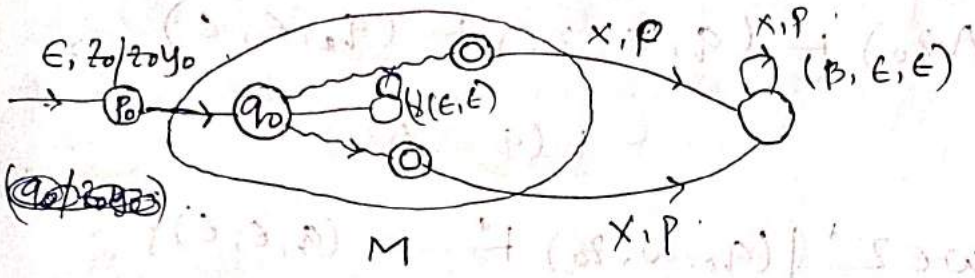


$L = \{0^n 1^n \mid n \geq 0\}$



1) Acceptance by final state
2) Acceptance by empty stack
} Check equivalence.

1) Acc. by final state to acc. by empty stack:



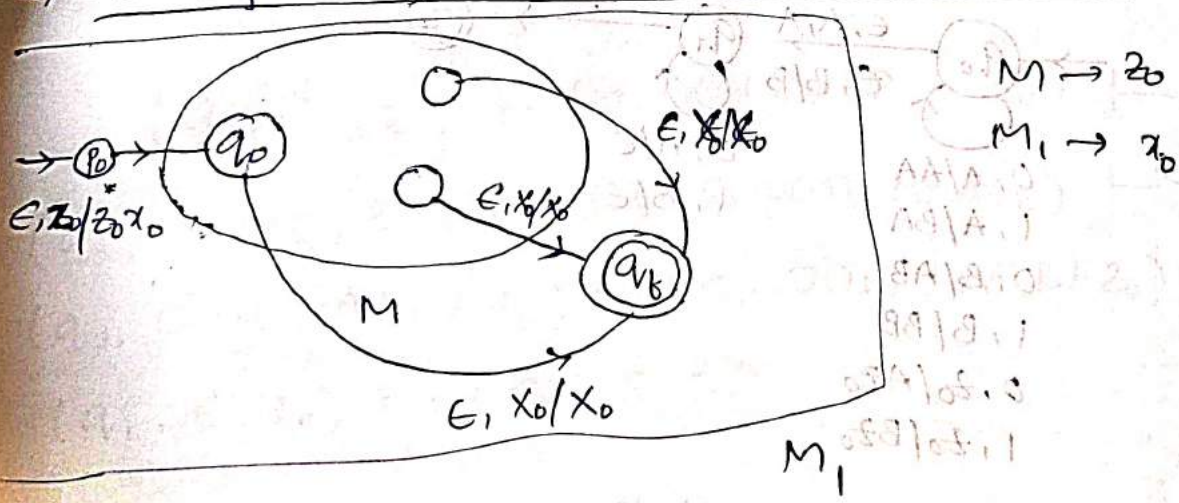
$$L(M) = \{ w \in \Sigma^* \mid (q_0, w, z_0) \vdash^* (q_f, \epsilon, \beta) \}$$

$q_f \in F$

$\beta \in \Gamma^*$

The string w is accepted if M reaches one of the final state upon scanning the whole input string

2) Acc. by empty stack to acc. by final state:



$$M = (Q, \Sigma, \Gamma, \delta, q_0, z_0, \phi)$$

$$M_1 = (Q \cup \{p_0, q_f\}, \Sigma, \Gamma \cup \{x_0\}, x_0, \{q_f\}, \delta_1)$$

Class of CFLs are

1) closed under
   a) Union
   b) Concatenation
   c) Kleene closure
   d) String homomorphism

2) Not closed under
   1) Intersection
   2) Complementation.

↳ Σ, Γ

- $h : \Sigma \to \Gamma^*$

$\Sigma = \{0,1\}$ $\Gamma = \{a,b,c\}$.

$h(0) = acc$

$h(1) = ba$

$h(011) = h(0)h(1)h(1)$

$= accbaba.$

The class of CFLs is not closed under complementation

$L, \bar{L}$

- If a lang. L is not context-free, can $\bar{L}$ be a CFL?

Turing Machines:

Alan Turing (mathematician) in 1936 gave the idea of turing machine which is the basis of modern computer.

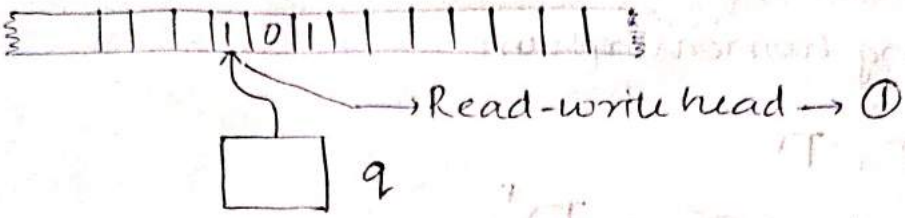
Motivation: To capture the notion of computability using a machine.

Computational problems are solved by algorithms.

"If there exists an algo to solve a problem, that algo can be implemented using a Turing Machine (TM)."

Church-Turing Thesis:

If a computational problem can be solved algorithmically, then it can be solved using a Turing machine and vice-versa.



② The tape extends infinitely in both directions.

③ The head can move in both directions.

Current state	Current i/p symbol	Next state	Symbol written on tape	Movement direction
q	1	p	0	R
q_0	a	q_1	x	R
q_1	a	q_1	a	R
q_1	b	q_2	y	R
q_2	b	q_2	b	R
q_2	c	q_3	z	L.

