FORMAL METHODS

$A \rightarrow$ set

Finite set $\rightarrow |A| < \infty$

$A, B \Rightarrow A \times B = \{(x,y) / \forall x \in A, \forall y \in B\}$

↓

all possible combinations of ordered tuples

$A^K = A \times A \times A \ldots \times A$; $K \in N$ (K times)

$A = \{a\}$ $B = \{b\}$

$C = \{c\}$

$A \times B \times C = \{(a,b,c)\}$

$|A| = m \ (< \infty)$

$|B| = n \ (< \infty)$ $\Big\}$ $|A \times B| = m.n$

Alphabet $\rightarrow$ finite set of symbols

Σ

Suppose $\rightarrow \Sigma = \{0,1\} \equiv$ strings: 0, 1, 01, 10, 11, 001, ...

lexicographic order – dictionary order

Short lexicographic order – same as dictionary order but short strings come first. (Length is also considered)

$\bar{A} \rightarrow$ complement of $A \equiv \{a \in U / a \notin A\}$

$\overline{A \cup B} = \bar{A} \cap \bar{B}$

If two sets are equal, $A = B$, then $A \subseteq B$ & $B \subseteq A$

$\overline{A \cup B} = \bar{A} \cap \bar{B}$ — (i) $\overline{A \cup B} \subseteq \bar{A} \cap \bar{B}$

(ii) $\bar{A} \cap \bar{B} \subseteq \overline{A \cup B}$

Let $x \in \overline{A \cup B}$

$\Leftrightarrow x \notin A \cup B$

$\Leftrightarrow x \notin A$ and $x \notin B$

$\Leftrightarrow x \in \bar{A}$ and $x \in \bar{B}$

$\Leftrightarrow x \in \bar{A} \cap \bar{B}$

$G = (V, E)$

$v \in V$, $\deg(v)$

$\sum_i \deg v_i = 2(\#E)$

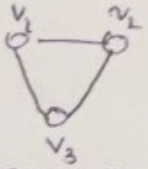
Proof Techniques:

→ Proof by construction:

Defn:

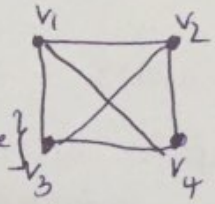
$K \in \mathbb{N}$, K-Regular graph $G(V, E)$

$\forall\, v \in V$; $\deg(v) = K$



$\deg(v_1) = 2$
$\deg(v_2) = 2$
$\deg(v_3) = 2$

$\forall K \geq 2$, $\exists$ a 3-regular graph $G(V, E)$ with $|V| = 2K$, $\forall K \in \mathbb{N}$



$\deg(v) = 3$

For a vertex set, V

$V = \{ v_1, v_2, v_3 \ldots v_{2K-1}, v_{2K} \}$

we need to construct an edge set such that the degree of the vertices = 3.

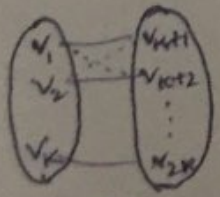
$$E = \{ (v_i, v_{i+1}) \mid i \in \{1, 2 \ldots 2K-1\} \} \cup \{ v_{2K}, v_1 \}$$

contribution = 2.

$$\cup \{ (v_1, v_{K+1}), (v_2, v_{K+2}), \ldots \ldots (v_K, v_{2K}) \}$$

mapping can be anything

contribution = 1



mapping can be anything but make sure the contributions match up

→ Proof by Contradiction:

Ex $\sqrt{2}$ is irrational

Suppose
According to proof by contradiction; $\sqrt{2}$ is rational

$$\sqrt{2} = \frac{p}{q}, \quad q \neq 0, \; p, q \in \mathbb{Z}$$

$\frac{1}{2} = \frac{2}{4} = \frac{3}{6} = \frac{10}{20}$

GCD(1,2) = 1.

GCD(p,q) = 1.

$$(\sqrt{2})^2 = \frac{p^2}{q^2}$$

$p^2 = 2q^2 \Rightarrow p^2$ is even.

Since p^2 is even; p must be even.

Since p is even;

$p = 2K$

$\Rightarrow (2K)^2 = 2q^2$

$4K^2 = 2q^2$

$q^2 = 2K^2 \Rightarrow q^2$ is even.

Since q^2 is even, q must be even.

Since p is even and q is even $\Rightarrow$ GCD(p,q) is atleast greater than or equal to 2

→ So, the assumption ($\sqrt{2}$ is rational) is wrong.

→ Proof by Induction:

① we prove that p(n) is true

② If p(K) is true (after any $K(>m \in \mathbb{N})$) then we show that p(K+1) is true.

Σ be an alphabet ;

Σ^* is the set of all string over Σ

Σ^n is the set of all string of length n

u, v over Σ

$u = u_1 u_2 u_3 \ldots u_m$; $|u| = m$

$v = v_1 v_2 v_3 \ldots v_n$; $|v| = n$

$uv = u_1 u_2 \ldots u_m v_1 \ldots v_n$; $|uv| = m+n$

Deterministic Finite Automaton –

A deterministic finite automaton (DFA) is a 5-tuple $(\Sigma, Q, \delta, q_0, F)$ where

Σ - alphabet

Q – set of states

q_0 – Initial / Start state

F – set of Final / Accept state

δ - transition function

$\delta : Q \times \Sigma \longrightarrow Q$ (Transition function).

$\Rightarrow \hat{\delta} = Q \times \Sigma^* \longrightarrow Q$ — Accepting a String

$u = u_1 u_2 u_3 \ldots u_n$ [String over Σ]

States : $r_0 \xrightarrow{u_1} r_1 \xrightarrow{u_2} r_2 \xrightarrow{u_3} \ldots \ldots r_{n-1} \xrightarrow{u_n} r_n$

These states need not be distinct, they can be repeating also.

Accepts only when
$\Rightarrow r_n \in$ Final State
$\Rightarrow q_0 = r_0$ (Initial State)

Transition function – $\delta(r_i, u_{i+1}) = r_{i+1}$

→ $w = uv$

$$\hat{\delta}(q, w) = \hat{\delta}(q, uv) = \hat{\delta}(\hat{\delta}(q, u), v)$$

→ $u = u_1 u_2 \ldots u_k$

$u \in \Sigma^*$

$\exists\ r_0, r_1, r_2 \ldots r_k \in Q$.

$\hat{\delta}(q_0, u) = q \in F$

(1) $q_0 = r_0$

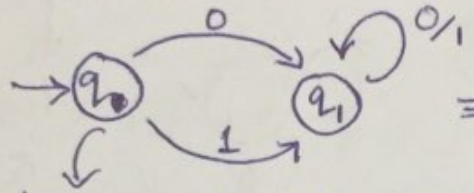
(2) $r_k \in F$

(3) $\delta(r_{k-1}, u_k) = r_k$.

$r_0 \xrightarrow{u_1} r_1 \xrightarrow{u_2} r_2 \ldots \xrightarrow{u_k} r_k$

$\delta(r_i, u_{i+1}) = r_{i+1}$ $\forall\ i \in \{0, \ldots u-1\}$

→



≡ Language ≡ ϕ = {Empty string}

for any symbol, this FSM goes to state q_1 except for an empty string.

[cannot hold arbitrary lengths of strings fixed memory]

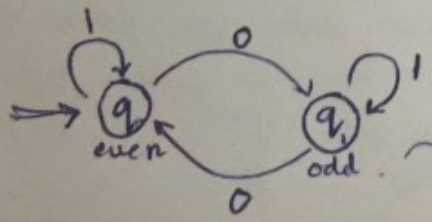
Regular Languages – Language L, if some DFA recognizes it.

Ex: $u = \{0, 1\}$, such that u has even number of 0s

↓ Language.

Do we need to remember the entire string

No, just the count of 0s till the symbol read – even or odd [2 states]



also contains empty string

↓ as it contains zero number of zeroes

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

$Q = \{q_{even}, q_{odd}\}$

$q_0 = q_{even}$ (empty string has even no. of zeroes)

$F = \{q_{even}\}$

→ $u, v \in \Sigma^*$

u is the substring of 'v' if there exists two strings x and y in Σ^* such that $v = xuy$.

