Grading Policy :-

Quiz → 20%
MidSem → 30%
Endsem → 50%

Minimum 2.
Maximum 4.

Books :- 1) Introduction to Automata Theory, Languages and Computation -
- John E Hopcroft, Jeffrey D. Ullman -
- " , Rajeev Motwani

2) Elements of the theory of computation
- Henry R Lewis
- Christos H Papadimition. (Pearson)

3) Peter Linz
4) Dexter C Kozen. } Reference.
5) Thomas A Sudkamp -

Formal languages and Automata Theory :-

Automaton :- Various kinds of systems can be modelled using an automaton.

Ex Finite State Machine -

(& used to design some hardware).

Any physical system can be modelled using an automaton.

Compiler :- converts a program (HLL) → Machine lang.. (characters) ka sequence.

(Lexical Analysis
↳ Step 1 for compilation)
↳ keyword
↳ Variable.
↳ constants.

Regular Expressions
↓
Stream of Tokens → (parser) Parsed Code → (Mach. language)
↓
Analyzes code for correct grammar in the stmts.

Parser ⇒ Push Down Automata.

Ultimate Model of Computation - (Turing / Curing) Machine - → Algorithm

Decidable Problems / Non Decidable Problems -

↳ How difficult is it to solve problem?

Language :-

Alphabet → Strings → Language.

Regular language
Context Free Lang → Visdom Automata.
Recursively Enumerable language.
↳ Recursive language.

For each language different Automata.

Turing Machine.

Alphabet :- (Σ) (Γ)

Finite non-empty set

Constraint :-
Finite
Non empty.

$\Sigma = \{a, b\}$. or $\Sigma_1 = \{0, 1\}$ $\Sigma_3 = \{q\}$.
↓
Symbols - of alphabet.

initial english alphabet. or digits.

String: Finite sequence of symbols from alphabet over an alphabet

Ex (a, b, a, a, b)

↳ abaab
↳ b
↳ aab
↳ aba
↳ ( )
↳ empty sequence is represented by epsilon [lambda sometimes]

ε / λ → empty sequence.

$|abaab| = 5$

Last letters of the alphabet - u, v, w, x, y, z - are used to represent strings -.

$x = abaa$

$|x| = 4$.

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

$\Sigma^* = \{\varepsilon, 0, 1, 00, 01, 10, 11, 000, 001 \ldots\}$.

↳ lexicographically ordered set of all strings over the alphabet. [ordered ace to length].

Language L over an alphabet Σ is a subset of Σ^*

Any subset of Σ^* is L.

$L \subseteq \Sigma^*$.

Concatenation :-

(x, y).
↓ ↙
strings.

$x = a_1 a_2 \ldots a_n$

$y = b_1 b_2 \ldots b_n$.

$xy = a_1 a_2 \ldots a_m b_1 b_2 \ldots b_n$.

$|xy| = |x| + |y|$

$\varepsilon x = x \varepsilon = x$.

A_n = Set of all strings of length n. over alphabet Σ.

$$\Sigma^* = \bigcup_{n \geq 0} A_n.$$

Substring:-

Prefix.

Suffix

|x|a

|x

Defining

Σ

L = ϕ

L = {ε}

L = {

L = {

L =

Conco

L_1 =

L_2 =

w

Substring:- x is a substring of y if x occurs in y

$y = uxv$, for $u, v \in \Sigma^*$.

Prefix. $y = uxv$

$u = \varepsilon$. $\Rightarrow$ then x is prefix of y.

Suffix $y = uxv$

$v = \varepsilon$ $\Rightarrow$ then x is suffix of y.

$|x|_a$ → how many times a occurs in x.

$|x|_y$ → how many times y occurs in x.

Defining a formal language.

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

$L = \phi$ → No element lang.

$L = \{\varepsilon\}$ → 1 element lang. [empty string].

$L = \{0x \mid x \in \{0,1\}^*\}$.

$L = \{x10y \mid x, y \in \{0,1\}^*\}$

$L = \{x \in \{0,1\}^* \mid |x| = 2n\}$.

Concatenation of languages:- $L_1 L_2$.

$L = L_1 L_2 = \{xy \mid x \in L_1 \text{ and } y \in L_2\}$.

$L_1 = \{0, 10\}$

$L_2 = \{11\}$ $L_1 L_2 = \{011, 1011\}$.

$L_1 L_2 \neq L_2 L_1$ because concatenation is not commutative.

$\varepsilon L_1 = L_1 \varepsilon = L_1$.

We can concatenate L with itself.

L

$LL = L^2$

$LLL = L^3$

$LLLL = L^4$.

$L^n = L^{n-1} L$.

Kleen closure/star.

$$L^* = \bigcup_{n \geq 0} L^n$$

$L^0 = \{\varepsilon\}$

$L^1 = L^0 L$

$L^+ = \{(01)^n \mid n \geq 0\}$.

$L^* = \{\varepsilon, 01, 0101, 010101, \ldots\}$

$L^* = L^0 \cup L^1 \cup L_2 \cup \ldots$

$L^* = \bigcup_{n \geq 0}$

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

$L = \{0, 1\}$

Positive closure

$L^+ = \bigcup_{n \geq 1} L^n$

$L^* = L^+ \cup \{\varepsilon\}$.

If ε is there in L, then it'll be there in L^+. If not, its not there.

1) $L_1(L_2L_3) = (L_1L_2)L_3$

2) $L_1L_2 \neq L_2L_1$

3) $L\{\varepsilon\} = \{\varepsilon\}L = L$.

4) $L\phi = \phi L = \phi$ → Proof by contradiction

Any concatenation can be written as x_1x_2

L_1 = L

L_2 = ϕ → not possible.

5) $\phi^* = \varepsilon$.

$L^* = L^0 \cup L^1 \cup L^2 \ldots$

$= \varepsilon \cup \phi^1 \cup \phi^2 \ldots$

$\boxed{\phi^* = \varepsilon}$.

6) $\{\varepsilon\}^* = \{\varepsilon\}$. # But ε is an element. $|\varepsilon| = 0$

7) $(L_1 \cup L_2)L_3 = L_1L_3 \cup L_2L_3$

$L_1(L_2 \cup L_3)$

$x \in (L_1 \cup L_2)L_3 \Leftrightarrow x = x_1x_2$

8) $L^*L = LL^* = L^{*+}$

↳ any element contained in L^+ will be there.

if $\varepsilon \in L$, $L^* = L^+$

$\Rightarrow LL^* = L^*L = L^* = L^+$ already..

9) $(L^*)^* = L^*$.

$(L_1L_2)^*L_1 = L_1(L_2L_1)^*$

$(L_1 \cup L_2)^* = (L_1^*L_2^*)^*$

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

$\{aba\}$

$\{a\}\{b\}$

↳ $\{ab\}\{a\}$

↳ $\{aba\}$.

ϕ $\{a, ba, ab\}$

$\{\varepsilon\}$.

$\{\varepsilon, a, aa, \text{---} 3$.

Regular Expression :- A regular expression over an alphabet Σ is defined recursively

1. ϕ, ε and a for each a in Σ, are regular expressions representing the languages ϕ, $\{\varepsilon\}$ and $\{a\}$ respectively.
2. If r_1 and r_2 are regular expressions representing languages R_1 and R_2 respectively, so are
   a) $(r_1 + r_2)$ its also regular exp. representing lang $R_1 \cup R_2$
   b) $r_1 r_2$ " " " " $R_1 R_2$
   c) (r_1^*) _____ R_1^*

$\phi \rightarrow \phi$ (corresponding lang)

$\varepsilon \rightarrow \{\varepsilon\}$

$a \rightarrow \{a\}$.

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

Reg Ex for $\Sigma^* = (0+1)^*$ → 0 or 1.

0^* → 0, 00, 000, ...

010

01100

:

any alphabet, any number of times.

$(1 \cup 01)^*$

$(1+01)^*$

$\Sigma = \{a_1, a_2, \text{---} a_n\}$

$(a_1 + a_2 \text{---} a_n)^*$.

The regular expression for the language containing 01 as substring —

$$(0+1)^* 01 (0+1)^*.$$

Reg ex for even length substrings only :-

$$((0+1)(0+1))^*$$

or $(00+01+10+11)^*$.

$(0+1)^* 00 (0+1)^*$

set of all strings such that no of zeros are even - ?

one occurrence of one pair of consecutive zeros.

$(1+01)^* (00) (1+10)^*$

Almost one pair of ~~a~~ consecutive: $(1+01)^*(0+00)(1+10)^*$

there are languages for which regular expressions are not possible.

* Σ^* → countably infinite set.

class of languages for which regular expression is possible
↳ Regular languages.

$\phi \rightarrow \phi$
$\varepsilon \rightarrow \{\varepsilon\}$
$a \rightarrow \{a\}$ $[\forall a \in \Sigma]$.

union
concatenation
Kleen closure.

} If these operations are applied finitely many times over a Σ, we get a class of regular languages.

$r \rightarrow$ regular expression
$L(r) \rightarrow$ languages represented by this r.

Ex: set of all strings over $\Sigma = [0,1]$ which do not begin with 01

→ $(11+00+10)(0+1)^*$ ~~U~~ $\varepsilon + 0 + 1$.

$L(r)$ should contain all strings made by r.
Also for L, r is not unique.

$r_1 \approx r_2$
if $L(r_1) = L(r_2)$

Properties of Reg. Exp:-

→ $r_1(r_2+r_3) \approx r_1 r_2 + r_1 r_3$
→ $r_1 r_2 \not\approx r_2 r_1$
→ $r\varepsilon \approx \varepsilon r \approx r$
→ $r_1(r_2 r_3) \approx (r_1 r_2) r_3$
→ $r\phi \approx \phi r \approx \phi$.
→ $b^+(a^*b^* + \varepsilon)b = b a^* b \quad b^+(a^*b^* + b)$

All properties for languages can be extended to regular expressions.

$\{a^n b^m \mid m, n \geq 0\}$ → Regular ✓

$r = a^* b^*$

$\{a^n b^m \mid n \geq 0\}$ → Not Regular.

$\{a^p \mid p \text{ is prime}\}$ → Not regular.

↓ a is p times

↳ length of all strings is prime.

Context Free Grammar :-

Sentence ⇒ <Sub><Verb><Object>

⇒ NP V obj

article Noun. ↙ verb

⇒ Article N V obj

verb

⇒ The boy eats NP.

⇒ The boy eats chocolate

Terminals.

(subject) (object)

NP verb NP

non terminals terminals terminals.

Formally, a content Free grammar (CFG) is a quadraple.

$$G = (N, \Sigma, P, S)$$

where, N is a finite set of non terminals.

Σ is a finite set of terminals

$S \in N$ is the start symbol

P is a finite set of production rules / productions

P contains all kinds of productions of the form ↓

(A, α) or $A \to \alpha$.

$A \in N$.

$\alpha \in (N \cup \Sigma)^*$.

Content Free Grammar generates content free lang.

$(N \cup \Sigma)^*$ → infinite set.

we have a binary relation over this set called derieve between 2 strings of this set.

