Date

17.03.12

Mathematics

Theory Of Computation

TOC → Peter Linz (Text Book)

Kenneth Rosen (Text Book) → Discrete Mathematics

- Keyword - formal → (no ambiguity) the meaning should be precise & not ambigous, it should convey only one meaning.

- Godel, Turing, Post } TOC makers

Formal Logic &

① formalisation of logic.
② limitation of logic.

• Incompleteness Theorem

★ every axiomatic science has a limitation.
★ there are certain things that are true but cant be proved.
★ any problem that has a solution can't be solved via logic (through computers).
★ some problems are beyond mechanical reach.

→ Problems that can be solved by machines or through reasoning :- Algorithm (the problems can be DECIDABLE)

★ If TOC says that the given problem is undecidable, then it means that it can't be solved by any computer.

TOC focuses on
- power (whether we can solve it or not)
- speed

★ There are problems that can't be solved via computers or reasoning mind.

Input → Process (Machine) → Output (Language)

Language :- bunch of grammar rules that govern the correctness of bunch of sentences. These legal bunch of sentences comprise of language.

Chomsky Heirarchy

TOC (Fundamental Model)

1 Languages
2 Grammar
3. Machine

Interconvertible into each other

- Regular
- Recursive ×
- CFL
- CSL ×
- RE
- Not RE (Not Recursively Enumerable)

★ Regular is type of CFL

→ 2^{Σ^*} :- set of all languages. 2^{Σ^*} is a power set that contains all the subsets of Σ^*.

$P(A) = 2^A$

$L \subseteq \Sigma^*$:- collection of all strings. (every possible strings.)

Language

$L_3 = \{ab, ba, a\}$

$L_1 = \{\epsilon, a, aa, aaa, \ldots\}$

$L_2 = \{\epsilon, ab, aabb, aaabbb, \ldots\}$

★ Grammar tells the rule to make the lanaguage.

	Regular	→ Regular Grammar
context free	CFL	→ CFG
context sensitive	CSL	→ CSG
	Recursive	→ no particular grammar
Recursively Enumerable	RE	→ PSG

★ There are some languages that have no grammar.

NOT RE has no grammar.

Machines can't trap those patterns which dont have grammar.

→ problem along with solution represents a language.

e.g. in C, for(i=0, i<100, i++) is correct, but for i = 1 to 100 is incorrect in C, because the grammar defined for C doesn't have this.

★ REC is also called DECIDABLE (it will say yes that it belongs to language for those members that actually belongs to language, & it will say no for those elements that do not belong to the language)

→ Algorithm:-
It must decide which elements/members belongs or doesn't belongs to the Language. It should give the result in finite amount of time.

★ RE → semi-decidable
because it can decide only those members which belongs to the languge,
but if we give a non-member, then it may hangup (i.e. the machine may hangup).

★ NOT RE (not even semi-decidable)
In this language, the grammar isn't there, & it can't decide whether the given element belongs to the languge or not,
i.e. it can neither say yes or no.

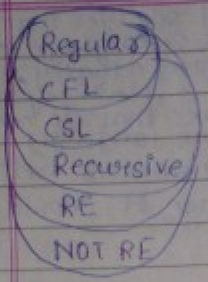
→ Regular :- Finite Automata
→ CFG :- Push down Automata (All power of FA + some extra power)
→ CSG :- Linear Bounded Automata
→ Recursive :- Halting Turing Machine < it will halt & gives the result / it will never hang.
→ RE :- General Turing Machine < either it will halt & gives the result / or it will hang.

★ $L(G_1) = L(G_2)$
This question is undecidable, G_1, G_2 - grammars
{ This problem is decidable when G_1 & G_2 are regular.

→ Not RE :- No machine

★ All the algorithms lies in Regular, CFL, CSL.

Date:
Page No.