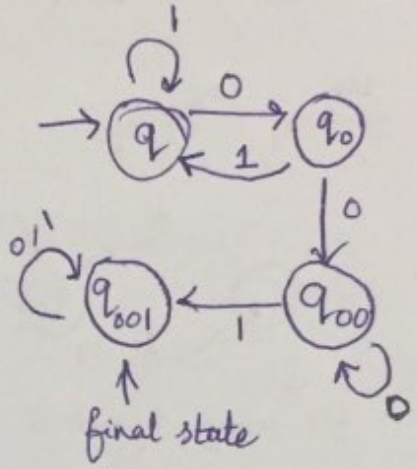
For Ex: DFA? which recognizes language of all strings $\{0,1\}$ that contain the string 001 as a substring

Since we cannot hold the string (symbols of) read till now., we need to remember —

- have not seen a single 0.
- have seen just a 0.
- have seen two zeroes
- have seen the substring in whole

account to.

four states ✓



→ $\Sigma = \{0,1\}$

$L_k = \{ w \in \Sigma^k \mid k^{th}$ character of w from the end $= 1 \}$

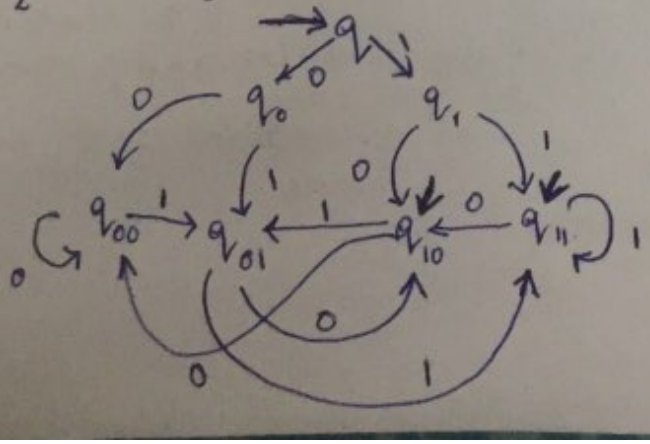
$L_1 = \{1, 01, 11 \ldots\}$.

$L_2 = \{10, 11, 010, 110, \ldots\}$ → infinite sets.

$L_3 = \{100, 101, 110, 111, 0111, \ldots\}$

remember k symbols. [new state for each string of length less than k]

L_2 → strings with length less than "2". {0, 1, 00, 01, 10, 11}



{initial state = q}

"q_{10}, q_{11} – final states"

$Q = \{ \langle w \rangle \mid |w| \leq k, w \in \Sigma^* \}$

states as strings : <0>, <1>, <00>, <01>

Transition function ≡

$$\delta(\langle \omega \rangle, a) = \begin{cases} \langle \omega a \rangle ; & \text{if } |\omega| < K \\ \omega_2 \ldots \omega_K a ; & \text{if } |\omega| = K \end{cases}$$

$\omega = \omega_1, \omega_2 \ldots \omega_K$

L_1 & L_2 are two regular languages over Σ, so

→ is $L_1 \cup L_2$ regular?

$L_i \rightarrow M_i(\Sigma, Q_i, \delta_i, q_i, F)$ for $i = 1, 2$

$$L_1 \cup L_2 = \{ u \in \Sigma^* \mid u \in L_1 \text{ or } u \in L_2 \}$$

$\delta_1 : Q_1 \times \Sigma \rightarrow Q_1$

$\delta_2 : Q_2 \times \Sigma \rightarrow Q_2$

$M \rightarrow L_1 \cup L_2 (\Sigma, Q, \delta, q, F)$

DFA : M ≡ $Q = Q_1 \times Q_2$

Input ≡ $\delta : (Q_1 \times Q_2) \times \Sigma \rightarrow Q_1 \times Q_2$

→ what if ? $Q = Q_1 \cup Q_2$

Suppose $q \in Q_2$, $\delta_1(q_1, a) \rightarrow$ doesn't apply

Transition function:

$$\delta((r_1, r_2), a) = (\delta_1(r_1, a), \delta_2(r_2, a)),$$

$q \in Q_1 \times Q_2$

$q = (q_1, q_2)$

$q \rightarrow q_1$, $q \rightarrow q_2$ ⇒ $(q_1, q_2) \in Q_1 \times Q_2$

$\notin Q_1 \cup Q_2$

final state :

$$F = \{ (r_1, r_2) \mid r_1 \in F_1 \text{ or } r_2 \in F_2 \}$$

Brief Revision – $\equiv ((F_1 \times Q_1) \cup (F_2 \times Q_2))$

DFA – 5 tuple

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

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

$(q, a) \rightarrow \delta(q, a)$

NFA – 5 tuple

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

domain = $Q \times \Sigma_\epsilon$ including epsilon

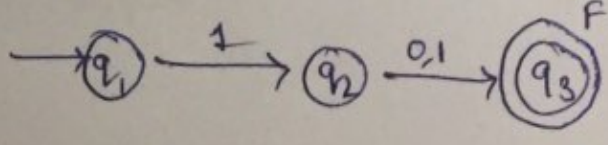
$\delta : Q \times \Sigma_\epsilon \rightarrow P(Q)$

$\phi \subseteq Q$

$\delta(q, a) = \phi$ → no further computation is allowed

output ≡ "sets"

For Ex: Consider →



This is not a DFA, as $(q_1, 0) \rightarrow$ undefined.

NFA? Yes.

	ϵ	0	1
q_1	$\{q_1\}$	ϕ	$\{q_2\}$
q_2	$\{q_2\}$	$\{q_3\}$	$\{q_3\}$
q_3	$\{q_3\}$	ϕ	ϕ

final state → on reading 0, 1 → its not defined. $\equiv \phi$

Regular operations:

U – union

o – concatenation

* – star

A – same language.

$$A^* = \{u_1, u_2 \ldots u_k \mid k \geq 0, u_k \in A\}$$

L_1 and L_2, two regular languages over Σ,

→ $L_1 \cup L_2$ is closed; i.e, also regular.

→ $L_1 \circ L_2$ (L_1 concatenated with L_2), is it regular?

↳ $Q = Q_1 \cup Q_2$

Theorem :- for any given NFA, ∃ a self DFA

$N(\Sigma, Q, \delta, q, F)$

$\delta : Q \times \Sigma_\epsilon \rightarrow P(Q)$ → gives the set of states as output.

$|Q| = K$

$P(Q) = 2^K$

Whereas; DFA $\equiv M(\Sigma, Q', \delta', q', F')$

$Q' = P(Q)$, $q' = \{q\}$

$\delta' \equiv P(Q) \times \Sigma \longrightarrow P(Q)$

A = inputset

$\delta'(A, a)$

from each state, read the symbol

$A = \{r_0, r_1 \cdots r_n\}$

Without ϵ transitions

$= \bigcup_{r \in A} \delta(r, a)$

machine can be in any of these set of states

$\delta(r_0, a)$
$\delta(r_1, a)$
:
$\delta(r_n, a)$.

What if there's ϵ ?

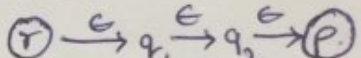
state

symbols in the alphabet. $\equiv$ including epsilon transitions

for each $r \in Q$;

$E(r) = \{p \in Q \mid$ p can be reached from r by moving towards along ϵ arrows$\}$

set of states.

$\delta \equiv$ 

one state with only epsilon transitions

for each $R \in Q$;

$E(R) = \bigcup_{r \in R} E(r)$

on traversing zero or more ϵ arrows

Including ϵ transitions;

NFA - δ

DFA - δ'

$$A = \bigcup_{r \in A} E(\delta(r, a))$$

$\delta' : P(Q) \times \Sigma \to P(Q)$

$\delta'(R, a) = \bigcup_{r \in R} E(\delta(r, a))$

$$F' = \left\{ A \subseteq Q \mid \text{A should contain at least one final state of NFA, N} \right\}$$

$$\left\{ A \subseteq Q \mid A \cap F \neq \phi \right\}$$

$R = \{r_1, r_2, \ldots r_n\}$

$\delta(r_1, a)$ $\delta(r_n, a)$

$\delta(r_2, a)$

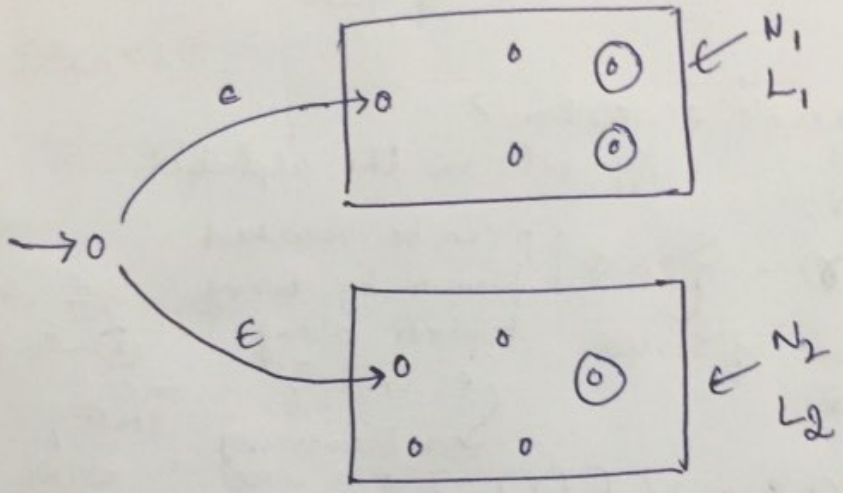
Other definition of regular languages :-