$\alpha \Rightarrow \beta$ (α, β satisfy the relation if)

$\alpha = \alpha_1 A \alpha_2$
$\beta = \alpha_1 \gamma \alpha_2$ and there is a production rule in the grammar of the form $A \to \gamma$

$\alpha_1 A \alpha_2 \Rightarrow \alpha_1 \gamma \alpha_2$ because $A \to \gamma$

one step Relation

$\alpha \overset{*}{\Rightarrow} \beta$ → α derieves β in zero or more steps

$\overset{*}{\Rightarrow}$ reflexive transitive closure

Then α and β are related by $\overset{*}{\Rightarrow}$

There exists a sequence of steps $i_1, i_2 \ldots i_n$ such that

$\alpha \Rightarrow \alpha_1 \Rightarrow \alpha_2 \ldots \alpha_{n-1} \Rightarrow \beta$

Then $\alpha \overset{*}{\Rightarrow} \beta$ (length of derievation)

$\alpha \in (N \cup \Sigma)^*$

α is said to be in ~~sentential~~ sentential form if α is derivable from $\overset{*}{\Rightarrow}$ of grammar. $S \overset{*}{\Rightarrow} \alpha$

α is a sentence if it contains only strings.

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

if S derieves x. (string from Σ^+) using production rules of the grammar.

$L(G) =$

α is generated

start symbols

Ex $\Sigma = \{a\}$

grammar for

Production

$P = \{ S \to$

$S \to$

$\Sigma = \{a\}$

$N = \{S\}$

$P = \{S \to$

Q: set of

1)

2) at

3)

CFG :-

$G = (N,$

$L(G) = \{$

$L(G)$

Ex $\Sigma =$

1) $S \to$

$L \to$

2) $S -$

$L \to$

$$L(G) = \{ x \in \Sigma^* \mid S \overset{*}{\Rightarrow} x \}.$$

L is generated by grammar G (S is start symbol).

start symbols → a, b, c...

$L(G) = \{ x \in \Sigma^* \mid S \overset{*}{\Rightarrow} x \}$.

Ex $\Sigma = \{a\}$

grammar for this language:-

Production rules:-

$P = \{ S \to \varepsilon$.
$S \to aS \}$ → every string in the language a^* is generated by these prod. rules.

$L(G) = a^*$

$\Sigma = \{a\}$
$N = \{S\}$
$P = \{S \to \varepsilon \mid aS\}$.

Means $S \to \varepsilon$ and $S \to aS$.

Q: set of all strings over $\{a, b\}$ having :-

1) exactly 2 a's.
2) at least 2 a's.
3) at most 2 a's.

CFG :-

$G = (N, \Sigma, P, S)$

$L(G) = \{ x \in \Sigma^* \mid S \overset{*}{\Rightarrow} x \}$.

$L(G) = \{\}$ if no strings are generated.

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

1) $S \to aS \mid bS \mid \varepsilon$.
$L \to (a+b)^*$

2) $S \to abS \mid \varepsilon$.
$L \to (ab)^*$.

3) $S \to \varepsilon \mid aS$
$L(G) = \{ a^n \mid n \geq 0 \}$
$L \Rightarrow a^*$.

4) $S \to aSb \mid \varepsilon$.
$L(G) = \{ a^n b^n \mid n \geq 0 \}$.
↳ No regular exp. but language can be generated.

5) $S \to asb \mid bsa \mid \varepsilon$

Number of a's = No. of b's wale strings

6) $S \to asa \mid bSb \mid \varepsilon$.

Palindrome of even length.

$S \to aSa$
$S \to bSb$
$S \to a$
$S \to b$

This means CFG starts from a, b.

7) $S \to asa \mid bSb \mid a \mid b$

odd length palindromes

8) $S \to asa \mid bSb \mid a \mid b \mid \varepsilon$.

→ all palindromes.

9) $b^*ab^*ab^*$.

↳ only 2 a's

Grammar →

$S \to BaBaB$

$B \to bB \mid \varepsilon$.

$S \to bS \quad S \to aA$
$A \to bA \quad A \to aB$
$B \to bB$
$S \to abb^*ab^* \quad A \to ab^*$
$S \to bS \quad S \to aab^*$

10) $S \to bS \mid aA$.

$A \to bA \mid aB$.

$B \to bB \mid \varepsilon$.

Grammar :- $b^*ab^*ab^*$.

[Same language as 9].

Two grammars are equivalent if they generate the same grammar :-

$L(G_1) = L(G_2)$ then $G_1 \approx G_2$

Grammar is not unique.

In order to keep track of the state of number of a's or b's that have been counted, we need non terminals.

Even a's wala / aur :- maintain 2 non terminals.

a → even / odd.

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

lang. such that no string contains substring abc.

We need non terminals to remember how much of the substring has been visited.

$S \to bS \mid cS \mid aB \mid \varepsilon$.

$B \to aB \mid cS \mid bC \mid \varepsilon$

$C \to aB \mid bS \mid \varepsilon$

Parse Tree or Derivation Tree :-

Nodes → Terminals / Non Terminals / ε.

Internal Nodes → non terminals -

$S \overset{*}{\Rightarrow} \alpha$ (Sentential form) $\alpha \in (N \cup \Sigma)^*$.

$S \overset{*}{\Rightarrow} x$ (Sentence)

Sentential form ka tree :-

$A \to x_1 x_2 \dots x_k$.

x_1 ↓ leftmost child

x_k ↳ rightmost child -.

Yield of the tree :-

↳ collect leaves from left to right.

$S \overset{*}{\Rightarrow} \boxed{x}$

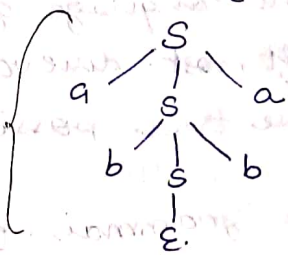
↳ yield of parse tree for S.

Ex $S \to aSa \mid bSb \mid \varepsilon$.

$S \Rightarrow aSa \Rightarrow abSba \Rightarrow abba$.

(Sa ↓ bSb; bS ↓ ε.)

Parse tree / Derivation



collect the leaves for the yield.

#. All strings over ~~abb~~ $\Sigma = \{a, b\}$ containing operators + and production :-

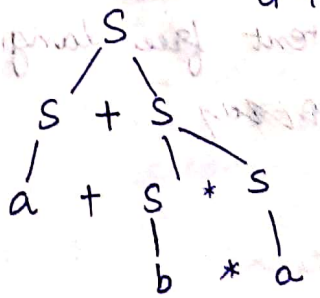
$S \to S+S \mid S*S \mid (S) \mid a \mid b$.

Ex $a+b*a$.

$S \Rightarrow S+S$
$\Rightarrow S + S*S$
$\Rightarrow a + S*S$
$\Rightarrow a + b*S$
$\Rightarrow a + b*a$

In the derivation one step can contain only one production -

$S \Rightarrow S*S$
$S*a$
$S+S*a$
$a+b*a$



Two derivations are equivalent if their parse trees are equivalent.

Leftmost Derivation for a string is unique :-

$S \Rightarrow x_1 x_2 - x_i - - x_k$. $x_i \to B$

Select the leftmost non terminal symbol and replace by ~~left~~ right side of production -.

$x_1 x_2 \text{ ---- } x_{i-1} B x_{i+1} \text{ ---- }$

$y_1 y_2 \text{ --- } y_i \text{ --- }$

left most non Terminal to be replaced by its production.

Correspondingly generate unique Right most derivn. of a String -.

Ambigous Grammar :-

If for some string in the language generated by this grammar has 2 left most derivations possible - / Two parse trees possible -

Can we write ambigious grammar for ~~or~~ every language ? No.

Inherently Ambigous :- No ~~un~~ unambigous way to write

Language Generated by CFG is called Context free Langu

$G = (N, \Sigma, P, S)$

$A \to \alpha \quad A \in N \quad \alpha \in (N \cup \Sigma)^*$

Context free language - (CFL)

if $A \to \alpha \in P \Rightarrow \alpha = xBy$ or $\alpha = x$.
$B \in N$
$x \in \Sigma^*$.

linear grammar - Every production contains at most one non terminal on RHS

Right linear grammar:- The non terminal occurs at the right most position -.
i.e, every production is of the form :-

$A \to xB$ $x \in \Sigma^*$ (any string)
$A \to x$. $B \in N$ (Non terminal)

Left linear Grammar:-.

Given a regular expression → you can construct a right linear grammar.

Given a Right-linear grammar → you have a regular expression.

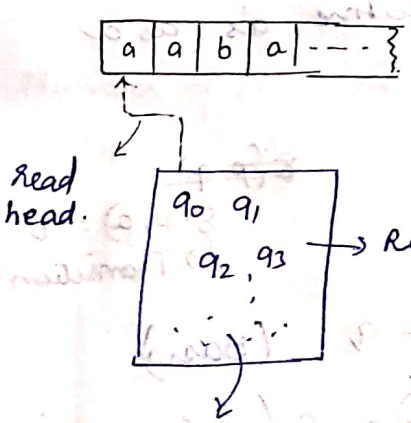
Automata :-.

Deterministic Finite Automata :-. (DFA)

$\Sigma \to$ alphabet.

an automaton is in a state.
changes state on recieving input. Keeps doing this until input terminates. Then acts acc to the last state.

Finite automata → finite no. of states.



→ Reads input from tap and changes state accordingly -

Notion → A string is said to be accepted by a finite automata if it reaches a finite state at the end of input.

Formally :- A DFA is a quintuple.

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

Q = finite set of states

Σ = input alphabet

$q_0 \in Q$ · is the start/initial state

$F \subseteq Q$ is the set of final/accept states.

δ = $Q \times \Sigma \to Q$ depending on what state we are in & what symbol we are reading, we go to a state.

(State transition fn)

$\delta(q, a) = p$

$p \in Q$

Formally :-

States are represented by ;

$q_0, q_1, q_2 \ldots$

$A = (Q, \Sigma, \delta, P, F)$

$Q = (P, q, r)$

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

$F = \{r\}$

δ	a	b
→ P	q	P
q	r	P
(r)	r	r

input state here.

final state.

Total 6 transitions —

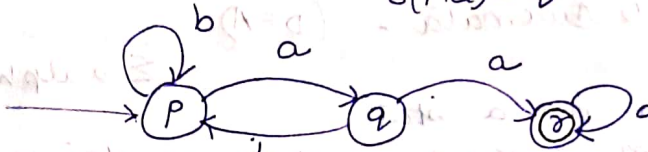
$\delta(P, a) = q$ $\delta(P, b) =$

$\delta(q, a) = r$ $\delta(q, b) =$

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

State Transition Diagram :-

$\delta(P, a) = q$



aa ✓

babx

Double circle → Final states.

any string containg 2 consecutive a's. as a substring is accepted.

Extended Transition Function :-

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

Ex. Transition function.

$\delta(P, a) = q$

↳ Transition F

1.) $\hat{\delta}(q, \varepsilon) = q$ (Basis)

2.) $\hat{\delta}(q, xa) = \delta(\hat{\delta}(q, x), a)$ (Inductive step)

Keep applying $\hat{\delta}$ to the prefix.

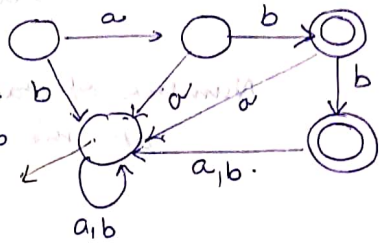
$L(A) = \{x \in \Sigma^* \mid aa$ is substring of $x\}$. & $|x|_{aa} \geq 1$

$$\hat{\delta}(p,aba) = \delta(\hat{\delta}(p,ab),a)$$
$$\delta(\delta(\hat{\delta}(p,a),b),a)$$

If $\hat{\delta}(q_0,x) \in F$, then x is accepted by DFA.

$L(A) = \{x \in \Sigma^* \mid \hat{\delta}(q_0,x) \in F\}$.
↳ language of automata.