$(q, 1, p, 0, R)$

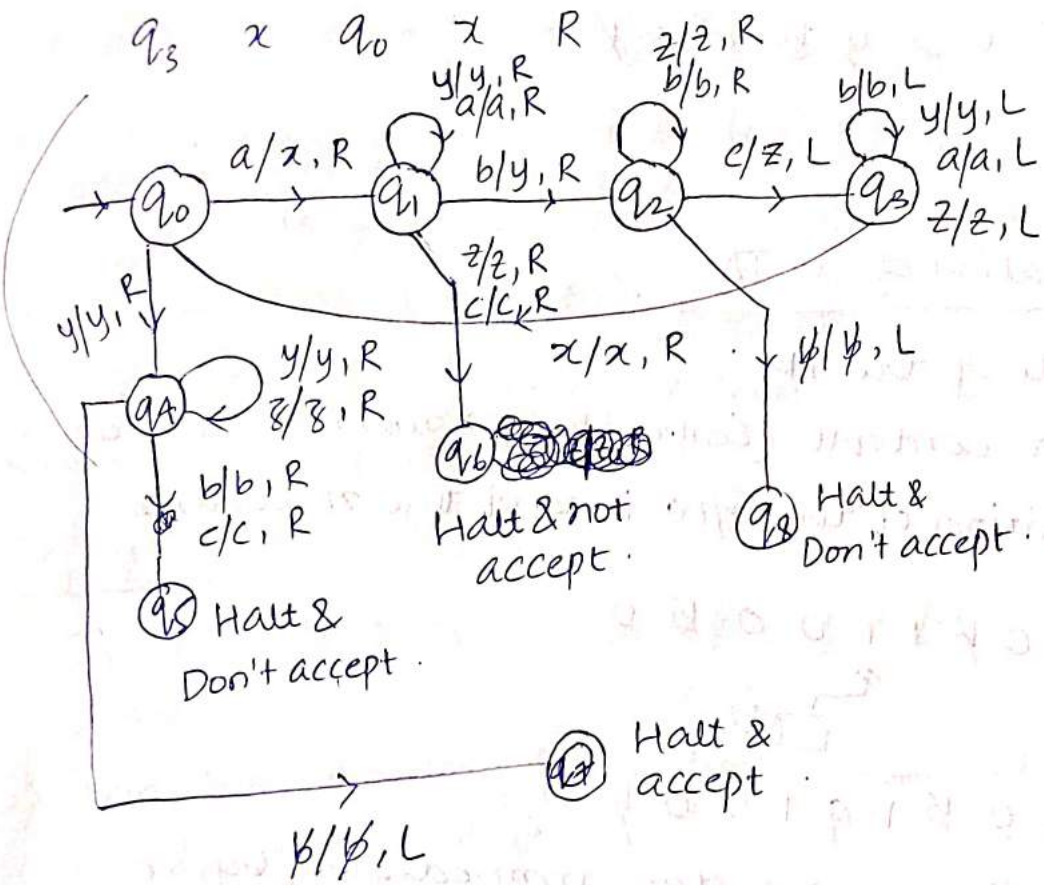
Q Σ - No. of possible tuples = $|Q| \times |\Sigma|$

If some state is not found, the machine halts (stops).

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

continued....

q_3	b	q_3	b	L
q_3	y	q_3	y	L
q_3	a	q_3	a	L
q_3	z	q_3	z	L



$Q = \{q_0, q_1, \ldots q_8\}$.

(Gamma) $\Gamma = \{a, b, c, x, y, z\} \cup \{ɸ\}$.

$\Sigma = \{a, b, c\}$.

δ

$ɸ \rightarrow$ blank

q_0

$F = \{q_7\} \subseteq Q$.

Tape symbols

Input alphabet.

$\Sigma \subseteq \Gamma$

Accepting Halt states

$TM = (Q, \Gamma, \Sigma, \delta, ɸ, q_0, F)$

$\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$.

$\{|ɸ|ɸ|a|a|b|b|c|c|ɸ|ɸ|\}$

Trace of Execution of a TM :

q_0 ɸ a a b b c c ɸ

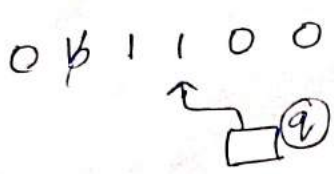
q_1 ɸ x a b b c c ɸ

q_1 ɸ x a b b c c ɸ

q_2 ⊔ x a y b c c ⊔

Configuration of a TM:

1. State of the TM
2. Tape content (Content btw left & right most blank elements)
3. Position of the r/w head of the machine.



$\Rightarrow$ 0 ⊔ 1 q 1 0 0 ⊔ 0.

Place the current state immediately before the r/w head pointing char.

We place the state of M/c before the symbol which is to be scanned.

Current state	Current Symbol	Next state	Symbol written	Move {L, R}
q	1	p	0	R

$C_1 \equiv$ 0 q 1 0 b 1

$C_2 \equiv$ 0 0 q 0 b 1

C_1 C_2 configuration.

- $C_1 \vdash C_2 \rightarrow$ Transitions involving 1 state
- $C_i \overset{*}{\vdash} C_j$ Transitions involving multiple states

$C_i \vdash C_{i+1} \vdash C_{i+2} \vdash \cdots \vdash C_j$

C_i – initial configuration.

* x q_0 p ∈ F

$(q_0 x)$ → Initial configuration

$q_0 x \overset{*}{\vdash} y p z$ (We can say that this thing exists)

$z, y \in T^*, p \in F$

$$L(M) = \{ x \in \Sigma^* \mid q_0 x \vdash^* ypz \; ; \; y, z \in \Gamma^*, p \in F \}$$

→ Recursively Enumerable languages (R.E):

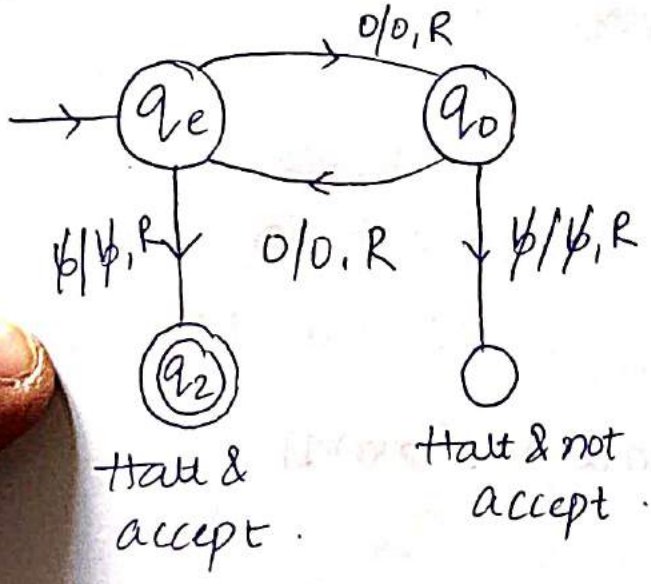
A lang. L is said to be R.E if there exists a TM M such that $L = L(M)$.