If it can be recognized by an NFA,

$Q = Q_1 \cup Q_2 \cup \{q_0\}$

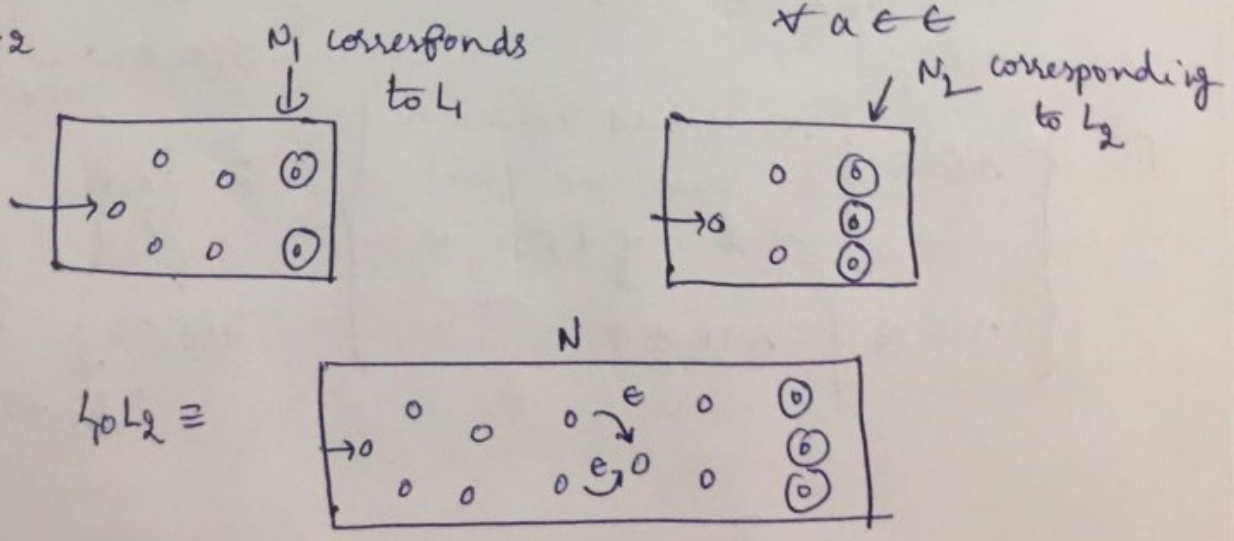
$\delta : Q_1 \times \Sigma_i \rightarrow P(Q)$

$$\delta(q,a) = \begin{cases} \delta_1(q_1,a) & \text{if } q \in Q_1 \\ \delta_2(q_2,a) & \text{if } q \in Q_2 \end{cases}$$



$$= \begin{cases} (q_1, q_0) & \text{, if } q = q_1, \forall\, a \in \epsilon \\ \phi & \text{, if } q \\ & \forall\, a \in \epsilon \end{cases}$$

$L_1 \circ L_2$



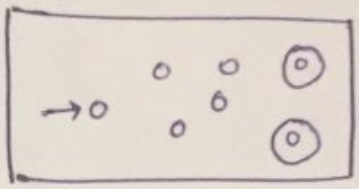
$L_1 \circ L_2 \equiv$

Star operation →

A^* , A → regular language → machine N corresponds to language A

$$A^* = \{ w_0 w_1 w_2 \cdots w_n \mid w_i \in A, i \geq 0 \}$$

break down such that each piece belongs to language A



Ex:

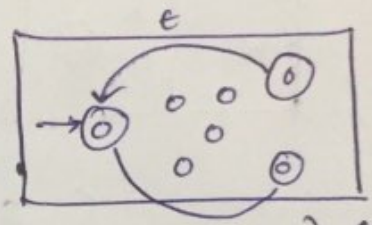
$A \equiv N_1 (\Sigma, Q_1, \delta, q, F_1)$

$B \equiv N_2 (\Sigma, Q, \delta_2, q_2, F_2)$

$A \cup B \equiv N$

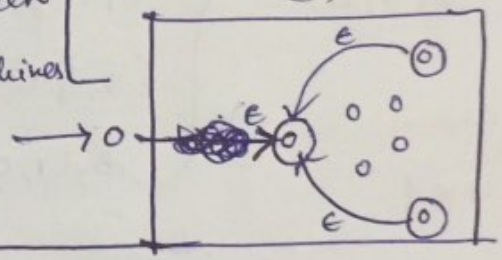
→ NFA : $A \cup B$ ← closed

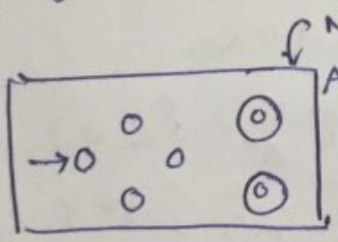
A, B are two regular languages →

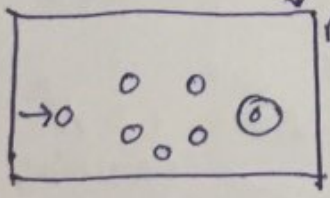


difference between these machines

ε (vs) actual way of doing it :)







Constructing NFA

$A \cup B \equiv A - N_1 (\Sigma, Q_1, \delta_1, q_1, F_1)$

$B - N_2 (\Sigma, Q, \delta_2, q_2, F_2)$

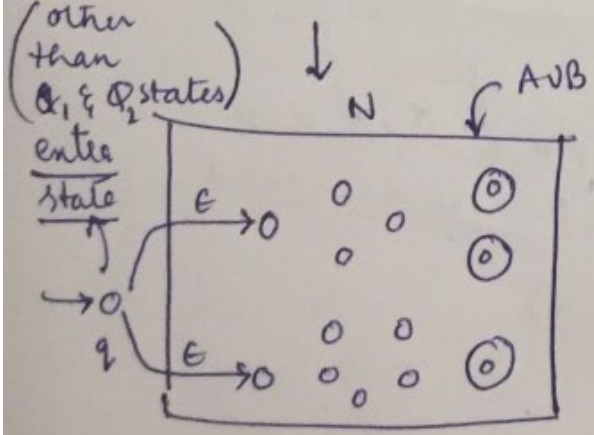
$A \cup B \equiv N (\Sigma, Q, \delta, q, F)$

$Q = Q_1 \cup Q_2 \cup \{q\}$ ← extra state

$\delta : Q \times \Sigma_\varepsilon \rightarrow \mathcal{P}(Q)$

$$\delta(r,a) = \begin{cases} \delta_1(r_s, a) & \text{if } r_s \in Q_1 \\ \delta_2(r_s, a) & \text{if } r_s \in Q_2 \\ \{q_1, q_2\} & \text{if } r = q, a = \varepsilon \\ \phi & \text{if } r = q, a \neq \varepsilon \end{cases}$$

(other than Q_1 & Q_2 states) extra state



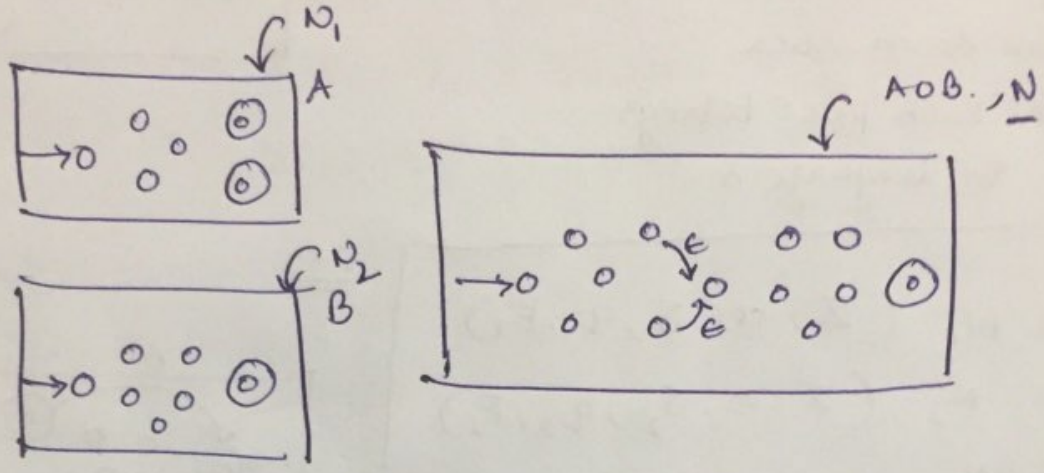
AoB → NFA ⇒ $Q = Q_1 \cup Q_2$

↓ closed

$F = F_2$

$q = q_1$

$\delta: Q \times \Sigma_\epsilon \to P(Q)$.



$$\delta(r,a) = \begin{cases} \delta_1(r_\bullet, a) & \text{if } r_\bullet \in Q_1/F_1 \\ \delta_1(r_\bullet, a) & \text{if } r_\bullet \in F_1 \text{ and } a \neq \epsilon \\ \{q_2\} \cup \{\delta_2(r,a)\} & \text{if } r \in F_1 \text{ and } a = \epsilon \\ \delta_2(r,a) & \text{if } r \in Q_2. \end{cases}$$

↓ closed

$A^* \Rightarrow N'(\Sigma, Q', \delta, q', F')$

$A \to N(\Sigma, Q, \delta, q, F)$

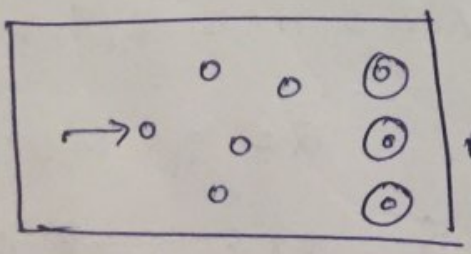
belongs to A^* ← A^i, $i \geq 0$.

$$A^i = \begin{cases} \{\epsilon\}, & \text{if } i = 0 \\ A \circ A^{i-1} & \text{if } i \geq 1 \end{cases}$$

↓

$$A^* = \bigcup_{i \in N \cup \{0\}} A^i.$$

A →



q' → new state that is introduced.