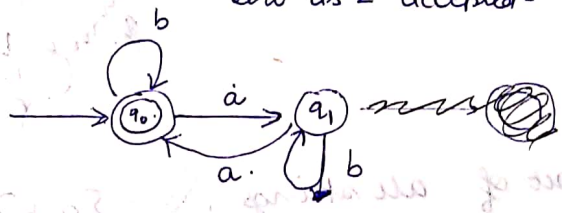
Ex $\Sigma = \{a,b\}$
$L = \{ab, abb\}$
$F =$
$Q =$



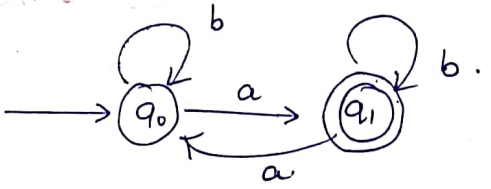
Any string that is not ab or abb eventually ends up in trap state. (Trap state)

Number of 0's = even.
zero a's = accepted.
$\Sigma = \{a,b\}$.

DFA

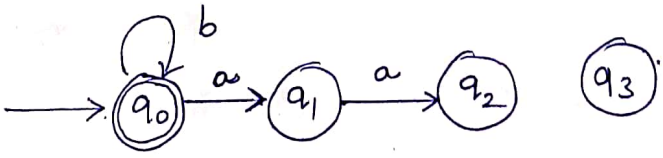


Number of a's = odd



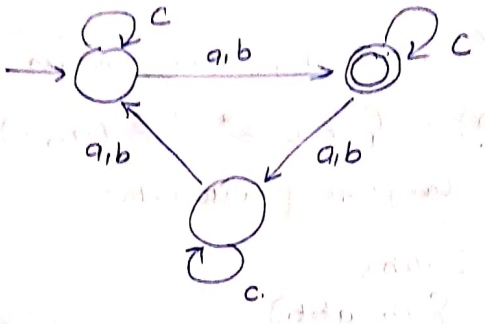
Number of a's = even
Number of b's = odd.

States = 4.



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

$L = \{x \in \Sigma^+ \mid |x|_a + |x|_b = 2 \bmod 3\}$



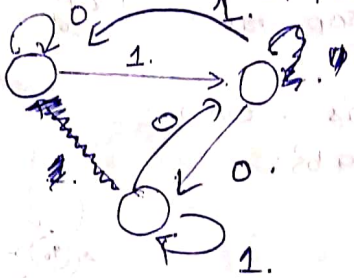
$\Sigma = \{0, 1\}$

Set of all strings such that binary rep. is a multiple of 3 -

Number of states → 3

at the end of reading remainder → 0, 1, 2. ∴ 3 states

11 ✓
110
111 ✗
1001 ✓



101

!!

set of all strings, $\{\Sigma = \{a, b\} \mid |x| = 2 + 3n\}$.

$L = \{a^m b^n \mid m \geq 1 \quad n \geq 0\}$.

set of all strings with the

$\{a^n b^n \mid n$

Note: Lang. acc

set of all strings, $\Sigma = \{a, b\}$, begins and ends with the same symbol.

$\{ a^n b^n \mid n \geq 0 \}$ → DFA not possible.

NOTE: Lang. accepted by DFA is regular.

Conversely :-

$x \in L(N')$

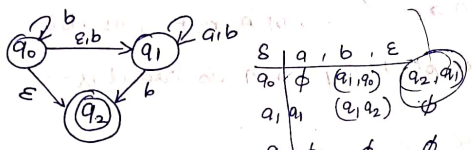
$\hat{\delta}'(q_0, x) \cap F' \neq \phi$

if $\hat{\delta}'(q_0, x) \cap F \neq \phi$

$q \in \hat{\delta}'(q_0, x)$ such that $E(q) \cap F \neq \phi$

$\Rightarrow q \in \hat{\delta}(q_0, x)$ and $E(q) \cap F \neq \phi$

Ex



δ	a	b	ε
q_0	ϕ	$\{q_1, q_0\}$	$\{q_2, q_1\}$
q_1	q_1	$\{q_1, q_2\}$	ϕ
q_2	ϕ	ϕ	ϕ

Construct δ such that no ε transitions -

$F' = F \cup \{q \in Q \mid E(q) \cap F \neq \phi\}$

$\hat{\delta} = \delta'$	a	b
By def - q_0	$\{q_1\}$	$\{q_0, q_1, q_2\}$
q_1	$\{q_1\}$	$\{q_1, q_2\}$
q_2	ϕ	ϕ

Final state in is final state in F'

$\delta'(q_0, a) = \hat{\delta}(q_0, a)$

$= E\left(\delta\left(\hat{\delta}(q_0, \varepsilon), a\right)\right)$

q_0, q_1, q_2

$\phi \quad q_1 \quad \phi$

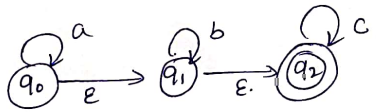
$E\left(\delta\left(\hat{\delta}(q_0, \varepsilon), b\right)\right)$

q_0, q_1, q_2

$q_1 \; q_0 \; q_1 \; q_2 \; \phi$

$q_1 \; q_0 \; q_2$

Ex



δ	a	b	c	ε
q_0	q_0	ϕ	ϕ	q_1
q_1	ϕ	q_1	ϕ	q_2
q_2	ϕ	ϕ	q_2	ϕ.

(q_0, c)

$\delta'(q_0, a) = \hat{\delta}(q_0, a)$

$= E(\delta(\hat{\delta}(q_0, \varepsilon), a)$

$\delta(q_1, b)$ $\quad E(\delta(q_1, a))$

$E(q_1)$ $\quad E(\phi)$

δ'	a	b	c
q_0	$\{q_0, q_1, q_2\}$	$\{q_1, q_2\}$	$\{q_2\}$
q_1	ϕ	$\{q_1, q_2\}$	$\{q_2\}$
q_2	ϕ	ϕ	$\{q_2\}$ -

Given a regular expression, How to construct a language ?

Basis → $\phi \rightarrow \phi$; $\varepsilon \rightarrow \{\varepsilon\}$; $a \rightarrow \{a\}$; $a \in \Sigma$

$r_1 \rightarrow$ ✓ ; $r_2 \rightarrow$ ✓ ; then $r_1 + r_2$ ✓

Final state for ϕ → No final state.

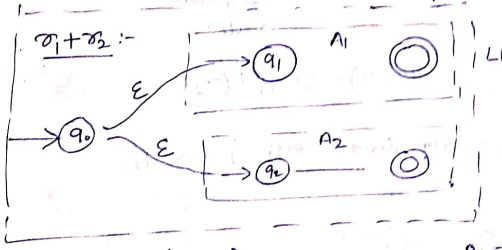
Final state for ε →

Singleton a →

Inductive hypothesis → DFA for r_1 & r_2 we have.

Inductive step → $r_1 r_2$, $r_1 + r_2$, r_1^* } construct a DFA for these.

$r_1 + r_2$:-



$L(A_1) = L(r_1)$

$L(A_2) = L(r_2)$.

$L(r_1 + r_2)$

$A_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$

$A_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$

assume $Q_1 \cap Q_2 = \phi$

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

$F = F_1 \cup F_2$

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

$$\delta(q,a) = \begin{cases} \delta_1(q,a) & \text{if } q \in Q_1 \\ \delta_2(q,a) & \text{if } q \in Q_2 \\ \{q_1, q_2\} & \text{when } q = q_0 \text{ and } a = \varepsilon. \end{cases}$$

if $x \in L(A)$

then $x \in L(A_1)$ or $L(A_2)$

without loss of generality, assume it follows q_1 wala path. Then it continues along -- but then $A_1 \in A$

$\therefore$ # $x \rightarrow L(A_1)$.

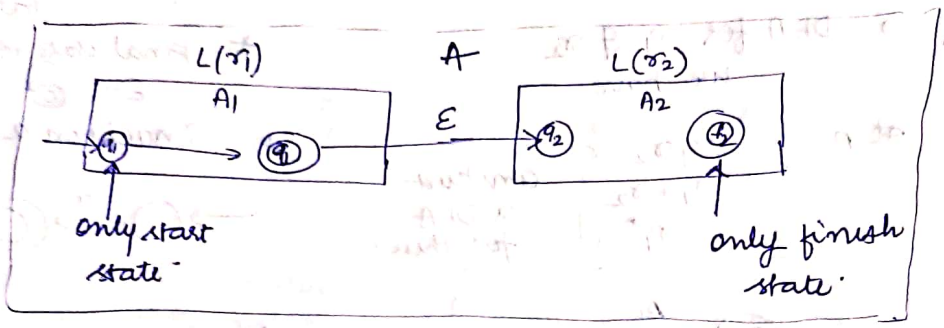
For any regular expression there exist an equivalent finite automaton.

$$L(r) = L(A)$$

$r_1 \checkmark \longrightarrow A_1$

$r_2 \checkmark \longrightarrow A_2$

Finite automaton for $r_1 r_2$

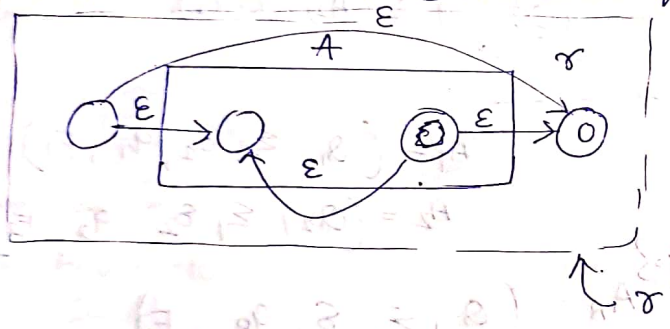


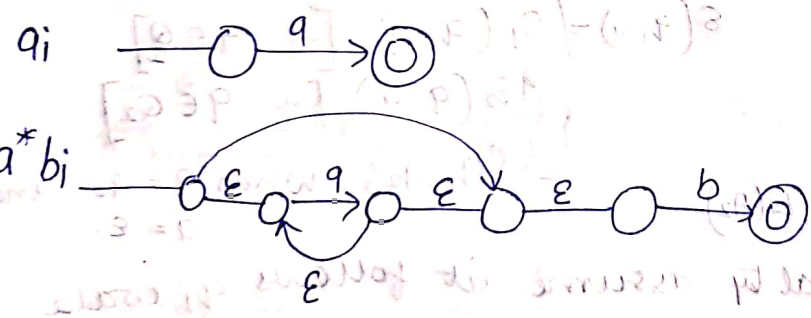
$Q = Q_1 \cup Q_2$

$F = F_2$

$L(r_1 r_2) = L(r_1)\, L(r_2)$

Only extra transition is ε transition from $f_1 \longrightarrow q_2$.





For every regulas expression there exist an equivalent finite automaton.

---

for every DFA there exists an equivalent regular expression.

→ consider a DFA.

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

$Q = \{q_0, q_1 \cdots q_n\}$

$F = \{q_{f_1}, q_{f_2}, q_{f_3} \cdots q_{f_K}\}$. → Finite number of final states

$R_i = \{x \in \Sigma^* \mid \hat{\delta}(q_0, x) = q_i\}$.