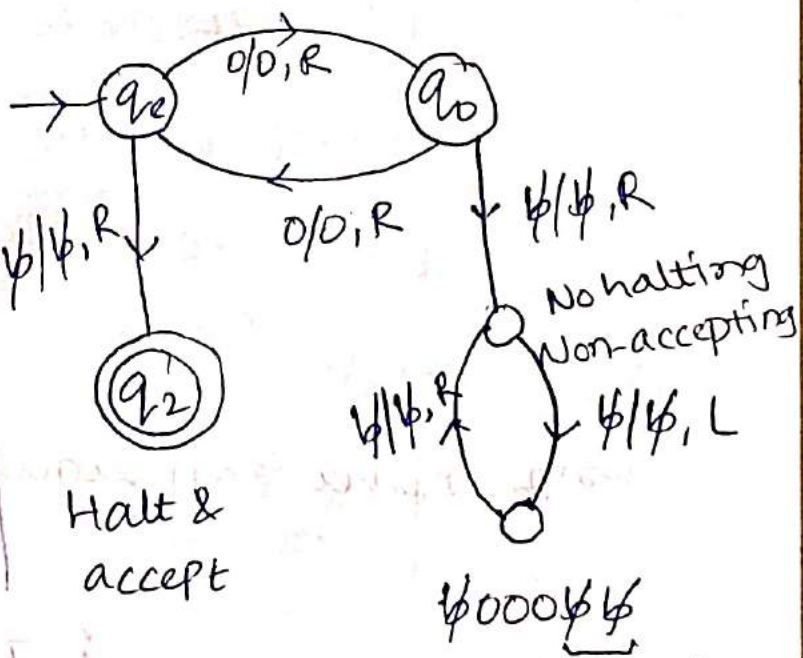
Accept: The machine reaches an accepting state and halts.

Rejection / Non-acceptance:

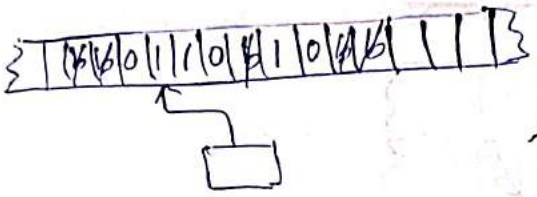
1) If the M/C reaches a non-acceptance state and halts.
2) The M/C continues to work without reaching any halting state.

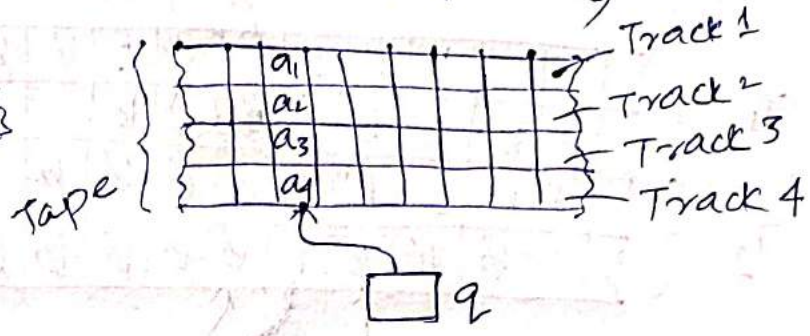
Ex: $L = \{ 0^i \mid i > 0 \text{ and } i \text{ is even} \}$.





Multi-track TM: (No power added in terms of solving)

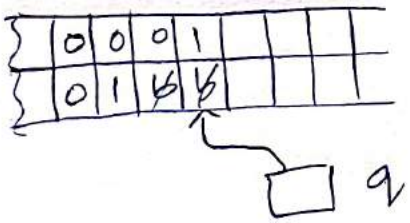




$q_1(a_1, a_2, a_3, a_4), p(a_1', a_2', a_3', a_4'), R$.

$|\Gamma|$ = finite $\quad \hat{\Gamma} = \{b \mid b \in \Gamma^4\}$.

$|\Gamma \times \Gamma \times \Gamma \times \Gamma|$ = finite.



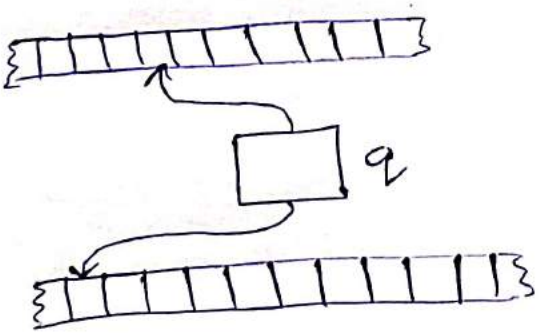
$\Gamma = \{0, 1, \not b\}$.

Possibilities: $00, 01, 0\not b$
$10, 11, 1\not b$
$\not b 0, \not b 1, \not b \not b$.

$q, (1, \not b), p, (0\ 0), L$.

q, f, p, a, L.

Multi-tape TM:

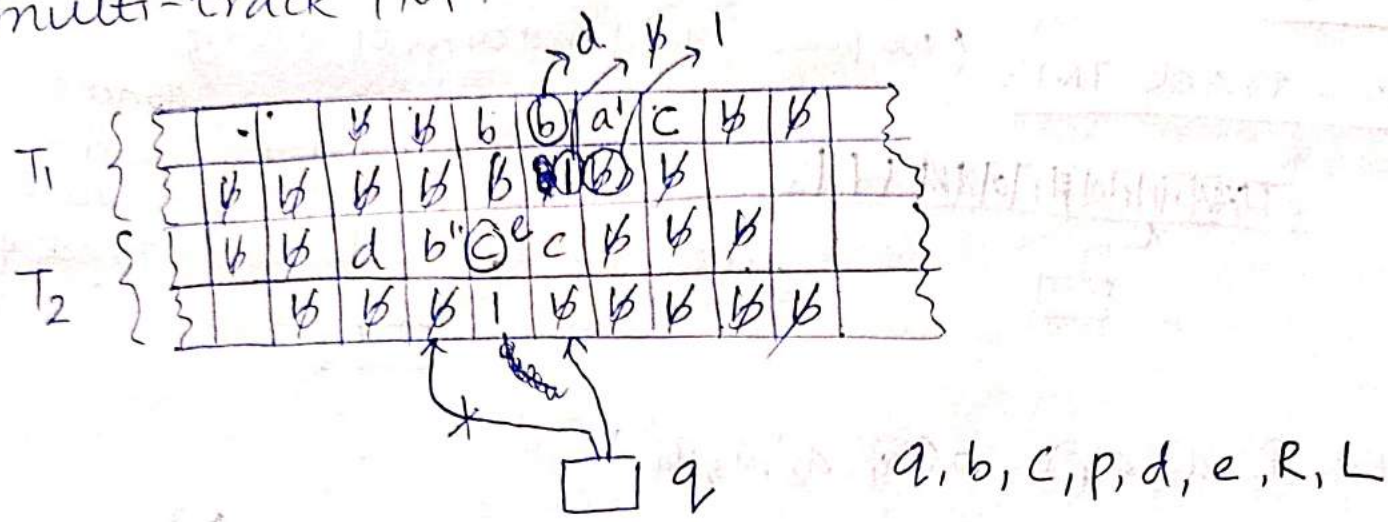


q, a, b, p, a', b', L, R

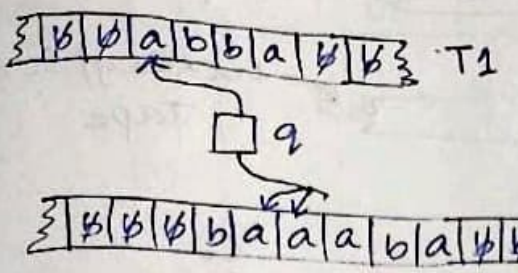
Simulate a single tape multi track TM using a single tape single track TM.