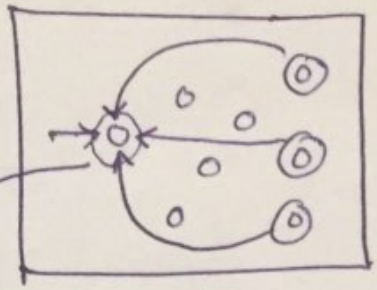
$Q' = Q \cup \{q'\}$ $F' = F \cup \{q'\}$

$\delta' = Q' \times \Sigma_\epsilon \to R(Q)$

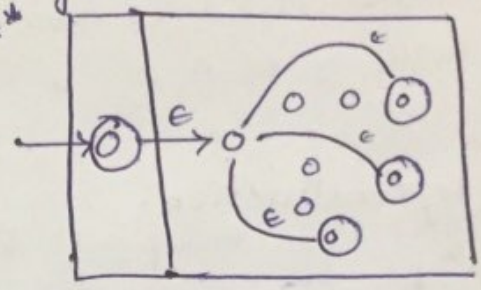
$$\delta'(r,a) = \begin{cases} \delta(r,a), & r \in Q/F \\ \delta(r,a), & r \in f, a \neq \epsilon \\ \delta(r,a) \cup \{q\}, & r \in F, a = \epsilon \end{cases}$$

No extra transitions possible.

(since the stent state is the final state) in this case it accepts strings outside A^* ↓ so ↓



$$\begin{cases} \phi, & r = q', a \neq \epsilon \\ \{q\}, & r = q', a = \epsilon \end{cases}$$



Regular Expressions:

Regular operations: $\cup$, $\circ$, *

Def^n: R is a regular expression if.

Basis
- (i) $a \longrightarrow a \in \Sigma$ for some Σ $\{a\}$
- (ii) $\epsilon \longrightarrow \{\epsilon\}$
- (iii) $\phi \longrightarrow$ language corresponding $\equiv \phi$

} languages corresponding ↵

(iv) $R_1 \cup R_2$, where R_1, R_2 are Regular ↓

(v) $R_1 \circ R_2$ ↓ $L(R_1) \cup L(R_2)$

(vi) R_1^* ↓ $L(R_1) \circ L(R_2)$

↓ $(L(R))^*$

$(0 \cup 1) = \{0,1\} = \Sigma$

$(0 \cup 1)^* = \Sigma^*$

$\Sigma^* a\ b\ a\ \Sigma^* \Rightarrow \Sigma = \{a\ b\}$

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

$(0\ \Sigma^*\ 0) \cup (1\ \Sigma^*\ 1) \cup 0 \cup 1$

* $L \circ \{\epsilon\} = L$

$\phi^* = \{\epsilon\}$

$L \circ \phi \swarrow^{L} \searrow_{\phi}$

$x = x_1 x_2$, $x_1 \in L$, $x_2 \in \phi$

↓

So; $x_2 = \{\epsilon\}$

$L \circ \phi = [illegible]$

$= \phi$

Theorem: Suppose L is the language described some regular expression R and then L is regular.

i) a → o —a→ ◎

ii) ϵ → ◎

iii) ϕ → o (no accept state)

↳ on each input, output ≡ empty set.

$R_1 \cup R_2$; NFA is closed

$R_1 \circ R_1$; NFA is "

ulg R^* ; "

Proof by Construction:

$(ab)^* b$

corresponds to a

→ o —a→ ◎

corresponds to b

→ o —b→ ◎

corresponds to ab. (concatenation).

→ o —a→ o —ϵ→ o —b→ ◎ ⇒ "ab"

corresponds to $(ab)^*$ (star).

→ ◎ —ϵ— a → o —ϵ→ o —b→ ◎ → $(ab)^*$

ϵ

corresponds to $(ab)^* b$

no longer remains the final state.

→ o —ϵ→ o —a→ o —ϵ→ o —b→ o

ϵ

o —b→ ◎

For Ex:

→ o —0U1→ o

q_1 q_2

(similar to)

→ q_1 —0→ q_2 , 1 (self-loop on q_1)

Theorem → If a language L is regular then ∃ some regular expression R which describes it.

Idea: $L \rightarrow DFA \Rightarrow GNFA_K \rightarrow R$.

GNFA — Generalised NFA. ⟶ Complete Graph (every two nodes, there is an edge).

(i) has unique final state
from every state to final state, transition exists but no outgoing arrow from final state

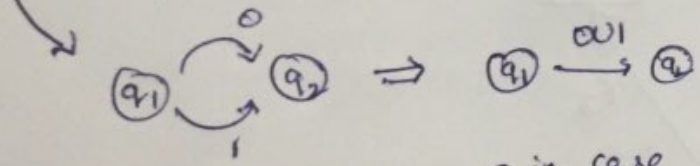
different from initial state (every state can reach to final state through some RE)
↑ (has incoming edges from states)

(ii) Initial state can reach to every other edge
(No incoming edges) — only outgoing edges

(iii) Each edge is defined by regular expressions unlike symbols.

* GNFA has atleast 2 states. (since it is a complete graph)

⇒ Given DFA, convert it to GNFA?
$\delta : Q \times \Sigma \rightarrow Q$.
multiple edges are clubbed using regular expressions



→ create an initial state (empty transitions to the states)

→ lly create a final state (empty transitions leading to that particular state)

Utilize "ϕ" states (in case no transition exists) but to complete the graph.

$GNFA_K$ → GNFA with K states.

⇓ convert

$GNFA_{K-1}$

⇓

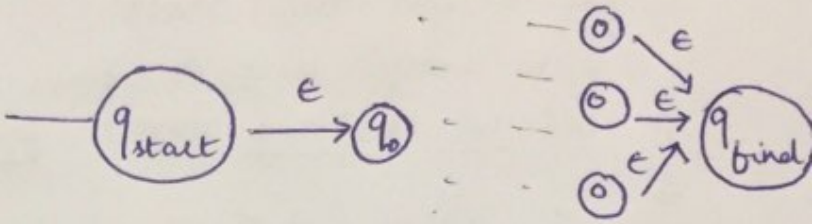
$GNFA_{K-2}$ . . . . ⇒ $GNFA_2$ (initial, final)