↳ set of all strings over Σ^* as we go from $q_0 \to q_i$

Say R_i set can be expressed by regular expression r_i

$$L(A) = r_{f_1} + r_{f_2} + r_{f_3} \cdots r_{f_K}$$

$L(A)$ → Language of DFA.

r_{f_1} → Regular exp. for $q_0 \to f_1$

r_{f_2} → Reg exp. for $q_0 \to f_2$

$R_j \Rightarrow$ Set of all strings that go from $q_0 \to q_j$

S_{ij} → Path from $q_i \to q_j$

$$R_j = R_0$$

$$r_j = r_0 S_{0j} + r_1 S_{1j} + r_2 S_{2j} \cdots + r_n S_{nj} \cdots + \varepsilon$$

general expression.

$$r_0 = r_0 S_{00} + r_1 S_{10} + r_2 S_{20} \cdots r_n S_{n0} \cdots + \varepsilon$$

$$r_1 = r_0 S_{01} + r_1 S_{11} + r_2 S_{21} \cdots r_n S_{n1} + \cdots$$

$$r_2 = r_0 S_{02} + r_1 S_{12} + r_2 S_{22} \cdots r_n S_{n2} + \cdots$$

ε → start state.

$$\vdots$$

$$r_j$$

$$\vdots$$

$$r_n = r_0 S_{0n} + r_1 S_{1n} \cdots r_n S_{nn} + \cdots$$

s, t

$r = t + rs$

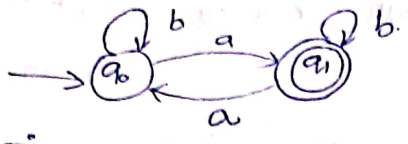
$r = t + rs$.

$r = ts^*$

Arden's Principle :-

$r = ts^*$

Ex



Thm is basically odd number of a's

Two equations — r_0 r_1

$r_0 = r_0 b + (r_1 a + \varepsilon)$

$r_1 = r_0 a + r_1 b$

$r_1 = r_1 b + r_0 a$

$r = rs + t$

$r = ts^*$

$r_1 = (r_0 a) b^*$

$r_0 = \left(\varepsilon + (r_0 a) b^*\right) b^*$

$r_0 = b^* + r_0 a b^*$

$r = t + rs$

$r = ts^*$

$r_0 = b^*(ab^*)^*$

Arden's Principle :-

$r_0 = (\varepsilon + r_1 a) b^*$ ✓

$r_1 = (\varepsilon + r_1 a) b^* a + r_1 b$ ✓

$\frac{r_0(1-b) - \varepsilon}{a} = r_1$

$\frac{r_0(1-b) - \varepsilon}{a} = r_0 a + r_1 b$

$r_0 - b r_0 - \varepsilon = r_0 a^2 + r_1 b a$

$r_0 [1 - b - a^2] = \varepsilon + [r_0(1-b) - \varepsilon] b$

$r_0 [1 - b - a^2] = \varepsilon + b r_0 - r_0 b^2 - \varepsilon b$

Proof of this construction :-

Thm: If A is a DFA then L(A) can be represented by a regular expression -

Proof :- $A = (Q, \Sigma, \delta, q_0, F)$.

$|Q| = n$

Apply induction on no. of states.

Basis :- $n = 1$ $[\to q]$

No. of final states = 2 $\begin{cases} F = \phi \to r = \phi \\ F = q \to r = \Sigma^* \end{cases}$

Both these are reg exp. Hence thm true -

I.H :- Say its true $\forall$ k such that $k < (n-1)$.

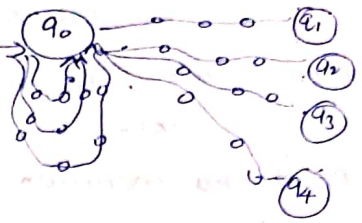
I.S :- $L(A) = L_1^* L_2$

L_1^* → The set of all strings such that you start from q_0 and end at q_0.

L_2 → Start state → final state - [there is no loop from $q_0 \to q_0$ here as that would be included in L_1^*.]

$P_{(q,x)}$ ⇒ set of all states that occur on the path from q while processing the string x.

$q \in Q$
$x \in \Sigma^*$



$x = a_1 a_2 \cdots a_n$.

$$P(q,x) = \bigcup_{i=1}^{n} \hat{\delta}(q, a_1 a_2 \cdots a_i)$$

All we need to show is that L_1 and L_2 have reg. expressions.

$$L_1 = \{x \in \Sigma^* \mid \hat{\delta}(q_0, x) = q_0\}.$$

$$L_2 = \begin{cases} L_3 & \rightarrow \text{if } q_0 \notin F \\ L_3 \cup \{\varepsilon\} & \rightarrow \text{if } q_0 \in F. \end{cases}$$

$$L_3 = \{x \in \Sigma^* \mid \hat{\delta}(q_0, x) \in F\}, \; q_0 \notin P_{(q_0,x)}\}.$$

q_0 should not occur on this path

$$L(A) = L_1^* L_2.$$

claims :-

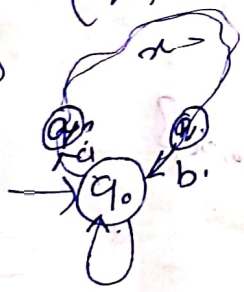
$L_1 \rightarrow$ is regular

$L_2 \rightarrow$ is regular

Proof of these claims :-

$$A'' = \left\{ (q,b) \in \Sigma^* \Sigma \;\middle|\; \begin{array}{l} \hat{\delta}(q_0, axb) = q_0 \; \forall x \in \Sigma^* \\ \delta(q_0, a) \neq q_0 \\ q_0 \notin P_{(q_a, x)} \end{array} \right. \text{ where } q_a = \delta(q_0, a)$$

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



q_0 does not appear on this path.

collect all such pairs (a,b) and call it

$$L_{(a,b)} = \{x \in \Sigma^* \mid \hat{\delta}(q_0, axb) = q_0 \text{ and } q_0 \notin P_{(q_a, x)} \text{ when } q_a = \delta(q_0, x)\}$$

set of all such strings x defines language $L_{(a,b)}$

$L_{(a,b)}$

$A_{(a,b)} = (Q', \Sigma, \delta', q_a, F')$

$Q' = Q - \{q_0\}$

$F' = \{q \in Q' \mid \delta(q,b) = q_0\}$

$\delta' = \delta$ [same as the DFA but restricted domain of Q'].

Theorem If A is a DFA, then L(A) is defined by a regular expression

Proof :- $A = (Q, \Sigma, \delta, q_0, F)$

$|Q| =$

Proof : by induction on

Claim :- L_1 is regular

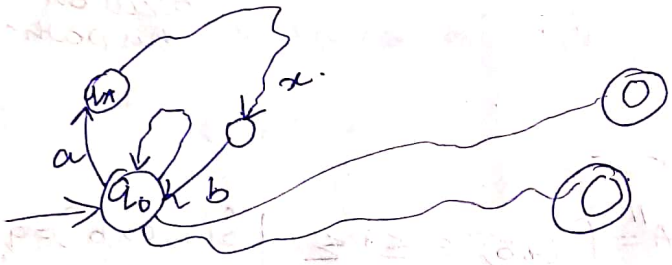
For each (a,b) define $L_{(a,b)} = \{x \in \Sigma^* \mid \hat{\delta}(q_0, axb) = q_0\}$

$A_{(a,b)} = (Q', \Sigma, \delta', q_x, F')$

$Q' = Q - q_0$.

$F' = \{q \in Q \mid \delta(q,b) = q_0\}$.

$\delta' = \delta|_{Q' \times \Sigma}$.



$L(A) = L_1^* L_2$

$L_1 = \{x \in \Sigma^* \mid \hat{\delta}(q_0, x) = q_0\}$.

$L_2 = \begin{cases} L_3 & \text{if } q_0 \notin F. \\ L_3 \cup \{q\} & \text{otherwise} \end{cases}$

$L_3 = \{x \in \Sigma^* \mid \hat{\delta}(q_0, x) \in F, q_0 \notin P(q_0, x)\}$.

Thm $x \in L(A_{(a,b)})$

$\hat{\delta}(q_0, axb) = \hat{\delta}(\delta(q_0, a), xb)$

$= \hat{\delta}(q_a, xb)$

$= \delta'(\hat{\delta}(q_a, x), b)$

$= \delta(p, b) \quad p \in F'$

$= q_0$.

$x \in L(a,b)$

$$L_1 = B \cup \bigcup_{(a,b)\in A} a L_{(a,b)} b.$$

$$B = \{ a \in \Sigma \mid \delta(q_0, a) = q_0 \}.$$

claim :- L_3 is regular.

$$C = \{ a \in \Sigma \mid \delta(q_0, x) \in q_0 \}.$$

$a \in C$

$$L_a = \{ x \in \Sigma^* \mid \hat{\delta}(q_0, ax) \in F,\ q_a \in P_{(q_0,x)} \}.$$

we show that L_a is a regular ~~lang~~ language.

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

$$Q' = Q - \{q_0\}.$$

$$F'' = F - \{q_0\}$$

$$L(A_a) = L_a.$$

$$x \in L(A_0) \iff \hat{\delta}(q_0, x) \in F''$$
$$\iff \hat{\delta}(q_0, x) \in F.$$
$$\iff \hat{\delta}(q_0, x) \in F.$$
$$\iff \hat{\delta}(\delta(q_0, a), x) \in F.$$
$$\iff \hat{\delta}(q_0, ax) \in F.$$
$$\iff x \in L_a.$$

$$C = \{ a \in \Sigma \mid .$$

Regular Grammar :-

(Right linear)

$$G = (N, \Sigma, P, S).$$

$A \to xB$ where $x \in \Sigma^*$

OR $A \to x$

claim :-

Rg Grammar = DFA.

Part I :- If A is a DFA, Then L(A) can be generated by a regular (right linear) grammar.

Proof :- $A = (Q, \Sigma, \hat{\delta}, q_0, F)$

$G_1 = (N, \Sigma, P, S)$

$N = Q$

$S = q_0$

$P = \{ A \to aB \mid \delta(A,a) = B \} \cup \{ A \to a \mid \delta(A,a) \in F \}$

{ if B is a final state then the production rule is different }

In addition, if the initial state is a final state $q_0 \in F$ & then.

$S \to \varepsilon$ in P.

Note :-

Say $x \in L(A)$

$x = a_1 a_2 \text{---} a_n$

corresponding to all these states there are production rules

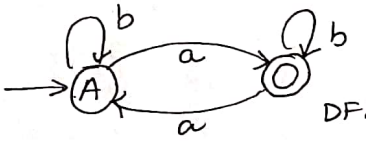
$\hat{\delta}(q_0, a_1 a_2 \text{---} a_n) \in F$

$\hat{\delta}(q_0, a_1) = q_1 \quad q_0 \to a_1 q_1$

$\delta(q_1, a_2) = q_2 \quad q_1 \to a_2 q_2$

$\delta(q_{n-1}, a_n) = q_n \in F \quad (q_{n-1} \to a_n q_n$ disposed as it is final state)

Ex



DFA for odd no. of a's.

Regular grammar :-

given a regular grammar G_1, construct a DFA

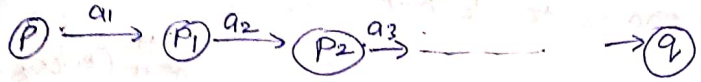
$G_1 (N, \Sigma, P, S)$

$A \to aB \qquad \delta(A,a) = B$

only if all productions are of the form $A \to aB$.

The only difference between the one constructed this way is that this will be a generalized finite automata.

$x = a_1 a_2 a_3 \cdots a_n$.



$G_1 = (N, \Sigma, P, S)$.

$G_{FA} = (Q, \Sigma, X, \delta, q_0, F)$

~~$Q = N \cup \{$~~

$Q = N \cup \{\$\}$

$q_0 = S$

$F = \{\$\}$.