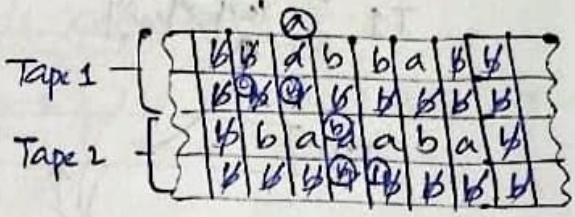
$MT \longrightarrow STMT \longrightarrow STST$.

A multi-tape TM can be simulated by a single tape multi-track TM.

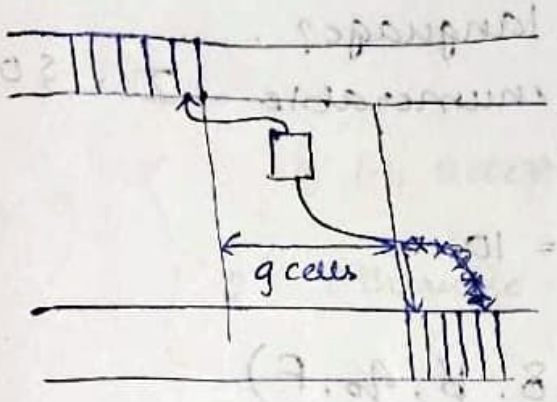


T. COMP





State	Symbol on T1	Symbol on T2	Next symbol on T_1	Next symbol on T2	Move T_1	Move T2	New State
q	a	a	a	b	L	R	P



$$q + (q+2) + (q+4) + \cdots (q+2n)$$
$$= (n+1)q + \underbrace{2+4+6+\cdots 2n}$$
$$= (n+1)q + O(n^2)$$

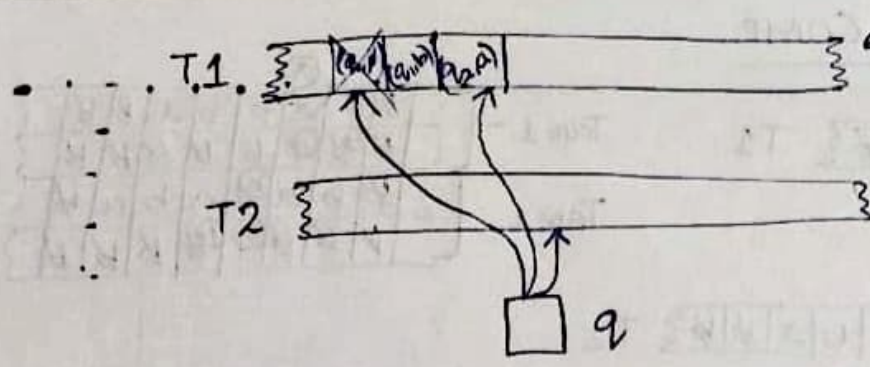
On conversion of MT to STMT
- computational time ↑ (but within poly. limits)
- computational capacity remains same.

Non-Deterministic TM:

(q, a, p, b, R)

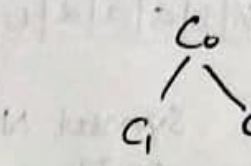
(q, a, p_1, b_1, R)
(q, a, p_2, b_2, L)
(q, a, p_3, b_3, R)

} Not a definite move
We have multiple choices



$(q_0, a) \rightarrow (q_1, b)$

$(q_0, a) \rightarrow (q_2, a)$



Add C_1, C_2 to queue and remove C_0

Is there any non-r.e language?

– The set of TM's is enumerable

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

$M \rightarrow L(M)$

$M \rightarrow 1010_b = 10$

$M = (Q, \Sigma, \Gamma, \delta, b, q_0, F)$

$q_i \; a_j \; q_k \; a_l \; L/R$

3 4 2 1 1

t_i = 1110 1111 011 0 1 0 1

Transitions are separated by "00"

δ = $t_1 00 t_2 00 t_3 00 t_4 00 t_5$

A TM can be represented by 0's & 1's.

"000" separates each of the components of the M/c

000 – "ill-formed"

– M accepts ϕ

Desc. of TMs ↓ / Bit strings →	0	1	00	01	10	11	000	001	...
0									
1									
00									
01									
i → 10									
11									
000									
001									
...									

↓ j

$L_d = \{ x \in \Sigma^* \mid$ The TM whose code is x doesn't accept $x \}$

$$a_{ij} = \begin{cases} 1 & \text{if } M_i \text{ accepts } j \\ 0 & \text{otherwise.} \end{cases}$$

Is L_d r.e ?

Suppose L_d is r.e, then there must be a TM to accept L_d. Let M_x be the TM accepting L_0.

(x_x)

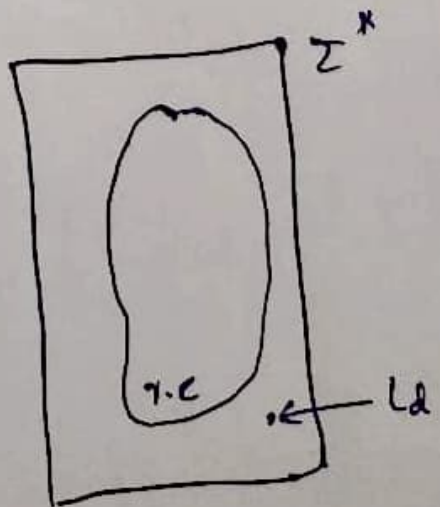
$L_0 = \{ x \in \Sigma^* \mid$ The TM encoded by x doesn't accept $x \}$

$x_x \in L_d \longrightarrow x_x \notin L(M_x)$

$L(M_x) = L_N$

(OR) $x_x \notin L_d$

$x_x \in L(M_x)$



Recursive languages:

A language L is recursive if there exists a TM that

1) accepts L
2) halts on every input.

We say that a language L is r.e if ∃ a TM that
① Accepts L.