GNFA → $(\Sigma, Q, \delta, q_{start}, q_{final})$

$\delta: (Q / \{q_{final}\}) \times (Q / \{q_{start}\}) \to R$

q_{start} ← × No Incoming edges

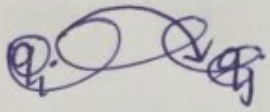
q_{final} → × No outgoing edges



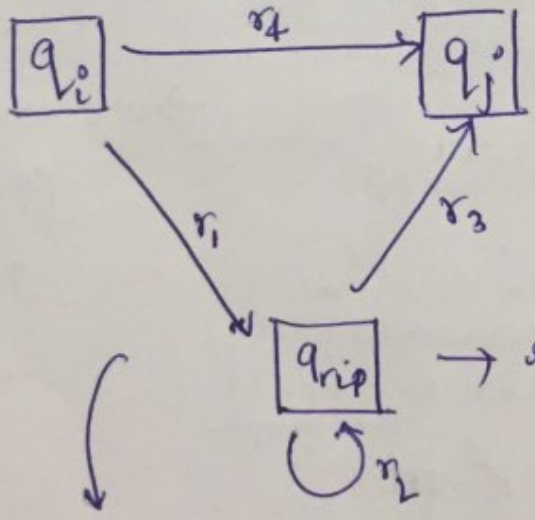
K > 2 →

$q_i \; q_j \in Q$

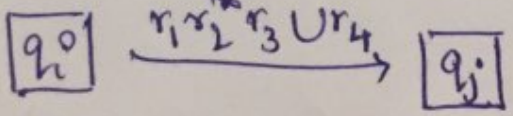
$q_{rip} \in Q$



for turning a K state GNFA into a K-1 state GNFA



Can be converted into regex for K-1 GNFA state



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

$Q = \{q_1, q_2\}$

$\delta \equiv Q \times \Sigma \to Q$

$q_1 \nearrow 0 \to q_1$

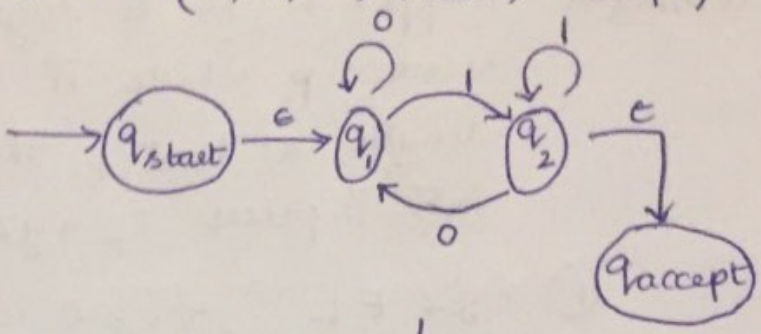
$q_1 \searrow 1 \to q_2$

$q_2 \nearrow 0 \to q_1$

$q_2 \searrow 1 \to q_2$

DFA $\equiv (\Sigma, Q, \delta, q_1, \{q_2\})$

GNFA $\equiv (\Sigma, Q, \delta', q_{start}, q_{accept})$



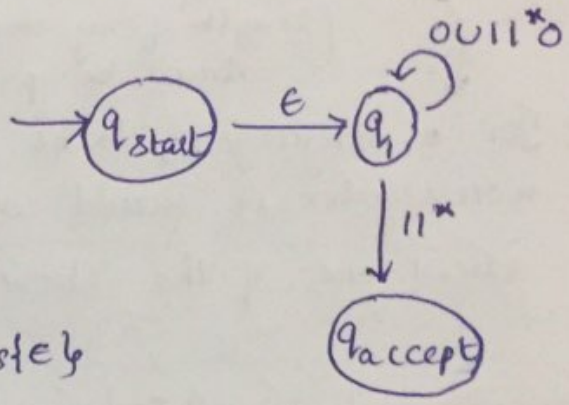
↓

Since there are 4 states. we'll remove one state, q_2

$q_1 \to q_1$ $(0 \cup 11^*0)$

↓

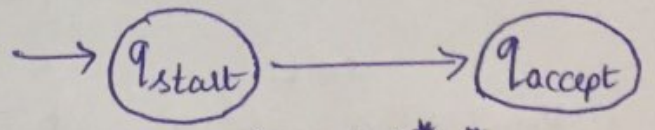
contains atleast 11, you just cant write 1^* coz it ~~can~~ contains $\{\epsilon\}$



↓

Now, remove q_1 state.

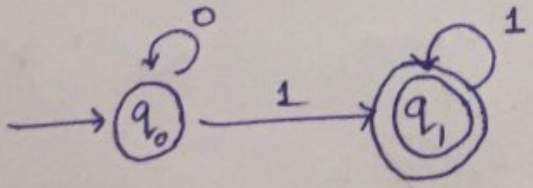
Final Regular Expression $\equiv$



$(0 \cup 11^*0)^*11^*$

Are all languages regular?

$L = \{0^n 1^n \mid n \geq 0\} \leq \Sigma^*$, $\Sigma = \{0, 1\}$ → Is L regular?



$A \Rightarrow B$

$\sim B \Rightarrow \sim A$

Theorem →

Pumping Lemma – Suppose L is a regular language, then ∃ a number p where if S in any string of length at least p then S can be divided into 3 pieces $S \equiv xyz$ such that

(i) $xy^iz \in L$, $\forall i \geq 0$.

(ii) $|y| > 0$, y should not be empty
$y \neq \epsilon$

(iii) $|xy| \leq p$.
(length of xy should atmost be p)

If y is empty ;
$xy^iz \rightarrow L$ (vacously true)
any string satisfies.

→ For a language L, to be non regular, it should contradict atleast one of the above conditions

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

Let ∃ p;

$S = 0^p 1^p = xy^iz$

$S \in L$

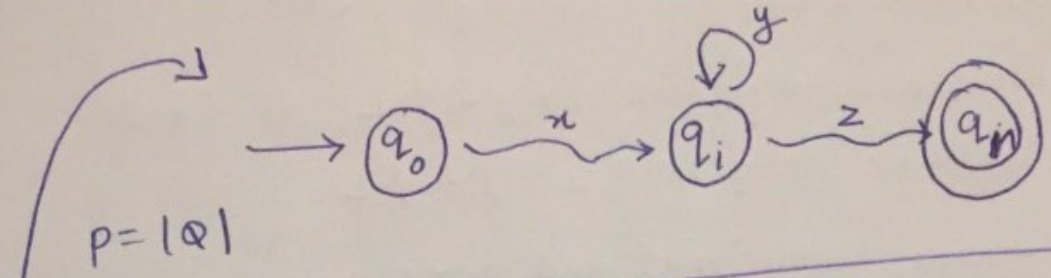
Possibilities → y contains only zeros (More zeros than ones)
y contains only ones (More ones than zeroes)
y contains some zeros and ones

$y = 01$

$y^i = 010101$

not $\notin L$

DFA $(\Sigma, Q, \delta, q_0, F) \rightarrow L$



$p = |Q|$

Since L is regular, $\exists$ DFA $M(Q, \Sigma, \delta, q_0, F)$ which recognizes L,

$p = |Q|$

$\hat{\delta}(q_0, s) \in F$

$s = x_1 x_2 x_3 \cdots x_p x_{p+1} x_{p+2} \cdots x_{n-1} x_n$, $n \geq p$

$q_0 \xrightarrow{x_1} q_1 \xrightarrow{x_2} q_2 \xrightarrow{x_3} q_3 \quad q_{p-1} \xrightarrow{x_p} q_p \cdots$

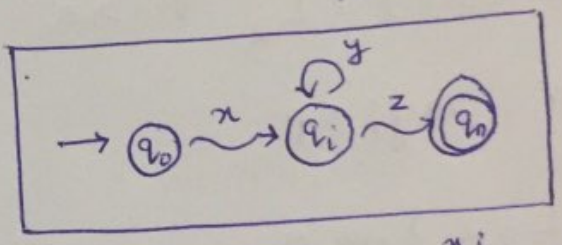
$q_{n-1} \xrightarrow{x_n} q_n$

$\Rightarrow xy^k z = s$

$x = x_1 x_2 \cdots x_i$

$y = x_{i+1} x_{i+2} \cdots x_j$

$z = x_{p+1} \cdots x_n$



$q_0 \xrightarrow{x_1} q_1 \xrightarrow{x_2} q_2 \cdots q_{i-1} \xrightarrow{x_i} q_i \xrightarrow{x_{i+1}} q_{i+1} \cdots q_{j-1} \xrightarrow{x_j} q_j \cdots$

$\cdots \rightarrow q_{p-1} \xrightarrow{x_p} q_p \cdots \rightarrow q_n$

<u>for y;</u>

$s' = x_{i+1} x_{i+2} \cdots x_j$

$s'' = s' s'$

$s''' = s' s' s'$

$\hat{\delta}(q_i, s') = q_j = q_i$

$\hat{\delta}(q_i, s'') = \hat{\delta}(q_i, s'')$

$= \hat{\delta}(\hat{\delta}(q_i, s'), s') = \hat{\delta}(q_i, s')$

Ex: $L_1 = \{0^n 1^n \mid n \geq 0\}$ → Not Regular, $\exists p$

$s = 0^p 1^p \in L_1$ $\quad s = xyz = 0^{p_1 p}$

① (i) If y contains only zeros, $xy^i z \notin L$ if $i \geq 2$

(ii) If y contains only ones, $xy^i z \notin L$

② If y contains some zeros and some ones.

$L_2 = \{w \in \{0,1\}^* \mid w$ contains equal number of zeros and ones$\}$

If $L_2 \rightarrow$ regular $\exists$ p → pumping length.

$s = 0^p 1^p = xyz$

$x = \epsilon, z = \epsilon$

$\rightarrow (0^p 1^p)^i \in L_2 \quad \forall i \geq 0.$

$\rightarrow |xy| \leq p$

$L_3 = \{0^m 1^n, m, n \geq 0\} = 0^* 1^*$ $\quad L_2 \cup L_3 = L_1$ → Not Regular.