★ Every regular language is a CFL, which means that even a^*b^* which is regular is also a CFL.
e.g. a^*b^* which is regular can be fo

LANGUAGE :-

- Alphabet Σ, Γ finite, non-empty set
- String
- Concatenation
- Recursion
- Null string
- Length of string
- Prefix
- Suffix
- Substring
- W^n
- Σ^*, Σ^+
- $L \subseteq \Sigma^*$
- $L_1 \cup L_2, L_1 \cap L_2, \bar{L}, L_1 - L_2, L_1 \oplus L_2$ (set operations)
- $L^n : L_1 . L_2$ (String operations) & L^R
  L^*, L^+

→ Alphabet

$\Sigma = \{a, b\}$ → symbols that can't be broken further
$\Gamma = \{0, 1\}$
$\Sigma = \{a\}$ → unary alphabet
alphabet $\Sigma = \{a, b\}$ → binary alphabet
↳ symbol

→ String

finite sequence of symbols from alphabet.

★ "ϵ" is a string, & not a symbol.

Date:

Page No.

the string has to finish.

→ **Concatenation**

$x = aab$

$y = ba$

$x.y\,(xy) = aabba$

it commutative, but not associative

$x.y \neq y.x$

$x.(y.z) = (x.y).z$

→ ~~Reversal~~

$x = a$

$y = aa$

$xy = aaa$

$w^2 = w.w$

→ **Reversal** :-

$x = aab$

reversal of $x = baa\ (x^R)$

$x = x^R$ [x :- palindrome.] &

if $L = L^R$ (it doesn't mean L is language of palindromes)

$L = L^R$ doesn't mean that that L is a language of palindrome.

e.g. take $L = \{ab, ba\}$

L^R will contain all sets of x^R, ~~ba~~ ab, ba reversal of ab is ba, ba is ab, ∴ $L^R = \{ba, ab\}$

which is equal to L, but as we can see that $L = L^R$ even though L is not a language of palindrome.

→ **Null String**

- $\epsilon.x = x.\epsilon = x$

ϵ → ~~symbol~~ it is representation of null string.

ϵ is identity element of concatenation.

→ **Length of string**

It is the no. of symbols.

~~Length of~~ $|aab| = 3$

$|a| = 1$ $|\epsilon| = 0$

e.g. $|a| = 1$, a is both string & symbol.

- $|x.y| = |x| + |y|$
- $|a.x| = |a| + |x| = 1 + |x|$ [if $a \in \Sigma$]
- $|x^R| = |x|$

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

→ prefix

set of all front part of the string.

prefix(w)

w = xy

x ∈ prefix(w)

y ∈ suffix(w)

prefix(aab) = {ε, a, aa, aab}

suffix(aab) = {ε, b, ab, aab}

Cardinality = (n+1) :- no. of prefixes & suffixes that a string of length n have.

when n - length of word.

→ Substring

any sequence of consecutive symbols in w.

abc - bc

(prefix, substring)

when the string contains all different symbols, the no. of substrings that the string have is $\frac{n(n+1)}{2}+1$

eg. abcd → substrings of length 1 = a, b, c, d

substrings of length 2 = ab, bc, cd

substrings of length 3 = abc, bcd

substring of length 4 = abcd

∴ Total sub no. of substrings = 4+3+2+1 = 10 $\left[\frac{n(n+1)}{2}\right]$ when n=4

Now, the '1' is added to it because (ε) is also a substring of every string.

☆ if no repeatation of symbols, then the no. of strings = $\frac{n(n+1)}{2}+1$

☆ ε is also a substring of each string

Take for example :- aabc, acc. to formula, it should generate 11 substrings but as there is repeatition of symbols, then no. of substrings is less than 11

→ Powers (w^n)

★ If some binary operation * on a set S follows postulates 1 to 4, then it is a group & if it also follows postulate 5, then it is an abelian group.
e.g. × (multiplication) on set of integers is ~~abelian monoid~~.

$w^0 = \epsilon$:- identity element
$w^1 = w$
$w^2 = w.w$
$w^3 = w.w^2 = w^2 w = w.w.w$
w^{-1} :- undefined
n :- non-negative numbers.
$w^{-1}.w = w.w^{-1} = \epsilon$

$(abc)^0 = \epsilon$
$(abc)^1 = abc$
$(abc)^2 = abcabc$

e.g. (∪, ∩, +, −, ×, /, ...)
* :- binary operations
ε :- identity element.
S :- a set of symbols

★ because we can't attach anything to a string to nullify it.

$(\Sigma^*, \cdot) \rightarrow$ Monoid
Σ^+
$\Sigma^+ = \Sigma^* - \epsilon$
Σ^+ :- semi group.