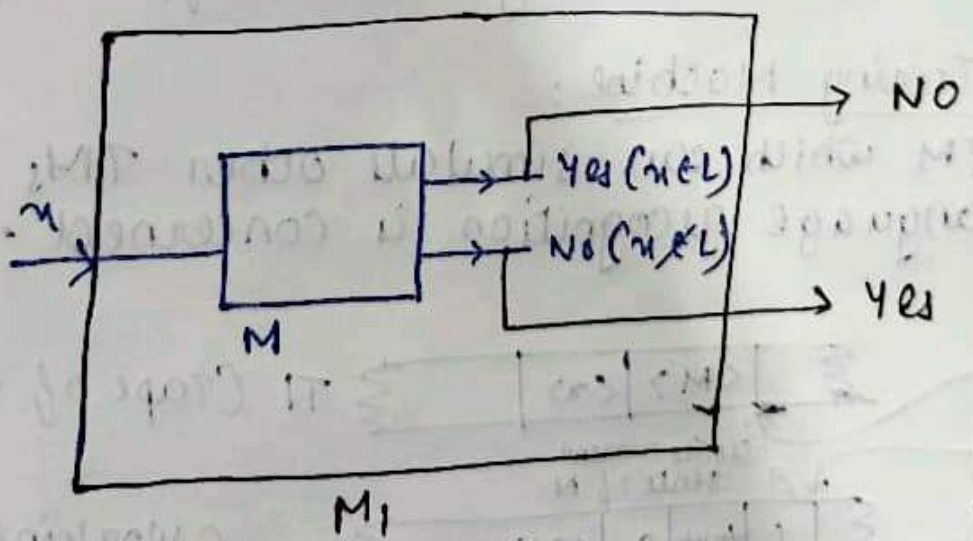
We say that a language L is recursive if ∃ a TM that
① Accepts L.
② Halts on every input

* If there is an algorithm that decides the membership decision problem of a set A then that problem is called decidable iff the set A is recursive.

① Are all r.e languages recursive?
Recursive languages are definitely r.e.

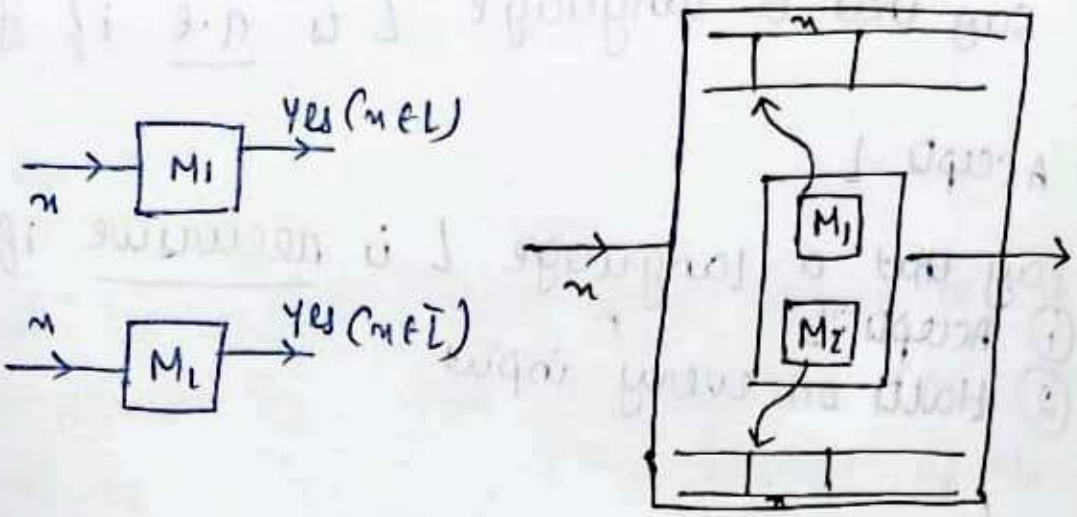
① if L is recursive, then so is $\bar{L}$
② if both L and $\bar{L}$ are r.e then L is recursive.

①



M accepts L
M_1 accepts $\bar{L}$

(2)



(*) Are all r.e languages recursive?

There are languages that are r.e but not recursive

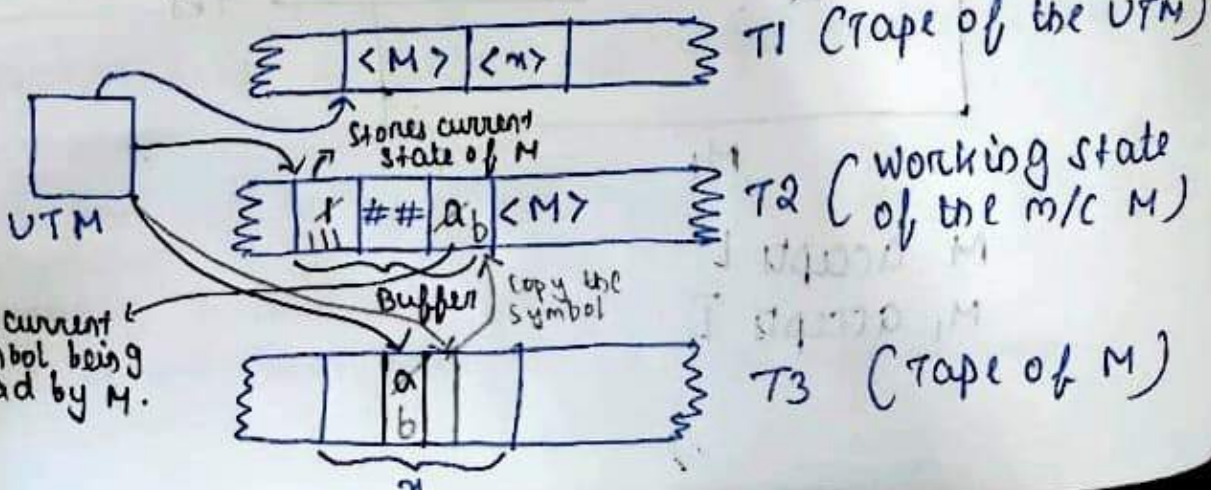
$$L_u = \{ \langle M, x \rangle / M \text{ accepts } x \}$$

* Is L_u r.e?
* Is L_u recursive?

We design a Universal Turing Machine for this.