$L_3 = \{ww, w \in \{0,1\}^*\}$, if L_3 is regular $\exists p$

Ex: $s = uu \quad u = 0.$

$x = \epsilon, y = u, z = u$

$s = 0^p 1 0^p 1 \in L_3$

$|xy| \leq p$ $\quad xy^2z \notin L_3$

y contains only zeros $\quad xy^3z \notin L_3$

$L_4 = \{ ww^2 \mid w \in (0,1)^* \}$

If $w = w_1 w_2 \ldots w_{n-1} w_n$ then $w^2 = w_n w_{n-1} \ldots w_2 w_1$

$L_5 = \{ a^{n^2} \mid n > 0 \}$, $S = a^{p^2} = xyz$

$|y| \leq |xy| \leq p$

$\Downarrow$

$|y| \leq p$

$xy^2z \notin L_5$

$p^2 < |xy^2z| < (p+1)^2$ $\rightarrow$ $p^2 = |xyz| < |xy^2z| < (p+1)^2$

$|xy^2z| = |xyz| + |y| \leq p^2 + p \leq p^2 + 2p + 1$

$y \notin \epsilon$

perfect square.

$L = \{ w \mid w$ contains equal number of 01, 10's as substrings$\}$ $= L_1 \cup L_2$

L_1 : starting with 1 $\quad$ L_2 : starting with 0.

00000 $\in L$ (zero no.)

1111..1 $\in L$ "

10 $\notin L$

100 $\notin L$

1000.... 00 $\notin L$

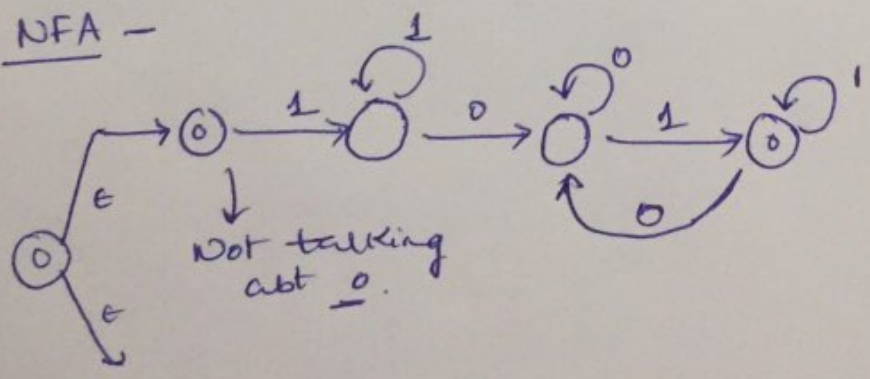
10b.... b01 $\in L$

in this case;

100.... 001

111..1 $\in L$ $\quad$ 111..10 $\notin L$

NFA –



Two cases should be handled ↓ one which starts with 1 and the other with 0

grammar ⇒ Context free Grammar (CFG)

(V, Σ, R, S)

V - finite set of vertices.

Σ - finite set. (disjoint, $V \cap \Sigma = \phi$
of terminals

R - Finite set of Rules
productions / substitution Rules

(single variable) $A \rightarrow \alpha$ (combination of terminals & V) string of variables and terminals

where $A \in V$, $\alpha \in (V \cup \Sigma)^*$

S → start variable $\in V$.

According to grammer: G1 ⇒ $\Sigma = \{0, 1, \#\}$

$A \rightarrow 0A1$
$A \rightarrow B$
$B \rightarrow \#$
} G_1

$A \Rightarrow 0A1 \Rightarrow 00\underline{A}11$
⇓
$000A111$
⇓
$000B111$
⇓
$000\#111$

$000\#111$

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

→ only # can also be produced
↓
$A \rightarrow B$
$B \rightarrow \#$

According to grammer: G2 ⇒

$A \rightarrow 0A1$
$A \rightarrow \#$

G3: $A \rightarrow 0B1$
$B \rightarrow A/\#$

Cannot produce only "#".

$\in \{0^n \# 1^n \mid n > 0\} = L$

G4: Language?

$S \rightarrow asb \mid ss \mid \epsilon \quad \rightarrow \Sigma = \{a, b\}$

for this, replace a with "{" ["left paranthesis"] and b with "}"

we need to have balanced parantheses

$\overset{(}{a}\overset{(}{a}\overset{)}{b}\overset{)}{b}$ $\quad$ $\overset{(}{s}\overset{)}{s}\overset{(}{s}\overset{)}{s}$

$A = \{ a^n b^n / \substack{a \neq b \\ a,b \in \{0,1\} \\ n \geq 0} \} \rightarrow \{0^n 1^n \mid n \geq 0\} \cup \{1^n 0^n \mid n \geq 0\}$

$= A_0 \cup A_1$

$A_0 = \{0^n 1^n \mid n \geq 0\}$

$S_0 \rightarrow 0 S_0 1 \mid \epsilon$

$A_1 = \{1^n 0^n \mid n \geq 0\}$

$S_1 \rightarrow 1 S_1 0 \mid \epsilon$

because $A = A_0 \cup A_1$

$S \rightarrow S_0 \mid S_1$

$\Downarrow$

$S \rightarrow S_0$

$S \rightarrow S_1$

A = set of palindromes over $\{0,1\}$

$w \in A$, if $w = w^R$.

where $w = w_1 w_2 \ldots w_{k-1} w_k$

$w^R = w_k w_{k-1} \ldots w_2 w_1$

$S \rightarrow 0S0 \mid 1S1 \mid \epsilon$

Grammar corresponding to language A;

Base case : $S \rightarrow \epsilon \mid 0 \mid 1$

$S \rightarrow \epsilon \mid 0 \mid 1 \mid 0S0 \mid 1S1$

Defn: $\sim \delta$

$uAv \Rightarrow uwv$

uAv yields uwv if $A \rightarrow w$ is a rule $\in R$.

Defn: $\sim \hat{\delta}$ Derivation $u \overset{*}{\Rightarrow} v$ (u derives v)

u derives v if $\exists\, u_1, u_2 \ldots u_k$, $k \geq 0$, s.t.

$u \Rightarrow u_1 \Rightarrow u_2 \Rightarrow u_3 \ldots \Rightarrow u_{k-1} \Rightarrow u_k \Rightarrow v$

CFG → $G_1 (V, \Sigma, R, S)$ $\quad L(G) \subseteq \Sigma^*$

Language described by G → $L(G) = \{ w \in \Sigma^* \mid S \overset{*}{\Rightarrow} w \}$

A → Regular Language

$\exists$ DFA $M (Q, \Sigma, \delta, q_0, F)$

$L(M) = A$

$w = w_1 w_2 \ldots w_{k-1} w_k \in A.$

$q_0 \xrightarrow{w_1} q_1 \xrightarrow{w_2} q_2 \ldots \xrightarrow{w_{k-1}} q_{k-1} \xrightarrow{w_k} q_k$, $q_k \in F$

$q_0 \to w_1 q_1 \to w_1 w_2 q_2 \to w_1 w_2 w_3 q_3 \ldots \to w_1 w_2 \ldots w_{k-1} w_k q_k$

To get only $[w_1 w_2 \ldots w_{k-1} w_k]$ string, $q_k = \epsilon$

for each state in Q, a new state is added.

$V = \{ R_0, R_1, R_2 \ldots R_n \}$ where $n+1 = |Q|$

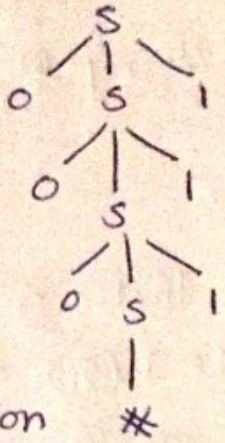
(i) $S = R_0$

(ii) $R_i^o \to a R_j$ if $\delta(q_i^o, a) = q_j$ $\quad (q_1) \to (w_2 q_2)$

(iii) $R_i^o \to \epsilon$ if $q_i \in F$

Eg: $S \rightarrow 0S1 \mid \#$

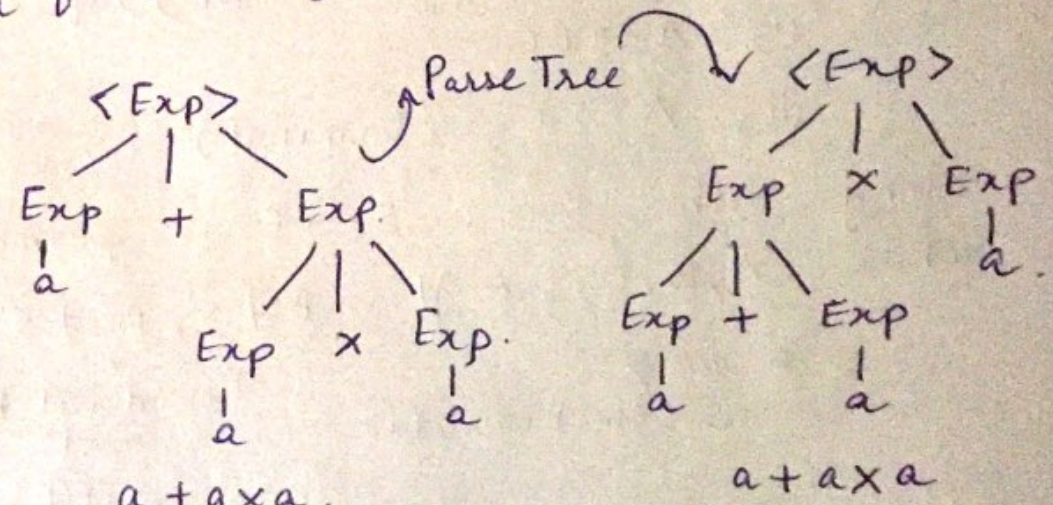
000#111



000#111 ↑ parse tree.