* $B \in \delta(A, x)$ if and only if $A \to xB \in P$

$\$ \in \delta(A, x)$ if $A \to x \in P$.

[This construction really works]

S_{ij} := ($q_i \to a_j$ path).

R_i := $q_0 \to q_i$

r_j = Reg. exp. for R_j

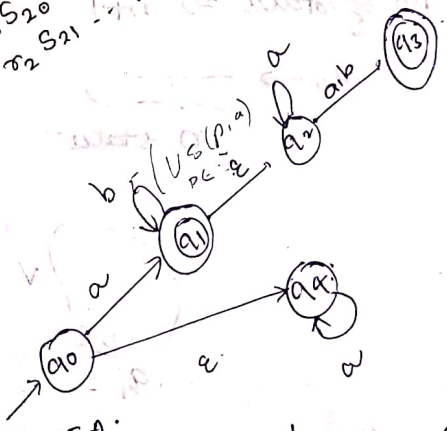
$r_j = r_0 S_{0j} + r_1 S_{ij} + \cdots + r_n S_{nj}$ $\hat{\delta}(\cdot$

$r_0 = r_0 S_{00} + r_1 S_{10} + r_2 S_{20} \cdots r_n S_{n0}$ $r_0 = r_0 b + r_1$

$r_1 = r_0 S_{01} + r_1 S_{11} + r_2 S_{21} \cdots r_n S_{n1}$.

$r_2 = t + r_2 S$.

$r_2 = t S^*$.



NFA → DFA.

How?

$\varepsilon \in L(N)$.

$F' = F \cup \{q \in Q \mid E(q) \cap F \neq \phi\}$.

$L(N) \neq L(N')$. $\hat{\delta}(q_0, \varepsilon) \cap F \neq \phi$: $\varepsilon \in L(N) \Leftrightarrow$ ~~$\hat{\delta}(q_0, \varepsilon)$~~ $E(q_0) \cap F \neq \phi$

$\delta' : Q \times \Sigma \to \$.P(Q)$. $q_0 \in F'$

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

$F' = F \cup \{q \in Q \mid E(q) \cap F \neq \phi\}$.

$(a + ab^+)^* ab^*$

$q_0 \in F'$.

$\therefore q_0 \in L(N)$.

$\hat{\delta}(q, xa) = E\left(\bigcup_{p \in \delta(q,x)} \delta(p, a)\right)$

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

How do we show that a language is not regular?

→ we need some tools to do that :-

Pumping lemma :-

Let L be a regular language.

Then there is a constant n such that if x is any string in L and $|x| \geq n$.

Then we may write $x = uvw$ in such a way that

i) $|u| \geq 1$.

ii) $|uv| \leq n$. and

iii) $\forall i \geq 0 \cdot uv^i w \in L$.

Furthermore n is (equal to the) number of states of the smallest DFA accepting L.

$q_0 \to q_1 \to q_2 \text{ ---------- } q_i \to q_{i+1} \cdots q_k \to q_{k+1} \to q_m$.

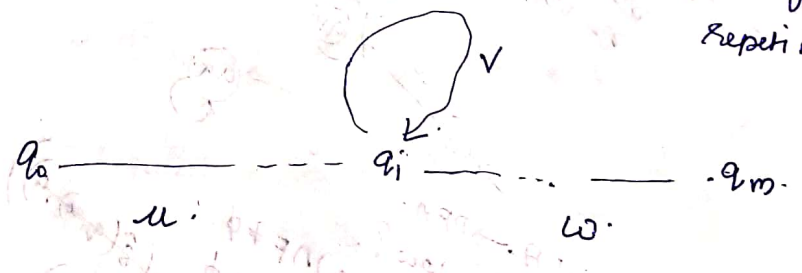
for $|x| \geq n$.

Say, this derivation is $|x|$ is exactly n.

no. of states $\Rightarrow n+1$ ∴ At least one of the states must repeat: say upto q_i, no repetation.

$q_0 \to \underbrace{\text{----}}_{n \text{ states}}$

Say $q_k = q_i$. i.e the repetition -



ii) $uv \leq n$.

i) [q_i is repetition, V cannot repeat].

∴ $|v| \geq 1$

iii) $q_0 \to q_m$: uw, uvw, uv^2w, uv^3w, ! ∴ $uv^iw \in L$.

Use the co

if a

then

Proof :- $\{a^n b$

Fix a con

x

v can not

∴ v

Ex 2 $\{w$

Pumpi

Ex 3 {

Pu

a^m

$a^m =$

$|uv$

Ex {

(Every reg language must sat this c However, a lang may sat this lem but this not

regular?.

every reg. lang. must satisfy this lemma. However, a lang. may satisfy this lemma but still not reg.

Use the contrapositive of this statement for non regularity

if a language does not satisfy Pumping lemma, then it is not regular.

Proof :- $\{a^n b^n \mid n \geq 0\}$. Not regular.. prove.

Fix a constant k. [For pumping lemma].

$x = a^k b^k$ $\quad v = ab$

v cannot be ab as then $|uv| \geq n$

∴ v has to be either a or b. but then it'll be $a^{i+k} b^k$. which does not belong to the lang.

Ex 2 $\{ww \mid w \in (a+b)^*\}$.

Pumping lemma const → k.

$x = a^k b^k a^k b^k$ $\quad |x| = 4k$.

Ex 3 $\{a^p \mid p \text{ is prime}\}$.

Pumping lemma constant → k.

a^m where $m \geq k$. (m → prime)

$a^m = uvw$ → for it to be regular.

$|uvw| = m = uvw$

Ex $\{ww^R \mid w \in (a+b)^*\}$ even length palindrome. Non regular?.

Properties under which Regular languages are closed :-

1) Union $r_1 + r_2$
2) Concatenation $r_1 r_2$
3) Kleen closure r_1^*
4) Complement. ——→ Yes. How so?

How?

$L \subseteq \Sigma^*$ if L is Regular

$\bar{L} = \Sigma^* - L$. then $\bar{L}$ is Regular too-

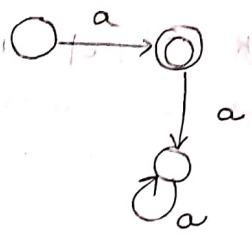
Proof :- L is Regular.

L has a DFA.

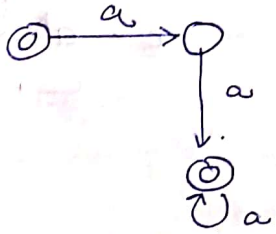
$\bar{L}$ → Final state of L will be nonfinal.

Non final state will be final

if x was not accepted by L., it is accepted by $\bar{L}$



$L(A)$



$L'(A)$

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

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

$x \in L(A)$ if $\hat{\delta}(q_0 x) \in F$

$\Leftrightarrow \hat{\delta}(q_0, x) \notin Q - F$.

5) Intersection

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

Reg Reg. union → Reg ✓

How do you construct DFA for such an intersection ?

$A_1 = (Q_1, \Sigma_1, \delta_1, q_1, F_1)$

$A_2 = (Q_2, \Sigma_2, \delta_2, q_2, F_2)$

$L_1 = L(A_1)$

$L_2 = L(A_2)$

$L = L_1 \cap L_2$

$A = \left((Q_1 \times Q_2), \Sigma, \delta, (q_1, q_2), (F_1 \times F_2)\right)$

$\delta = ?$

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

$a \in \Sigma$, $b \in \Sigma$, $Q_1 \times Q_2$.

Proof :- Take x

$x \in L(A_1)$ & $x \in L(A_2)$

Then $x \in L(A)$

6) Reverse of a language :-

x , x^R.

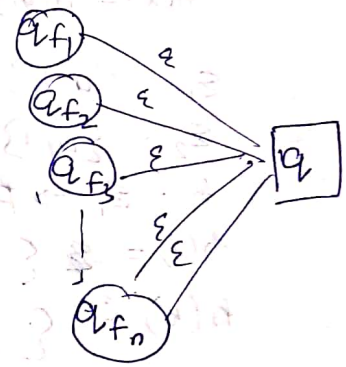
L $\quad L^R = \{ x \in \Sigma^* \mid x^R \in L \}$.

say we have DFA for L. DFA for L^R —

1) make one final state out of all the finals—

2) make all start state final

3) For δ , reverse all arrows—.



7) Right quotient of L_1 by L_2

$L_1, L_2 \in \Sigma^*$

$L_1/L_2 = \{ x \in \Sigma^* \mid \exists y \in L_2 \text{ s.t. } xy \in L_1 \}$

Ex $L_1 = \{a, ab, bab, baba\}$

$L_2 = \{a, ab\}$

$L_1/L_2 = \{\varepsilon, bab, b\}$.

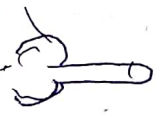
Ex $L_1 = 10^*$.

Regular langs are closed under :—

↳ Union
↳ Concatenation
↳ Klean closure
↳ complement
↳ Intersection
↳ Reversal
↳ Difference
↳ Right Quotient.
↳ Substitution.
& Homomorphism

Regular languages are closed under right quotient :-

If L is regular then so is L/L' for any L'.



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

$L(A) = L$

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

$F' = \{q = Q \mid \hat{\delta}(q, x) \in F \text{ for some } x \in L'\}$

$L(A') = L/L'$

8)

Homomorphism :-

$\Sigma_1 \qquad \Sigma_2$

$h : \Sigma_1^* \longrightarrow \Sigma_2^*$ $\qquad h(xy) = h(x)\,h(y)$

$h(\varepsilon) = \varepsilon$

$x = a_1 a_2 a_3 a_4 \ldots a_n \in \Sigma_1^*$

$h(x) = h(a_1)\,h(a_2) \ldots h(a_n)$.

$\Sigma_1 = \{a, b\}$

$\Sigma_2 = \{0, 1\}$

$h : \Sigma_1 \longrightarrow \mathcal{P}(\Sigma_2^*)$.

$h(a) = \{0^n \mid n \geq 0\} = L_1$

$h(b) = \{1^n \mid n \geq 0\} = L_2$

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

$L_1 = \{x_1, x_2 \ldots x_k\}$

$\downarrow$

$a_1 a_2 \ldots a_n$

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

$h(L) = 0^* 1^*$

For every x in Σ, if substituted by a regular language

If L is regular, an r must exist -

$L \longrightarrow r$ (Reg. exp.)

$L = L(r) \quad \forall x \in R$.

DFA mini

For $a \in \Sigma$

$r = \delta(p, a)$

$s = \delta(q, a)$

say r and s

be

Then what

Distingu

[we

DFA minimization :-

$p \equiv q$ indistinguishable or equivalent.
distinguishable or non equivalent.

$\forall x \in \Sigma^*$

$\hat{\delta}(p, x)$ → final

$\hat{\delta}(q, x)$ → final.

Equivalent states in a DFA if

For $a \in \Sigma$

$r = \delta(p, a)$

$s = \delta(q, a)$

say r and s are known to be distinguishable.

Then what about p, q ?

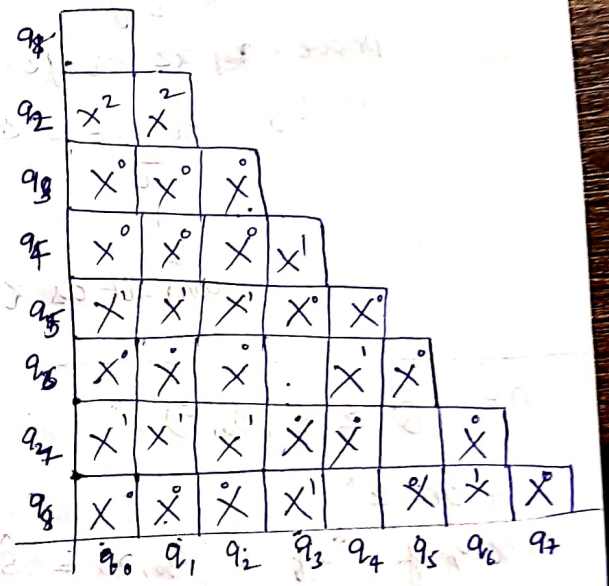
Distinguishable by ax

distinguishable [with $x = \varepsilon$]. < $p \to$ final, $q \to$ non final.