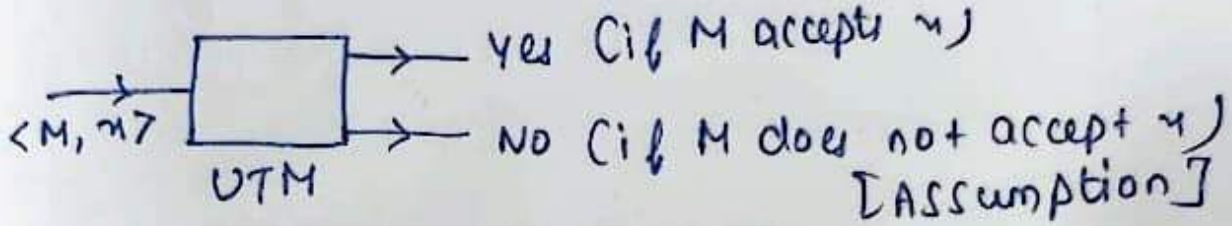
Universal Turing Machine:

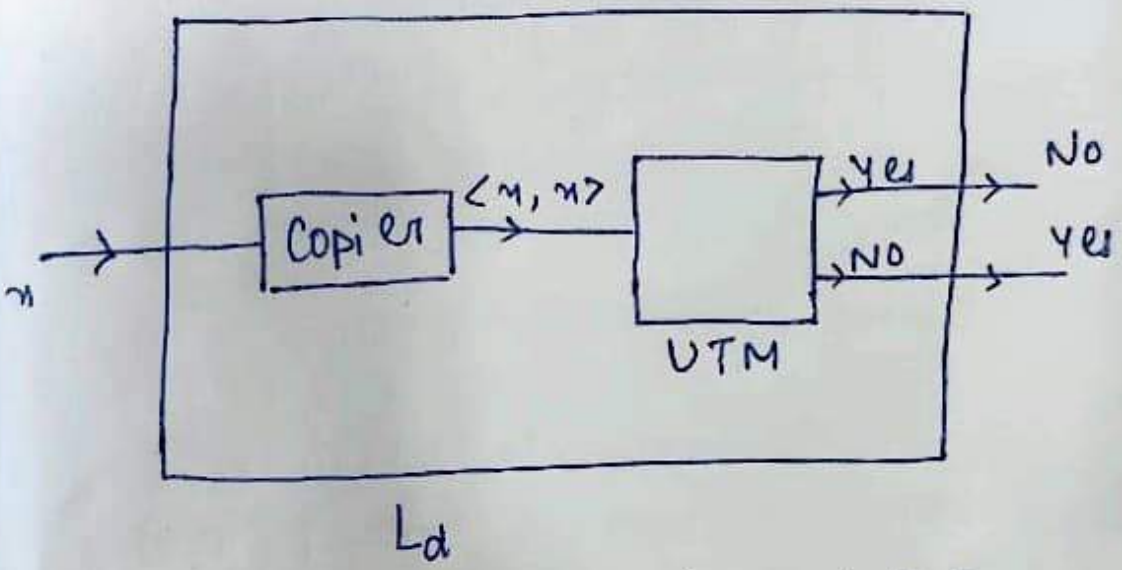
A TM which can simulate other TMs so far as language recognition is concerned.



If M accepts x, then the UTM accepts <M, x>.
Since we can build a machine that accepts Lu,
so Lu is r.e.

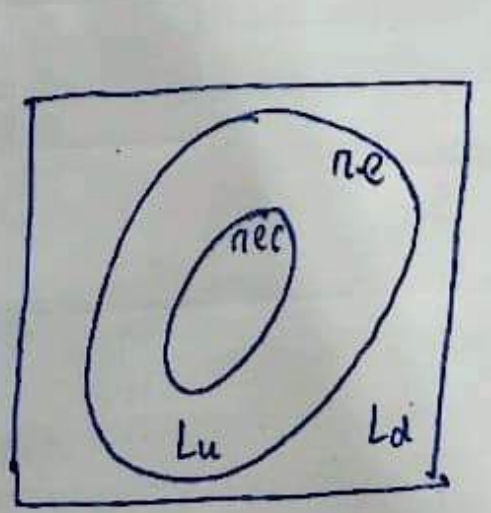
Now is Lu recursive ?





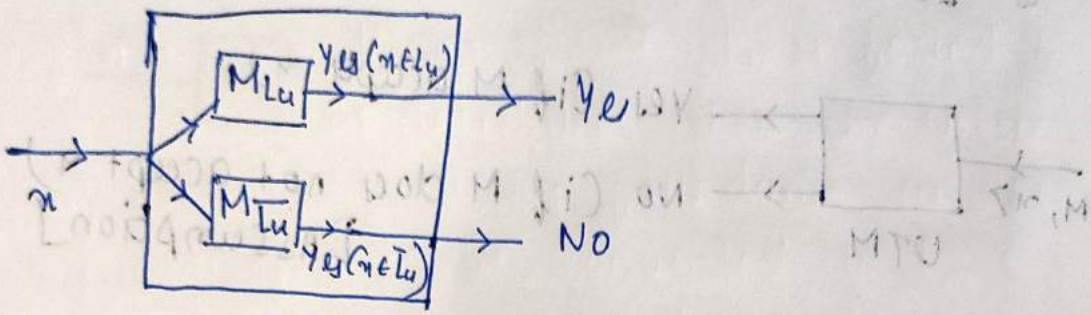
L_d

But we know L_d is not r.e
so our assumption is wrong.



(*) L_u is r.e

Let $\bar{L}_u$ is also r.e



M

This machine M says Yes if $x \in L_u$

No if $x \in \bar{L}_u$

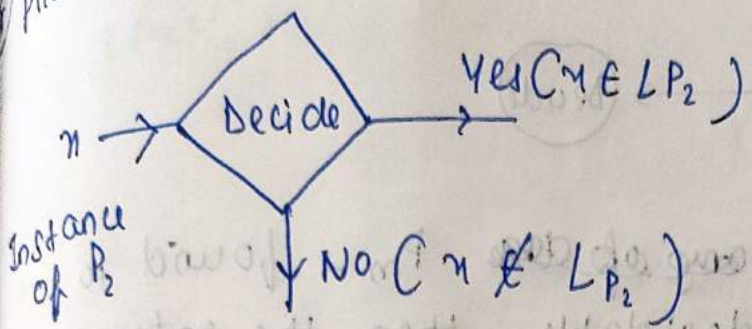
So M always halts, so L_u turns out to be recursive, but we know L_u is not rec., so our assumption is wrong. $\therefore \bar{L}_u$ is not r.e.

(#) Reduction

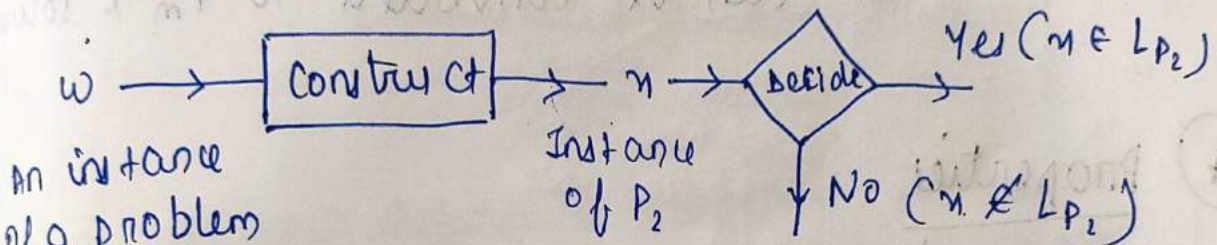
Problem P $\quad \Sigma^* \quad x \in \Sigma^* \quad \langle x, \text{Yes} / \text{NO} \rangle$

$L_P \quad x \in \Sigma^*$, for which the answer is 'Yes' in terms of problem P.