Postulate 1 :- Closure. If a & b are in S then a*b is in S.
Postulate 2 :- Associativity. If a, b, c are in S then (a*b)*c = a*(b*c).
Postulate 3 :- Identity element. There exists a unique element (called the identity element) e is S such that for any element x in S, ~~...~~ x*e = e*x = x.
Postulate 4 :- Inverse. For every element x in S there exists a unique element x' in S such that x*x' = x'*x = e. The element x' is called the inverse of x w.r.t. *.
Postulate 5 :- Commutativity. If a, b ∈ S then a*b = b*a

Semigroup :- Those that conforms to postulate 1 & 2.
Monoid :- Those that conforms to postulates 1, 2, 3.
Group :- Those that conforms to postulates 1, 2, 3, 4
Abelian :- Those that conforms to postulate 5.

→ Σ^*
set of every possible string
$\Sigma = \{a\}$
$\Sigma^* = \{\epsilon, a, aa, aaa, ...\}$ = (regular expression) $L(a^*)$
$= \{a^n\}, n \geq 0$

Informal {
- Set Notation → describe all languages
- Regular Expression → describes only regular languages

Formal {
- Grammar
- Machine

} → ~~almost~~ describe any lanaguage upto RE

if $\Sigma = \{a, b\}$
$\Sigma^* = \{\epsilon, a, aa, ab, ba, bb, ...\}$

Σ^* :- universal language

★ A PDA is a 7-tuple relation containing $Q, \Sigma, \Gamma, \delta, q_0, F, Z_0$ where Γ is a set of symbols ~~are~~ used in stack, Z_0 is the initial symbol in the stack.

~~A PDA is acce~~ A string is accepted in the ~~stack~~ PDA if

$L \subseteq \Sigma^*$

(i) If the stack gets empty on reaching the end of string.
(ii) Or we reach a final state at the end of the stack.

Powers of language :

$L \subseteq \Sigma^*$

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

$L = (0+1)^*$ - all combinations are legal,

but we use

$L \subset (a+b+ \ldots +z)^*$ → when all combinations of symbols are not legal.

$L_1 = \{a, b\}$

$L_2 = \{a^n b c^{n+2}, n \geq 0\}$

★ Finite automata doesn't have external memory.

★ No. of states are always finite in finite automata.

$L_1 = \{\epsilon, a, aa, aaa, \ldots\}$ → can be detected by finite automata, as it doesn't need 'a' to be stored.

$L_2 = \{a^n b^n \mid n \geq 0\}$ → can't be detected by finite automata, as it needs 'a' to be stored in memory to count the no. of a's.

★ Push-down Automata = Finite Automata + Infinite Stack.

but, it has limitations when we have aab:aab as it is stored in stack

| b | ← top
| a |
| a |

now, next 'a' will be compared with bottom of stack which is impossible without popping top.

★ PDA can detect ww^R, but only when $|w|$ = even length & not when $|w|$ = odd length.

LBA (Linear Bounded Automaton):- This contains two tapes, one is known as input tape which is read only & other is working tape that has a R/W head only. The input ~~tape's~~ head can only move towards right, while the working ~~tape's~~ head can move in either direction.

★ LBA (doesn't have full power of Turing Machine)

In General Turing Machine → it has infinite memory. But in LBA → the input length is once fixed, & it can't be changed later, but in General Turing Machine the length of input string is dynamic, which can be changed later.

[लेकिन in case of turing machine इसके साथ एक बार input दे दी input tape में तो भी हम input की length बदल सकते हैं।]

| \$ | a | a | b | a | a | b | \$ | → LBA

↳ the length of input string is fixed

[एक बार input दे दी input tape में तो फिर input की length नहीं बदल सकते (in case of LBA). *]

| a | a | b | a | a | b | → General Turing Machine

• length of input string is dynamic.

★ Set operations on languages:-

$L_1 = \{ab, bc\}$ $\qquad L_1 \cup L_2 = \{ab, bc, aa\}$

$L_2 = \{aa, ab\}$ $\qquad L_1 \cap L_2 = \{ab\}$

$L^c = \Sigma^* - L$

e.g.

$\Sigma = \{a\}$

$L = \{\epsilon, a, aa\}$

$L^c = \{aaa, aaaa, \ldots\}$

& $\Sigma = \{a, b\}$ [Finding complement of a language.]