[we can merge indistinguishable pairs to minimize DFA]

δ	a	b
→ q_0	q_3	q_2
q_1	q_0	q_2
q_2	q_8	q_5
(circled) q_3	q_6	q_1
(circled) q_4	q_2	q_5
q_5	q_4	q_3
(circled) q_6	q_1	q_0
q_7	q_4	q_6
(circled) q_8	q_2	q_4
q_9	q_7	q_{10}
q_{10}	q_5	q_7

$\{q_0, q_3, q_2, q_1, q_5, q_8, q_4, q_7\}$

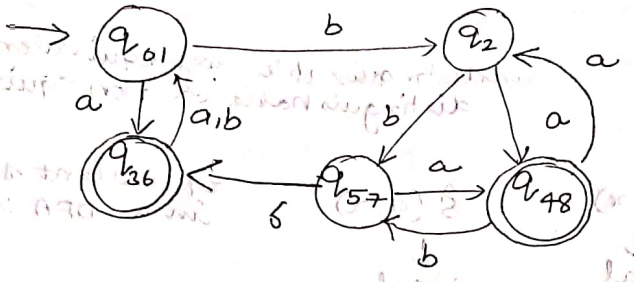


$\delta(q_0, b) = q_2$
$\delta(q_5, b) = q_3$ > marked

$\delta(q_0, b) = q_2$
$\delta(q_7, b) = q_6$ > marked.

q_{01} q_{36} q_{48} q_{57}

q_2 ↘ distinguishable with all-



How does this yield minimal DFA ?

An equivalent relation $\sim$ on Σ^* is said to be right invariant if for $x, y \in \Sigma^*$

if $x \sim y \Rightarrow (xz \sim yz)\ \forall z$.

Let L be a language over Σ. Define the relation $\sim_L$ on Σ^* by $x \sim_L y$ iff $\forall z\ (xz \in L \Leftrightarrow yz \in L)$

$\sim_L$ is equivalent relation.

$\sim_L$ is right invariant as well-

How ?

$x \sim_L y$ given $\rightarrow$ $xz \in L \Leftrightarrow yz \in L$

Prove $xz \sim_L yz$.

↳ $xzw \in L \Leftrightarrow yzw \in L$.

(zw = u)

can act as z. $\therefore \sim_L$ right invariant

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

$\sim_A$ on Σ^*

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

$z \in \Sigma^*$

$\hat{\delta}(q_0, xz) = \hat{\delta}(\hat{\delta}(q_0, y), z) = \hat{\delta}(q_0, yz)$.

Myhill Nerode

Let L be a

Σ. The follo

are equi

1) L is acc

2) there exi

relation

unior

3) the equiva

Proof

FOR

equiv

$L =$

say

Becau

Myhill Nerode Theorem (characterization)

Let L be a language over Σ. The following statements are equivalent.

iff condition:
if ✓ then regular
if × then not regular

1) L is accepted by a DFA.
2) There exists a right invariant relation equivalent relation $\sim$ of finite index on Σ^* such that L is the union of some of the equivalent classes of $\sim$.
3) The equivalent relation $\sim$ is of finite index.

Proof ① ⇒ ②

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

$L = L(A)$

for $x \in \Sigma^*$: if $\hat{\delta}(q_0, x) = p$

$[x]_{\sim_A} = \{ y \in \Sigma^* \mid \hat{\delta}(q_0, y) = p \}$.

For $q \in Q$

$C_q = \{ x \in \Sigma^* \mid \hat{\delta}(q_0, x) = q \}$

equivalent class of q

L = Set of strings accepted by DFA.

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

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

$$= \bigcup_{p \in F} C_p.$$

② ⇒ ③ $\sim_L$ is a refinement of $\sim$

i.e no. of equivalent classes of $\sim_L$ is less than no. of eqv. classes of $\sim$.

belong to same equ class

Say $x, y \in \Sigma^+$ s.t $x \sim y$. then $x \sim_L y$.

$x \sim y \Rightarrow \forall z \, (xz \sim yz)$.

$x \sim_L y$

$\forall z \, (xz \in L \Leftrightarrow yz \in L)$

Because xz, yz belong to the same eqv. class, then xz and yz either both belong OR don't ∴ eqv classes of $\sim_L$ are either equal or disjoint.

(3) ⇒ (1) Assume $\sim_L$ is of finite index.
Construct a DFA for L.

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

$Q = \{ [x] \mid x \in \Sigma^* \}$. the set of eqv. classes of Σ^* w.r.t $\sim_L$

corresponding to the eqv. classes of $\sim_L$ we have finite number of final states. These appear in Q

$\delta([x], a) = [xa]$
$\forall [x] \in Q, a \in \Sigma$

How to say δ is well defined —?

$\delta([x], a) = [xa]$

$[x] = [y] \Rightarrow x \sim_L y \Rightarrow xa \sim_L ya$

$\Rightarrow [xa] = [ya]$

$q_0 = [\varepsilon]$

F = set of all those eqv. classes such that

$\{ [x] \in Q \mid x \in L \}$.
↳ accepted by L

∴ A is a DFA.

How to show $L(A_L)$ accepts L. i.e $L(A_L) = L$

$\hat{\delta}(q_0, w) = [w] \quad \forall w \in \Sigma^*$

$w \in L \Leftrightarrow [w] \in F$.

Apply induction on length of w —

$|w| = 0 \quad w = \varepsilon$.

Basis: $\hat{\delta}(q_0, \varepsilon) = q_0 = [\varepsilon]$ ← class of a

$\hat{\delta}(q_0, a) = \hat{\delta}([\varepsilon], a) = [\varepsilon a] = [a]$

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

Induction Hypothesis

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

$= \delta([x], a)$

$= [xa]$

↳ Basis — ∴ Done YAY. ☺

Scanned by CamScanner

$\Rightarrow$ Index $(\sim_L) \leq$ Index $(\sim_A)$ [index for $\sim$ is the number of eqv. classes of $\sim$].

$\therefore A_L$ is the minimal DFA accepting L.

$\sim_A$

$x \sim_L y$ iff $(xz \in L \iff yz \in L)$

$L = \{x \in \{a,b\}^* \mid ab \text{ is a substring of } x\}$.

Equivalent classes:-

1. $[\varepsilon]$
2. $[a]$
3. $[ab]$

> They don't belong to same eq. class because if you take $z = b$, then $xz = b \rightarrow$ does not. $yz = ab \rightarrow$ belongs to L

Let $x \in \{a,b\}^*$

either x contains ab as substring / ab is not a substring

↓ belongs to class 3

↓

a^n — (2)

b^n — (1)

$b^m a^n$ → (2)

Only three eqv. classes.

3 states is minimal number of states for any DFA.

Show that $L = \{a^n b^n \mid n \geq 0\}$ is non regular.

→ By Myhill Nerode Thm, if L is infinite eqv classes, non regular.

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

✓ ← $a^n \longrightarrow b^n$.

✗ ← $a^m \nearrow$ $m \neq n$

$\therefore a^n$ and a^m belong to diff classes. ∞ such classes

$\therefore$ Not regular.

Thm For every DFA 'A', there is an equivalent minimum state DFA A'

Proof:- $A = (Q, \Sigma, \delta, q_0, F)$

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

$Q' = \{[q] \mid q \text{ is accessible form of } q_0\}$.

The eqv. classes of Q wrt $\equiv$ that contains states accessible from q_0.

$[q_0'] = [q_0]$

$F' = \{[q] \in Q \mid q \in F\}$.

$\delta' = Q' \times \Sigma \to Q'$

$\delta'([q], a) = [\delta(q, a)]$

$p \equiv q$

$[p] = [q]$

$\hat{\delta}(\delta(p, a), x)$ $\quad \hat{\delta}(p, ax)$

$\hat{\delta}(\delta(q, a), x)$ $\quad \hat{\delta}(q, ax)$

Therefore $\hat{\delta}'$ is well defined -

How to prove that $L(A) = L(A')$

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

$\forall x \in \Sigma^*$

can prove by induction on length of string x.

$$\hat{\delta}'[[q_0], xa) = \delta'(\hat{\delta}'([q_0], x), a)$$
$$= \delta'([\hat{\delta}(q_0, x)], a)$$
$$= [\delta(\hat{\delta}(q_0, x), a]$$
$$= [\hat{\delta}(q_0, xa)]$$

How to show that this is minimal ?

Claim:2 A' is minimal state DFA accepting L.

The index of $\sim_L$ = index of $\sim_{A'}$

Index of $\sim_L \leq$ index of $\sim_{A'}$.

Proof by contradiction. say index of $\sim_L \geq$ index of $\sim_{A'}$

$\exists\ x, y \in \Sigma^*$ s.t

$x \sim_L y$ but $x \not\sim_{A'} y$

Related by $\sim_L$ but not by $\sim_{A'}$.

Not possible why ?

$$\hat{\delta}'[[q_0], x)] \neq \hat{\delta}'([q_0], y)$$

$$p = [\hat{\delta}([q_0], x)] \neq [\hat{\delta}(q_0, y)] = q.$$

Finite Auto

moore machine

$\lambda: Q \to \Delta$

say $x =$

Output

$\Sigma = \{0, 1$

$\Delta = \{0, 1$

Mealy M

(output

funct

transiti

#

Finite Automata with Output :-

Moore machine :- $M = (Q, \Sigma, \Delta, \lambda, \delta, q_0)$ → Transition fn.

λ produces output seq.

$\lambda : Q \rightarrow \Delta$

Σ → input alphabet; Δ → output alphabet.

say $x = a_1 a_2 \cdots a_n$ $\quad a_i \in \Sigma$.

output → $\lambda(q_0)\,\lambda(q_1)\,\lambda(q_2) \cdots \lambda(q_n)$

where $\delta(q_0, a_1) = q_1$

$\delta(q_1, a_2) = q_2$

$\delta(q_2, a_3) = q_3$

$\delta(q_{n-1}, a_n) = q_n$.

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

$\Delta = \{0, 1, 2\}$

I/P → 0 0 1 0 1

O/P → 0 0 0 1 2 2

If the input seq. is a binary int. this moore machine gives ~~##~~ residue module 3.

Mealy Machine :-

$M = (Q, \Sigma, \Delta, \lambda, \delta, q_0)$

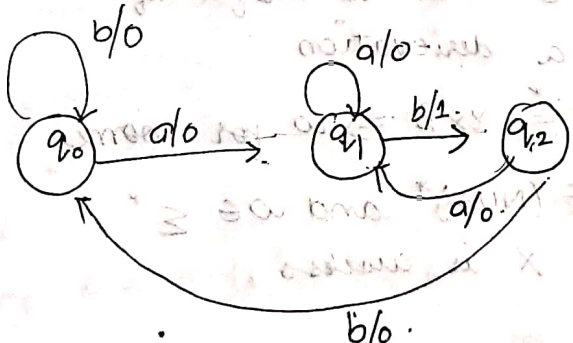
Associate output with all transitions --

[No final state concept]

(output function) $\lambda : Q \times \Sigma \rightarrow \Delta$.

transition δ

#



δ	a	b
q_0	q_1	q_0
q_1	q_1	q_2
q_2	q_1	q_0

As soon as string ab occurs → produces 1.