$\langle Exp \rangle \rightarrow \langle Exp \rangle + \langle Exp \rangle \mid \langle Exp \rangle \times \langle Exp \rangle \mid a$

↑ a + a x a from this grammer.



↓ has two different parsetrees.

Def^{n}: Leftmost derivation

$A \rightarrow a_1 a_2 a_3 A_1 A_2 a_4 A_3 A_7$

$A_1 \rightarrow a_4$

$A_3 \rightarrow a_3$

$A \rightarrow a_1 a_2 a_3 A_1 A_2 a_4 A_3 A_7$

↓

$a_1 a_2 a_3 A_1 A_2 a_4 a_3 A_7$

$\Rightarrow$ G is CFG, $w \in L(G)$

We say that w is derived ambiguously (in grammar G) if there are at least two different parse trees which yield the same string w

eg; $(a+a) \times a \neq a+(a \times a)$

Defn: Let G be a CFG; we say that G is ambiguous if $\exists\, w \in L(G)$ s.t w is derived ambiguously in G.

Defn: Let G be a CFG, we say that G is in "Chomsky Normal Form" (CNF), if

(i) $A \to BC$

(ii) $A \to a$ (terminal) $\longrightarrow$ Can replace any variable with terminal

along with this; $S \to \epsilon$ is permitted

where $A, B, C, S \in V$, $B \neq S$, $BC \neq S$

(Should not be start.)

* $a \in \Sigma$
S $\to$ start variable

a CFG

Theorem: If G is a C.F.G, then $\exists$ G' which is equivalent to G and G' is in C.N.F.

Proof: (i) G not being in CNF, reasons:

(1) Introduce a new (starting) variable say, $S_0 \to S$ (start)

→ Initial variable may appear in the right side

→ $A \to \epsilon$; ϵ-rules

→ $A \to B$; unit-rules (it should be giving a terminal but not a variable)

→ $A \to u_1 u_2 \ldots u_k$ $u_i \in V \cup \Sigma$ $k \geq 0$

(length-2 is being violated)

(2) For each occurence of A, we replace it with empty string

$X \to uAv \mid uv$

$Y \to uAvAw \mid uvAw \mid uAvw \mid uvw$

③ $A \to B$ (to remove such a rule), what are the extra rules to be introduced

& $B \to X$ } $\to A \overset{*}{\Rightarrow} X \to$ (can be more than 2)

if $A \to B$ is removed, going to be removed, X cannot be derived.

do it for every unit rule — just a variable

So, we have to introduce $A \to X$

④ $A \to u_1 u_2 \ldots u_{k-1} u_k$ [strings of length more than 2]

where each $u_i^\circ \in V \cup \Sigma$ (variable or terminal)

$A \to u_1 A_1$
$A_1 \to u_2 A_2$
$\vdots$
$A_{k-2} \to u_{k-1} u_k$

Every rule is a concatenation of 2 symbols

$A \overset{*}{\Rightarrow} u_1 u_2 \ldots u_k$

for length more than 2:

$Z \to uAV$
$Z \to uA_1$
$A_1 \to AV$

[redacted], [redacted]
$U \to u$ if $u \in \Sigma$ → u is terminal

u_i° → can be either variable or terminal

if $u_i^\circ \equiv$ terminal; $U_i^\circ \to u_i^\circ$ → introduce a new variable, U_i

[$A \to BC$] → B and C both have to be variables

$S \to A$
$\times$ ~~$A \to B$~~
$\times$ ~~$B \to C$~~
$C \to a$

$S \to A$
$B \to C$
$C \to a$
$A \to C$

We are not going to re-introduce a removed rule

~~S~~ Not needed as the variables that can be reached from B and can be reached from A and S can reach A, so S can reach those variables

Ex: $S \to ASA \mid aB$
$A \to B \mid S$
$B \to b \mid \epsilon$ ~ epsilon is allowed $\Rightarrow \alpha \in (V \times \Sigma)^*$ for $A \to \alpha$

→ ① Introduce a new start variable

$S_0 \to S$
$S \to ASA \mid aB$ → $S \to ASA$, $S \to aB \mid a$
$A \to B \mid S$
$B \to b \mid \epsilon$ → $B \to b$, $XB \to \epsilon$ (removing this rule)

$A \to B|S|\epsilon$ (as $B \to \epsilon$ is removed)

→ Now;

$S_0 \to S$

$S \to ASA|aB|a$

$A \to B|S|\epsilon$

$B \to b$

Removing $A \to \epsilon$ (ε-rule)

↓

$S_0 \to S$

$S \to ASA|aB|a|SA|AS|S$

$A \to B|S$

$B \to b$

↓

Removing unit rules;

→ ✗ $A \to B$

can be removed.

$S_0 \to S$

$S \to ASA|aB|a|SA|AS|S$ ∞ [∵ $S \to S$ not required]

$A \to b|S$

$B \to b$

✗ $A \to S$

$S_0 \to S$

$S \to ASA|aB|a|SA|AS$

$A \to ASA|aB|a|SA|AS|b$

$B \to b$

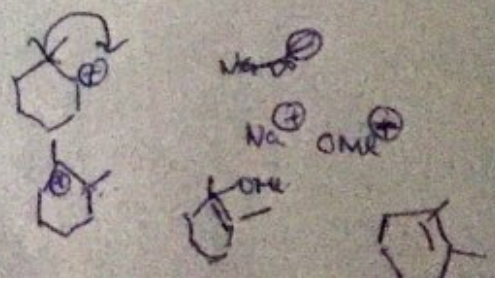
✗ $S_0 \to S$

$S_0 \to ASA|aB|a|SA|AS$

$S \to ASA|aB|a|SA|AS$

$A \to ASA|aB|a|SA|AS|b$

$B \to b$



For lengths more than 2;

→ $S_0 \to AA_1 / A_2B / a / SA / AS$

$A_1 \to SA$

$A_2 \to a$

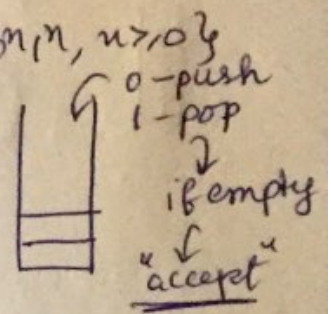
$S \to AA_1 / A_2B / a / SA / AS$

$A \to AA_1 / A_2B / a / SA / AS / b$

$B \to b$

→ In CNF form

$\{0^n 1^n, n \geq 0\}$



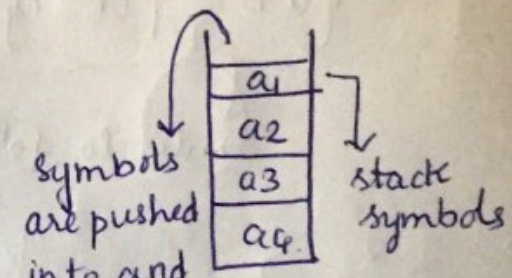
Infinite memory (stack) ? will DFA, NFA accept more languages.

Push down Automata → NFA + stack (infinite memory)

NFA $(Q, \Sigma, \delta, q_0, F) \to \delta: Q \times \Sigma_\epsilon \to P(Q)$

$(Q, \Sigma, T, \delta, q_0, F) \equiv$ PDA

T - finite set / stack symbols.



$\delta: Q \times \Sigma_\epsilon \times T_\epsilon \to P(Q \times T_\epsilon)$

$\delta(r, a, b) = (r', c)$

b is at top of the stack

reaches state r' from r upon a.

b is popped out and c is pushed in

$\epsilon, \epsilon \to \epsilon$ (don't disturb the stack, on reading empty symbol) (same as NFA)

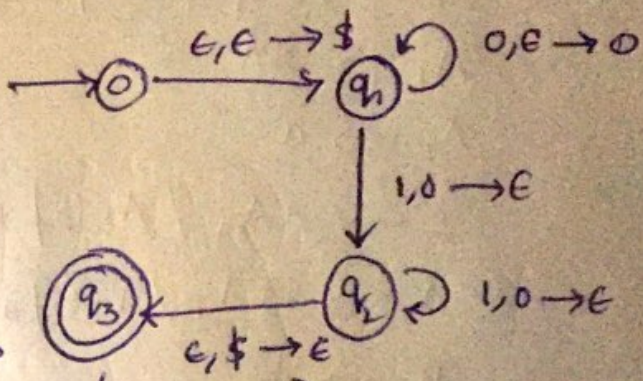
$L = \{0^n 1^n \mid n \geq 0\}$ $\xrightarrow{T}$ add a new symbol, $\$ \in T$

$w = w_1 w_2 w_3 \epsilon = w_1 w_2 w_3$

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

on encountering 0, push zero onto the stack.