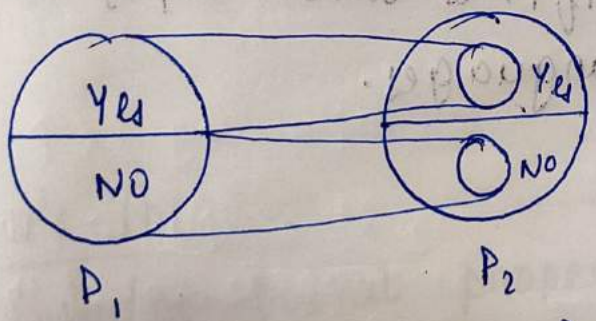
Problem P_2 (unknown)



P_1 is an undecidable problem.



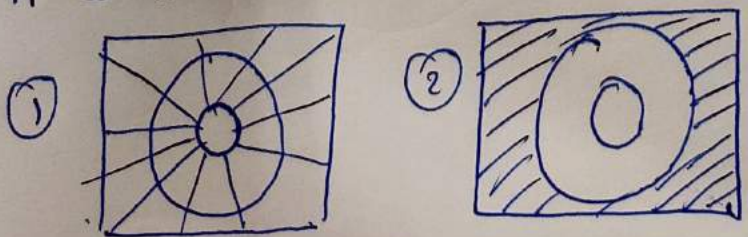
If this construct exists, then we have a decider for P_1, but we know P_1 is undecidable, so such construct doesn't exist. If a construct exists so that $w \to x$, then x is also undecidable.



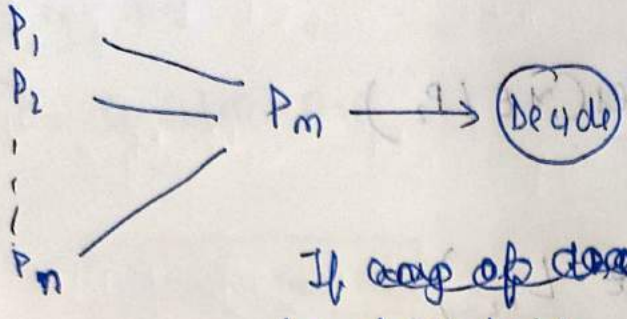
If there is a reduction from P_1 to P_2

① If P_1 is undecidable, then so is P_2.

② If P_1 is non-re then so is P_2



*



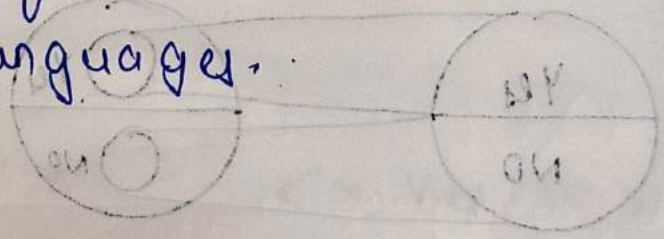
If ~~any of these~~ P_m is found to be decidable, then the entire hierarchy falls down, as others can be converted to P_m & solved.

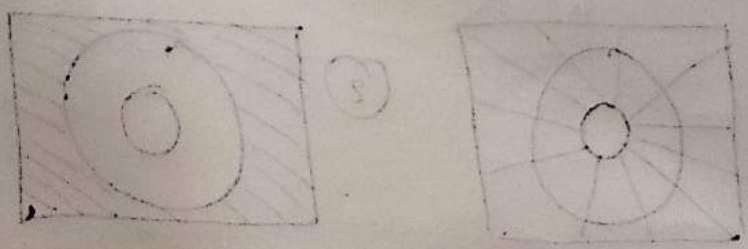
(*) Properties

- Trivial (Properties followed by all languages)
- Non-Trivial (Properties followed by some subset of language)

A property is trivial if it is either empty or satisfied by all r.e languages.

Otherwise Non-trivial.





Turing Machine / Property	M_1	M_2	M_3	M_n	M_4
	$L(M_1)$	$L(M_2)$	$L(M_3)$	$L(M_n)$	$L(M_4)$
P_1					
P_2					
P_3	✓	×	✓	✓	×
⋮					
P_n					

Thus we can now define a property P as

P → subset of the class of r.e languages whose members satisfies P.

$P_3 = \{ L(M_1), L(M_3), \ldots L(M_n), \ldots \}$

$L_{P_3} = \{ M_1, M_3, \ldots M_n, \ldots \}$ → Here we use binary encodings/desc of TMs which is finite

(*) Rice's Theorem

Every non-trivial property of a r.e language is undecidable.

L_P → language of property P

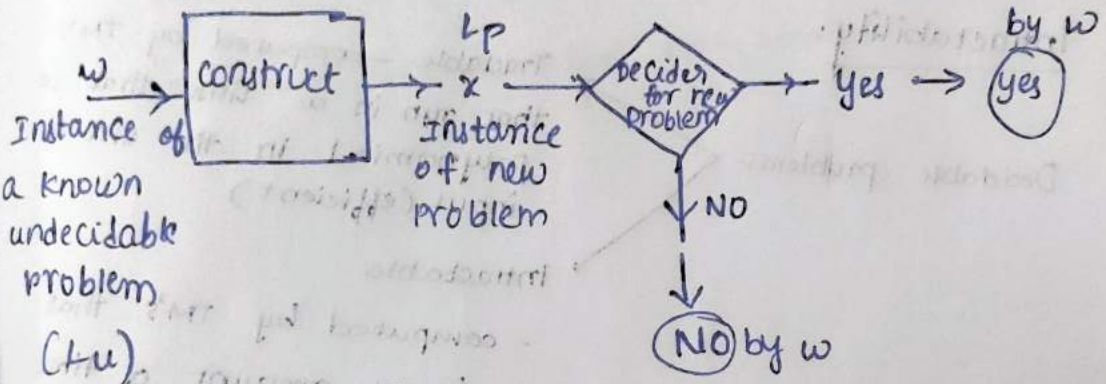
Reduce $L_u \longleftrightarrow L_P$

Rice's theorem: ~~All~~ every non-trivial properties of a R.E language are undecidable