λ	a	b
q_0	0	0
q_1	0	1
q_2	0	0

2- DFA [The read head can move to either side] - (right or left.

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

$q_0 w \vdash^* wp$ Read all symbols and eventually reach $p \in F$

$L(N) = \{ w \in \Sigma^* \mid q_0 w \vdash^* wp \text{ for some } p \in F \}$.

equivalent to DFA.

Claim :- Let there be a ~~language~~ DFA M.

if M has n states;

m is finite if M accepts only strings with length $< n$

M is infinite if - $\exists\, x \in \Sigma^*$ $x = uvw$

$n \leq |x| < 2n$ (Pumping demma)

Take x with $n \leq |x| < 2n$ if x is accepted by α, infinite.

~~If~~ If finite, check for w with $|w| < n$.

Now if w is accepted, non empty language-

CFG

$G = (\omega, \Sigma, P, S)$

$A \rightarrow \alpha \;\; \alpha \in (N \cup \Sigma^*)$

Simplification of CFG :-

Useless Symbol :- $G = (N, \Sigma, P, S)$

A symbol $x \in N \cup \Sigma$ is useful if there is a derivation

$S \overset{*}{\Rightarrow} \alpha x B \overset{*}{\Rightarrow} w$ for some

$\alpha, B \in (N \cup \Sigma)^*$ and $w \in \Sigma^*$

Otherwise X is useless.

For X ⇒ non terminal :-

1) $S \overset{*}{\Rightarrow} \alpha X B$ (occurs in a derivation)

2) $X \overset{*}{\Rightarrow} x$.

(But even if X is satisfying these 2 conditions, it might be useless.

Step 1 :- Remove an alp

Step 2 :- Consid Reach

$G_1 = (N$

$G' (N$

Generating non t

Algorithm :-

1) old N ←

2) New N

3) while

4)

5)

6) N' ←

Reachable Set:

$N' \cup \Sigma'$

Step 1: And S

Generating Set

Step 1:- Remove the non terminals that don't produce an alphabet / terminal symbol.

Step 2:- Consider all non terminals that are not reachable from start state and remove them too.

$G = (N, \Sigma, P, S)$

$G' (N', \Sigma', P', S')$ $\quad A \in N'$, $A \Rightarrow^{*} x$

Generating non terminals: 1) all A's such that $A \to x$ production exist.

2) $A \to x_1 x_2 \dots x_j \dots x_k$.

For A to be generating, either x_i should be a terminal, or should belong to the generating set.

Algorithm:- to find Generating set:-

1) old $N \leftarrow \phi$
2) New $N \leftarrow \{ A \mid A \to x \in P$ for some $x \in \Sigma^* \}$.
3) while old $N \neq$ new N do
4) old $N \leftarrow$ new N
5) new N = old $N \cup \{ A \mid A \to \alpha \in P$ for some α in $(\Sigma \cup$ old $N)^* \}$
6) $N' \leftarrow$ New N.

Reachable Set:-

$G = (N, \Sigma, P, S)$

$G' = (N', \Sigma', P', S)$

For every $X \in (N' \cup \Sigma')$ for which $S \Rightarrow^{*} \alpha X \beta$ for $\alpha, \beta \in (N' \cup \Sigma')^*$

$N' \cup \Sigma' = \{ S, \dots$

Step 1: Include S itself

$A = x_1 x_2 \dots x_k$

Say $A \to \alpha_1 \mid \alpha_2 \dots \mid \alpha_k$.

$\forall \alpha_i$, either α_i is a terminal reachable from S, $\in \Sigma'$

or is a non terminal reachable from S. $\in N'$

Generating Set → Reachable Set → (we reach a set that has only useful symbols-

$S \to AB \mid a$
$A \to a$

$\to$ $S \to a$
$A \to a$

$\to$ $S \to a$

✓ Generating then reachable.

$S \to AB \mid a$
$A \to a$

$S \to A \mid a$
$A \to a$

✗ (Reachable then generating)

Ex $S \to aS \mid A \mid c$
$A \to a$
$B \to aa$
$C \to acB$

Gen $= \{A, B, S\}$
$\not\in$ Gen $\to A \to a$.

C (recursive) ✗

$S \to AS \mid A$
$A \to a$
$B \to aa$.

Reachable

$S \to AS \mid A$
$A \to a$.

Theorem:- For any CFG G. $G = (N, \Sigma, P, S)$
There is a CFG G' with no. ε production or unit production such that

$$L(G') = L(G) - \{\varepsilon\}$$

Proof:- $\hat{P}$ $\hat{G}$

1) If $A \to \alpha BB$ and $B \to \varepsilon$ are in $\hat{P}$
Then $A \to \alpha B$ is in $\hat{P}$ and.

2) If $A \to B$ and $B \to \gamma$ are in $\hat{P}$, then $A \to \gamma$ is in $\hat{P}$

$$L(G) \subseteq L(\hat{G})$$
$$L(\hat{G}) \subseteq L(G)$$

Whatever is derievable under $L(\hat{G})$ is derievable under $L(G)$ Therefore $L(\hat{G}) = L(G)$

$$S \overset{*}{\Rightarrow} \alpha_1 B \alpha_2 \overset{*}{\Rightarrow} \alpha_1 \alpha_2 \overset{*}{\Rightarrow} x.$$

$$A \to \alpha BB$$

⊙⊙ $S \Rightarrow \gamma A S \xRightarrow{1} \gamma\alpha B B S \xRightarrow{m} \alpha_1 B \alpha_2 \xRightarrow{n} \alpha_1\alpha_2 \xRightarrow{k} x$

$(m+n+k+1)$

$\therefore B \to \varepsilon$ is a useless production.

From $\hat{G}$, throw out all unit productions and all ε productions.

Standard/Normal Form of CFG

CNF → Chomsky Normal Form.

$A \to BC$
$A \to a$

$G = (N, \Sigma, P, S)$

$A \to x_1 x_2 \cdots x_k$ $\quad x_i$ = terminal or non terminal.

→ x_i = non terminal → dont do anything

→ x_i = terminal. say a. Introduce a new non terminal $N' = N \cup \{A_a, \ldots\}$ to the Grammar

Replace by A_a in the X and add a production.

$X = x_1 x_2 \cdots x_i \cdots A_a \cdots$

$A_a \to a$

Do this to every production

$A \to B_1 B_2 \cdots B_k$

$A \to B_1 C_1$

$C_1 \to B_2 B_3 \cdots B_k$.

$C_1 \to B_2 C_2$

$C_2 \to B_3 \cdots B_k$.

Continue till RHS of all is 2

GNF (Greibach Normal Form).

$A \to a B_1 B_2 \cdots B_k \quad k \geq 0$.

Lemma 1:- Let G be (N, Σ, P, S) be a CFG.

Let $A \to \alpha_1 B \alpha_2$ be a production in P and

$B \to B_1 \mid B_2 \mid \cdots B_r$ be the set of all B productions

Let $G_1 = (N_1, \Sigma_1, P_1, S)$ be obtained from G by deleting the prodn $A \to \alpha_1 B \alpha_2$ from P and adding productions $A \to \alpha_1 B_1 \alpha_2 \mid \alpha_1 B_2 \alpha_2 \mid \cdots \mid \alpha_1 B \alpha_2$. Then $L(G) = L(G_1)$

Lemma 2:- Let $G_1 = (N, \Sigma, P, S)$ be a CFG. Let $A \to A\alpha_1 \mid A\alpha_2 \mid \cdots \mid A\alpha_r$ be the set of all A-productions for which A is the leading symbol in the RHS. Let $A \to B_1 \mid B_2 \cdots \mid B_s$ be the remaining A-productions. Let $G_2 = (N \cup \{B\}, \Sigma, P_2, S)$ be the CFG formed by adding the non terminal B to N and replacing all A-productions by the productions

1) $A \to \beta_i$, $A \to \beta_i B$ $\quad 1 \le i \le s$

2) $B \to \alpha_i$, $B \to \alpha_i B$ $\quad 1 \le i \le r$

Proof:-

$$A \Rightarrow A\alpha_{i_1} \underset{G_1}{\Rightarrow} A\alpha_{i_2}\alpha_{i_1} \cdots \Rightarrow A\alpha_{i_p}\alpha_{i_{p-1}} \cdots \alpha_{i_1}$$

$$\cdots \Rightarrow B\alpha_{i_p}\alpha_{i_{p-1}} \cdots \alpha_{i_1}$$

~~$A \Rightarrow B$~~

$$A \Rightarrow \beta_j B \Rightarrow \beta_j \alpha_{i_p} B$$

Theorem:- Every CFL w/o ε can be generated by a grammar for which every production is of the form $A \to a B_1 B_2 \cdots B_k$ $k \ge 0$

Proof

$A \to BC$, $A \to a$ $\quad G_1 = (N, \Sigma, P, S)$

$N = \{A_1, A_2, \cdots, A_m\}$

Step 1:- $A_i \to A_j \gamma$ $\quad j > i$

$i \le i < k$ $\quad A_k$ - productions.

$A_k \to A_j \gamma$ $\quad j < k$

Replace with RHS of A_j productions - (apply lemma 1.) Repeat this for $k-1$ times and $A_k \to A_l \gamma$ $\quad l \ge k$.

this is bound to happen at some point.

$A_j \to A_i \gamma$ → $i > j$

Apply lemma 1

new non ter

A

Step 2:- $A_i \to A_j \gamma$

$A_i \to a\gamma$

$B_i \to \gamma$

$\gamma \in (N$

leftmost symb

will be te

Step 3 $B_1 B$

Example:-

$A \to BB$

$B \to A$

$C \to AB$

A_1

A_2

~~A_3~~

$A_3 \to$

$A_3 -$

$A_j \rightarrow A_i \gamma$ ↳ $i > j$

$A_K \rightarrow A_j \gamma$ $\quad j < k$.

$A_K \rightarrow A_l \gamma$ $\quad l \geq k$

Apply Lemma 1 [Left Recursion].

new non terminal B added. $B \rightarrow \cdot \alpha_i$

$B \rightarrow \alpha_i B$.

$A \rightarrow A\alpha_1 | A\alpha_2 | \cdots$ $\quad A_j \rightarrow A_i \gamma | \cdots$

↓

Step 2:- $A_i \rightarrow A_j \gamma$ $\quad j > i$

$A_i \rightarrow a\gamma$ $\quad a \in \Sigma$

$B_i \rightarrow \gamma$

$\gamma \in (N \cup \{B_{11}, B_{21}, \cdots B_{r1}\})^*$

$A_m \rightarrow a\gamma$

$A_{m-1} \rightarrow \cdot A_m \gamma$

↓ replace by RHS of A_m -

leftmost symbol will be terminal symbol if there's any -

Step 3 $B_1 B_2 \cdots B_m$

Example :-

$A \rightarrow BB$

$B \rightarrow AC | a$

$C \rightarrow AB$

$A_1 \rightarrow A_2 \cdot A_2$

$A_2 \rightarrow A_1 A_3 | a$

$A_3 \rightarrow A_1 A_2 | A_2 A_1 | a$

$A_1 \rightarrow A_2 A_2$

$A_2 \rightarrow \underbrace{A_2 A_2 A_3}_{\alpha} | a \rightarrow A_2 \rightarrow aB_2$

~~$A_3 \rightarrow A_2 A_2 A_2 \cdot | A_2 A_1 | a$~~

$B_2 \rightarrow A_2 A_3$

$B_2 \rightarrow A_2 A_3 B_2$.