→ add a special symbol.



on encountering 1, push nothing for the first time but pop a zero.

we are not going to see any 1s as we never push 1s onto the stack

the last symbol ≡ $

end of input = ϵ

What if there are still more symbols (either 1s or 0s) in the input string

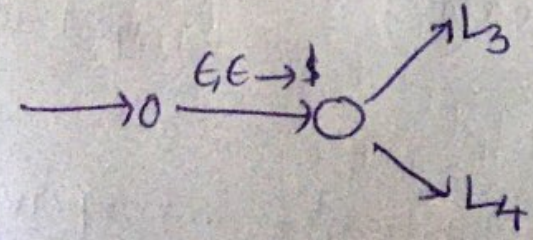
$\delta(q_3, 1, \epsilon) = \phi$

$L_2 = \{a^i b^j c^k \mid i,j,k \geq 0,\ i=j \text{ or } i=k\}$

$= \{a^i b^i c^k \mid ik \geq 0\} \cup \{a^i b^j c^i \mid i,k \geq 0\}$

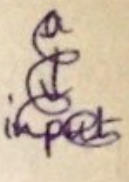
similar to $\{0^n 1^n\}$

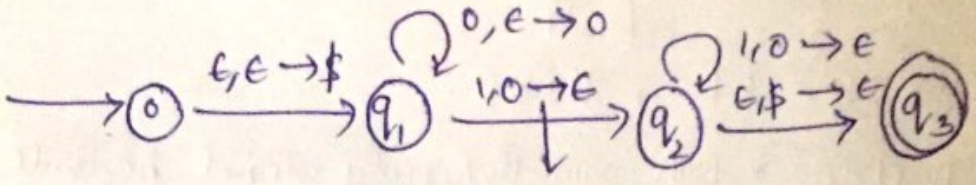
dont need to bother about C



PDA – Push down automata.

$L_1 = \{0^n 1^n \mid n \geq 0\}$





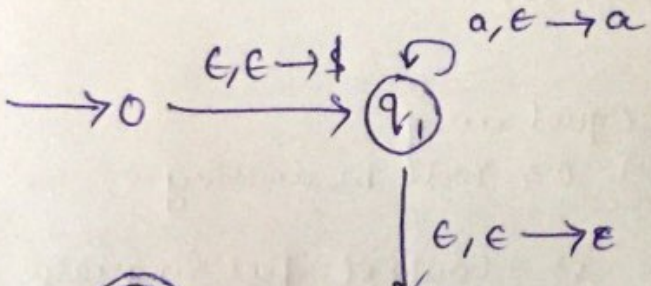
for each, popping zero, but pushing nothing onto the stack.

accepts only strings having equal no. of 1s and 0s

$L = \{ww^R \mid w \in \{0,1\}^*\}$ → only even length palindromes.

$a = \{0, 1\}$

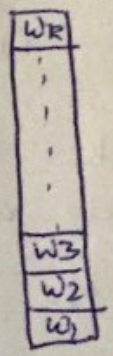
just a subset of "set of palindromes"



pop out all the possible.

$w = w_1 w_2 \ldots w_k$

stack. $w^R = w_k \ldots w_3 w_2 w_1$



Can think of multiple stacks for example all the possible actions

Non Deterministic Version of Push Down Automata

⇒ Context free Grammer and Push Down Automata :-

↓

→ Is there a context free grammer for a push down automata? Yes. and same for vice versa.

$L = \{a^n b^n c^n \mid n \geq 0\}$ → is it context free language?

↓

not context free [as how will you account for the "c" alphabets in the stack.

Pumping lemma for Context free languages ⇒

∃ a number p, p → pumping length

$s \in L$, $|s| \geq p$, $s = uvxyz$, such that

(i.) $uv^i xy^i z \in L$, $\forall i \geq 0$.

(ii.) $|vy| > 0$ ↘ both simultaneously cannot be null (empty)

(iii.) $|vxy| \leq p$.

Proof :-

$$s = a^p b^p c^p$$

$$s = uvxyz$$

→ vxy is ~~subsystem~~ string of s

Acc^{n} to (i.) condition; $i = 0$;

↳ $uxz \in L$

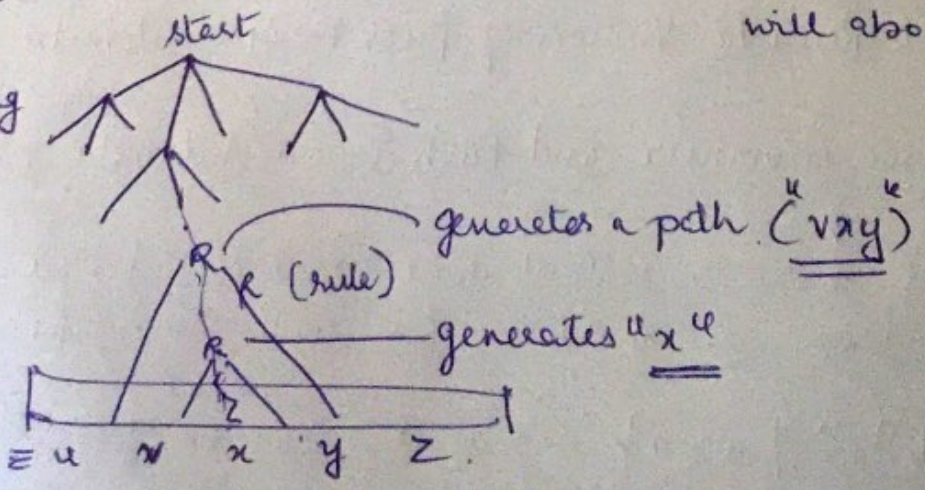
↓ unequal no. of

either b's or c's will be missing

→ Since the language is context-free language, there exists a context free grammer

If the string $\in L$ is long, then the parse tree ↘ will also be large.

Parsetree ≡ ↑ for string s



so can we pump ⇒ $uv^i xy^i z$.

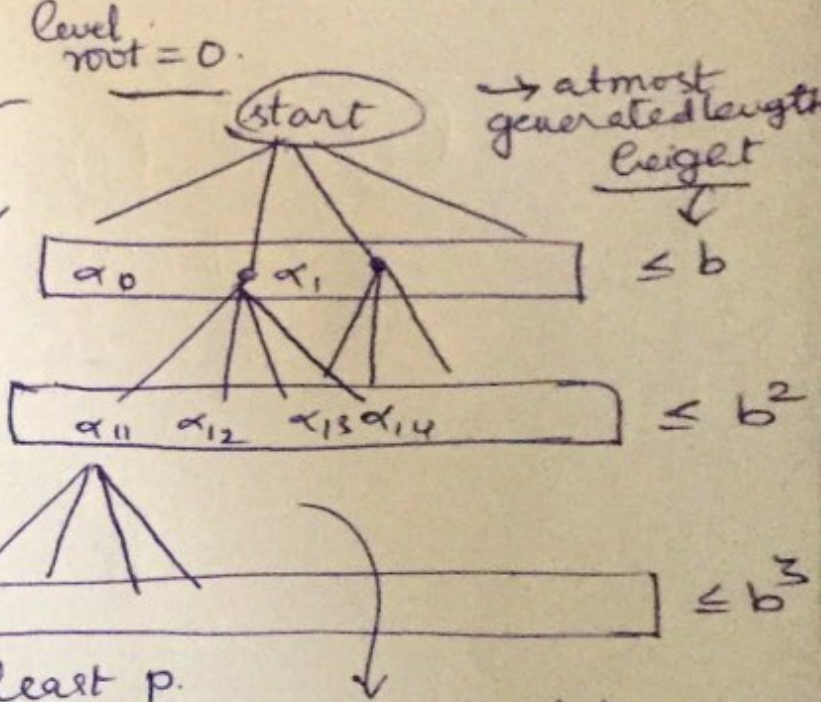
Basically; start $\overset{*}{\Rightarrow}$ uRz

$R \overset{*}{\Rightarrow} vRy$

$R \overset{*}{\Rightarrow} x$.

For a parse tree

If height is h, then string length $\leq b^h$



So, what should be the value of P?

$P = b^{|V|+1}$

height $\equiv |V|+1$

s (string) → of length atleast P.

all the intermediate symbols → variables

Contains $\equiv |V|+2$ Nodes

Due to this, two variables must repeat.

$|V|+1 \equiv$ Variables $\Rightarrow$ length 2 $\Rightarrow$ Nodes = 3

$1 \equiv$ Terminal

↓

2 nodes + 1 terminal