$P = \{L_1, L_2, \dots L_k, \dots\}$ $\quad L_i \in P$

$P = \{M_1, M_2, \dots M_k, \dots\}$ $\quad L(M_i) = L_i$

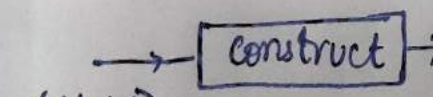
$L_i = L(M_i)$



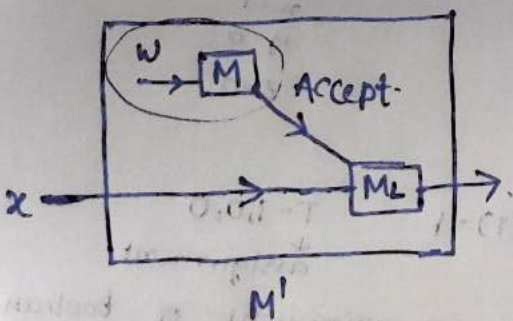
$L_u = \{ \langle M, w \rangle \mid M \text{ accepts } w \}$

$L_p = \{ M \mid L(M) \text{ satisfies } P \}$

→ we are assuming empty set is not in those sets.



ϕ (empty set) $\notin P$

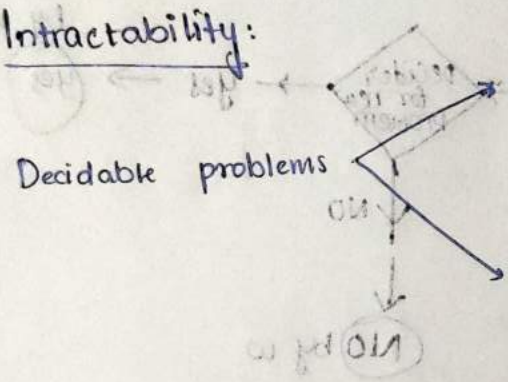


→ if ϕ is in L_p then you construct for $\bar{P}$

By this construction, we try to make the same machine two diff ways.

If we don't know whether a problem is undecidable problem or not. If we can deduce a construct to the new problem from an undecidable problem. We can prove that the new problem is also undecidable.

→ Intractability:



→ Satisfiability Problem:

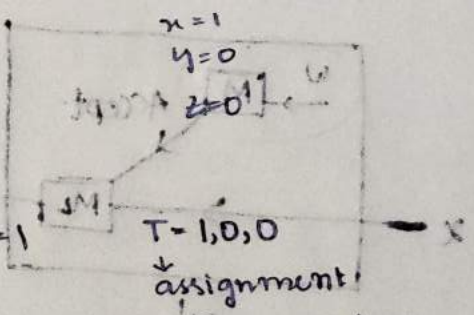
Boolean expression satisfiability problem:

$E = x \wedge \neg(y \vee z)$

$E = 1 \wedge \neg(0 \vee 0)$

$= 1 \wedge \neg 0$

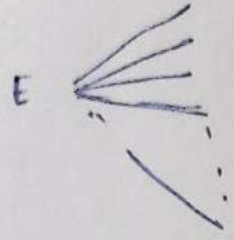
$= 1 \wedge 1 = 1$



A truth assignment is an assignment of boolean values to the variables.

Satisfiable: for at least 1 variable assignment if E is 1

if we can have non-deterministic TM



verifies
~~solves~~ each case in
8 possibilities → polynomial time

solved by non-deterministic in exponential time
verifies in polynomial time

NP → non-deterministic polynomial.

NP problems : The class of problems that can be solved by Non-deterministic turing machines (NDTM) operating in polynomial time.

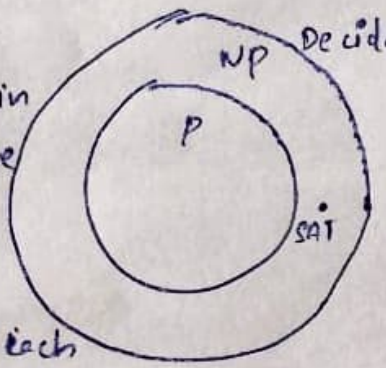
P problems : The class of problems that can solved by deterministic turing machines operating in polynomial time.

A turing machine is said to be of time complexity $T(n)$ [running time $T(n)$] if M is given an input 'x' of length 'n', M halts after making atmost $T(n)$ moves, regardless of whether M accepts.

Non-deterministic Polynomial time:

A language L is in NP if there is a NDTM M and a polynomial time complexity $T(n)$ such that $L = L(M)$ and when M is given an input of length n, there is no sequence of more than $T(n)$ moves of M.

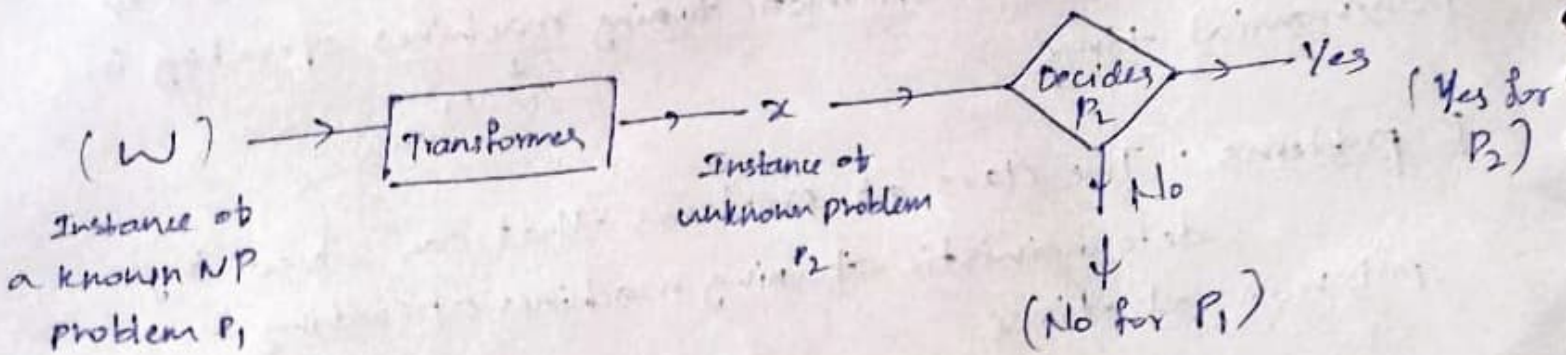
P ⊆ NP: The reason is that a NDTM running in polynomial time has the ability to guess an exponential no. of solutions and check each one of them in polynomial time.



$P \subseteq NP$

$P \neq NP$

Polynomial time reduction:



Let L be a language corresponding to a problem in NP.

(1) L is NP complete if the following statements are TRUE about L:

(a) L is in NP

(b) For every language L in NP, there is a polynomial time reduction algorithm of L' to L.