$B \rightarrow \alpha$.

$B \rightarrow \alpha B$

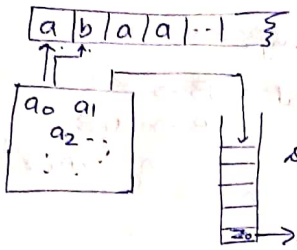
$A_3 \rightarrow A_1 A_2 | A_2 A_1 | G$.

~~$A_3 \rightarrow A_2 A_2 A_2 | A_2 A_1 | a$~~.

$A_3 \rightarrow a B_2 A_2 A_2 | a B_2 A_2 A_2 | a B_2 A_1 | a$

Lavargen ♡ Valintyne Dey Mekanikal Ingeneer Ayefayetean ♥ Masheen Learning Ingeenear Antrapranar Cadarr Dotar Deer Snek Meemarn

(Non Deterministic) Pushdown Automata :- (PDA)



Σ = symbols
Γ = stack symbols-

Notion of acceptence of a language by PDA :- If after the string input ends, if the stack is empty.

Formally :- A PDA is a 7 tuple.

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

- Q → set of states
- Γ → stack symbol set
- z_0 → initial stack symbol, $z_0 \in \Gamma$

$$\delta : Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \longrightarrow \text{Finite subset of } Q \times \Gamma^*$$

$$\delta(q_1, a, z) = \{(p_1, \gamma_1)(p_2, \gamma_2) \ldots (p_k, \gamma_k)\}$$

$\gamma_i \in \Gamma^*$

$\gamma_i = x_1 x_2 \ldots x_m$ → push this first-

$\delta(q, \varepsilon, z)$ → read head does not move-
Pop off p_i and push γ_i

(q, w, γ) ← configuration-

$q \times \Sigma^* \times \Gamma^*$

Instantenious description (ID) of PDA.

The PDA goes from $ID_1 \rightarrow ID_2$ if there is a transition like-

$(q, aw, z\gamma) \vdash (p, w, \alpha\gamma)$

if $\delta(q, a, z)$ contains (p, α)

(n moves) $ID_1 \xrightarrow{n} ID_2$

Accepting configu

$L(M) = \{$

$M = (Q, \Sigma$

$L(M) = \{$

Examples -

1) $\{ a^n b^$

PDA :-

$\{ a^n b^$

Ex

Accepting configuration :-

$$L(M) = \{ x \in \Sigma^* \mid (q_0, x, Z_0) \vdash^* (p, \varepsilon, \alpha) \text{ for some } p \in F \text{ and } \alpha \in \Gamma^* \}$$

↳ entering a final state notion.

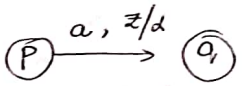
$M = (Q, \Sigma, \Gamma, \delta, Z_0, \phi)$. ↳ (no final state)

$$L(M) = \{ x \in \Sigma^* \mid (q_0, x, Z_0) \vdash^* (p, \varepsilon, \varepsilon) \text{ for some } p \in Q \}$$

↳ empty stack at the end of input.

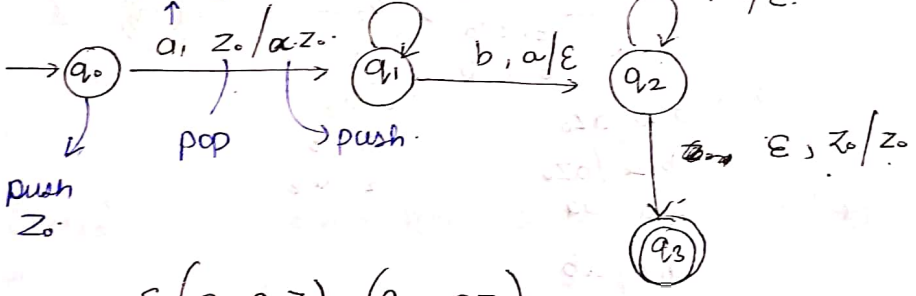
Examples -

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



$\delta(p, a, Z)$ contain (q, α) } Notation --

PDA :-



$\delta(q_0, a, Z_0) = (q_1, aZ_0)$

$\delta(q_1, a, a) = (q_1, aa)$

$\delta(q_1, b, a) = (q_2, \varepsilon)$

$\delta(q_2, b, a) = (q_2, \varepsilon)$

$\delta(q_2, Z_0, \varepsilon) = (q_3, Z_0)$

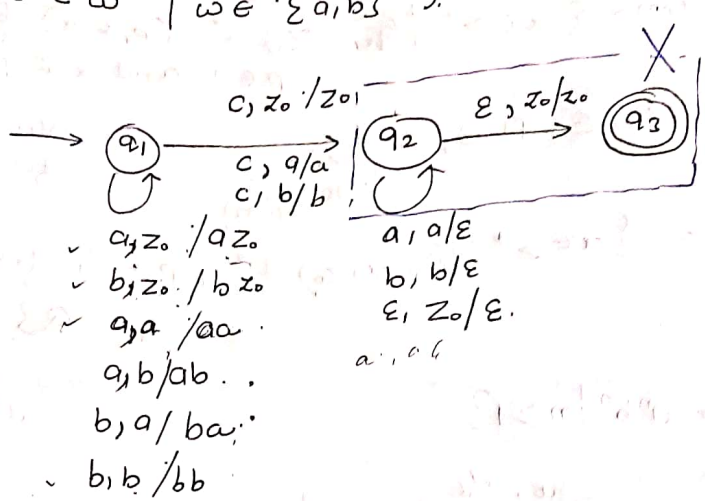
For $\{ a^n b^n \mid n \geq 0 \}$.

simply make q_0 a final state too.

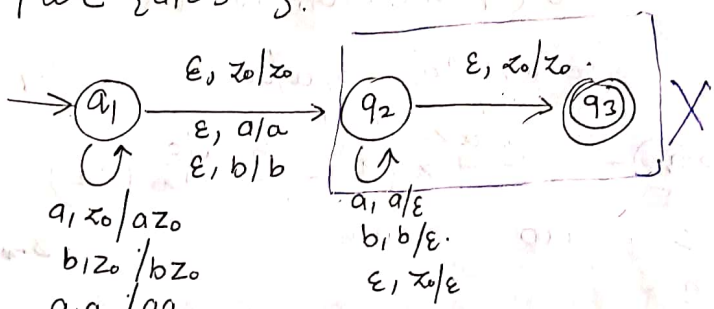
Rest same.

Ex q_0, aabb

Ex $\{\omega c \omega^R \mid \omega \in \{a,b\}^* \}$.

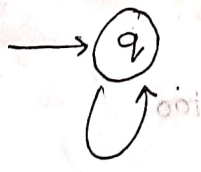


Ex $\{\omega\omega^R \mid \omega \in \{a,b\}^* \}$. PDA ?



Ex $\Sigma = \{(,)\}$ set of all balanced parenthesis.

(( ) ( ))



(, z0 / (z0
(, (/ ((
), (/ ε
ε, z0 / ε

DPDA :- A PDA
z ∈ Γ
is empty
?) δ(q,a,z
q ∈ (Σ ∪ {ε

{ωω^R | ω
{ωcω^R |

ωω^R → ac

class of la
are called
(acce

L(M) →
N(M) →

If L(M) =
for some

Proof (M

(M

1)

2)

3)

DPDA :- A PDA is deterministic if 1) $\forall q \in Q$ and $Z \in \Gamma$ whenever $\delta(q, \varepsilon, Z)$ is non-empty $\delta(q, a, Z)$ is empty for all $a \in \Sigma$

2) $\delta(q, a, Z)$ has only one choice - $\forall q \in Q$ $Z \in \Gamma$ and $a \in (\Sigma \cup \{\varepsilon\})$ (prevents multiple next choices)

$\{ww^R \mid w \in \{a,b\}^*\}$ → non deterministic PDA (condition 1 is violated -

$\{wcw^R \mid w \in \{a,b\}^*\}$ → Deterministic PDA ✓

ww^R → accepted by PDA but not not DPDA -.

(# there must be a way to guess that middle of string is reached)

class of languages accepted by DPDA are called DCFL (Deterministic Context Free Lang).

$L(M)$ → lang. accepted by entering final state -.

$N(M)$ → lang. accepted due to empty stack.

If $L(M) = L$ for some PDA M, then $L = N(M')$ for some PDA M'

Proof ~~Ans~~ $(M = Q, \Sigma, \Gamma, \delta, q_0, Z_0, \Gamma)$

~~Ans~~ $(M' = \{Q \cup \{q_0, q_0'\}, \Sigma, \Gamma \cup \{X_0\}, \delta', q_0', X_0, Q\}$

1) $\delta'(q_0', \varepsilon, X_0) = (q_0, Z_0 X_0)$

↑ initial configuration -.

2) $\delta'(q, a, Z) = \delta(q, a, Z)$ $\forall \cdot q \in Q$ $Z \in \Gamma$ $q \in \Sigma \cup \{\varepsilon\}$

3) ~~Ans~~

$\forall q \in F$, $Z \in \Gamma \cup \{X_0\}$. $\delta'(q', \varepsilon, Z)$ contain (q_e, ε)

$\delta'(q_e, \varepsilon, Z) = (q, \varepsilon)$

$\forall Z \cdot \in \Gamma \cup \{X_0\}$.

if $x \in L(M)$ $\delta(q_0, x, Z_0) \vdash^* (p, \varepsilon, \alpha)$ $p \in F$ $\alpha \in \Gamma^*$

$\delta'(q_0, x, x_0) \vdash_{M'}^{1} (q_0, x, z_0 x_0)$

$\vdash^* (p, \varepsilon, Z'\alpha)$ α is any string.

~~$\vdash$~~

$\vdash^1 (q_e, \varepsilon, \alpha)$

$\vdash^* (q_e, \varepsilon, \varepsilon)$

Then $x \in N(M')$

$p \in F$.

Show that :- If $L = N(M)$ for some PDA M, then there exists $L = L(M')$ for some PDA M'

A DPDA accessed by empty stack has very lil power-

If $L = N(P)$ for some DPDA P, then L has the prefix property.

Prefix property :- No proper prefix of x is in L.

a^* → Does not have prefix property.

↓ ∴ a^* is not accepted by a DPDA accessed by empty stack.

(Although a^* is regular)

Assump.

x, xy ← Both these strings are accepted

$(q_0, x, z_0) \vdash^* (p, \varepsilon, \varepsilon)$

$(q_0, xy, z_0) \vdash (p, y, \varepsilon)$

Conclusion :- They aren't.

∴ Prefix property req.

If L is accepted by some DPDA P that is accepted by empty stack, then there exists a DPDA P' that accepts L by final state. i.e. $L = L(P')$

If L has the prefix property and $L = L(P)$ for some DPDA P then there exists a DPDA P' such that $L = N(P')$

A language $L = N(P)$ for some DPDA P iff L has the prefix property and $L = L(P')$ for some DPDA P'.

Proof :- Given a grammar G now to construct PDA :-

$G = (N, \Sigma, P, S)$ $\quad L = L(G)$

GNF : $A \to aB_1B_2B_3 \cdots B_k \quad k \geq 0$

$M = (\{q\}, \Sigma, \Gamma, \delta, q, S, \phi)$

$\delta(q, a, A)$ contains (q, α) whenever $A \to a\alpha$ is in P.

(stack symbol)

{There are multiple next choices}