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

$\Sigma^* = a^* b^* + (a+b)^* ba (a+b)^*$

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

$\therefore L^c = \{(a+b)^* ba (a+b)^*\} \cup \{a^m b^n \mid m \neq n\}$

★ Set difference:- $(L_1 - L_2)$

$L_1 - L_2 = L_1 - (L_1 \cap L_2) = L_1 \cap L_2^c$

$L_2 - L_1 = L_2 - (L_1 \cap L_2) = L_2 \cap L_1^c$ → Theoritical purpose

(→ numerical purpose)

Symmetric diff. :-

Date:
Page No.

★ $L_1 \oplus L_2 = (L_1 - L_2) \cup (L_2 - L_1)$

★ ~~x_1~~ x_2

$\cup, L^c$ or $\cap, L^c$ → Functionally Complete

Concatenation

$L_1 . L_2 = \{xy \mid x \in L_1, \& \ y \in L_2\}$

$L_1 = \{a, ab\}$

$L_2 = \{c, cd, a\}$

$L_1 . L_2 = \{ac, acd, aa, abc, abcd, aba\}$

Note:- $L_1 . L_2 \neq L_2 . L_1$ & $L_1 . (L_2 . L_3) = (L_1 . L_2) . L_3$

$|L_1 . L_2| = |L_2 . L_1|$

★★ $|L_1 . L_2| \neq |L_1| . |L_2|$ but $|L_1 . L_2| \leq |L_1| . |L_2|$

Imp. because there can be some repeatations

e.g. $L_1 = \{a, aa\}$ $|L_1| = 2$

$L_2 = \{ab, b\}$ $|L_2| = 2$, but $|L_1 . L_2| = 3 \neq 4$

$L_1 . L_2 = \{aab, ab, aaab\}$

Reversal

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

$L = \{a, aa, aab, ab\}$

for palindromes $L = L^R$ always.

I. $L = L^R$ if & only if L is a set of palindrome ✗

II. $L = L^R$ if L is a set of palindrome ✓

Imp e.g. $L = \{$set of even no. of a's$\}$

L^R will also contains all elements with even no. of a's.

Date:
Page No.

$L = \{aab, baa\}$
$L^R = \{baa, aab\}$

→ L^n

$L^n = \{w^n \mid w \in L\}$
$L^0 = \{\epsilon\}$ L^0 contains ϵ
$L^1 = L$
$L^2 = L.L$
⋮

e.g. $L = \{ab, ba\}$
$L^2 = L.L = \{abab, abba, baab, baba\}$
$L^3 = L^2.L$

e.g. abbaabbb ~~ab~~ (will it belong to L^4)
now, in L we have 2 bit symbols, so we will break the string in 2 bits.
ab ba ab bb
(check this) but bb ~~can't~~ isn't in L, ∴ the string isn't in L^4
★ & if $L = \{ab, baa\}$ & $L = \{ab, ba, \epsilon\}$

→ L^* (Klean closure of L)
$L = \{ab, ba\}$
$L^* = L^0 \cup L^1 \cup L^2 \cup L^3 \cup \ldots$
$L^0 = \{\epsilon\}$
$L^1 = \{ab, ba\}$
$L^2 = \{abab, abba, baab, baba\}$

w = abbaaa → doesn't belong to L^*

→ L^+ (Positive closure of L)

$\Sigma^* \subseteq L^*$

Grammar :-

$L = \{w \mid n_a(w) = n_b(w)\}$

Grammar is set of rules that is called production

$u \to v$

<S> → <np> <verb>
<np> → <art> <noun>
<art> → a|the
<noun> → boy|dog
<verb> → runs/walks

→ Grammar (G)

L(G) = {a boy runs, a boy walks, ...}

G = (V, T, S, P)
- V → variable
- S → starting variable
- P → set of rules

$S \to N_p V^*$
$N_p \to AN$
$A \to a, A \to b$
$N \to c|d|c$
$V \to +|S$

V = {S, Np, V, A, N}

Production, $u \to v$

Regular Grammar (Type 3 grammar)
CFG (Type 2 grammar)
CSL (Type 1 grammar)

Unrestricted Grammar (Type 0 grammar)

★ Unrestricted Grammar ($u \in (V \cup T)^*$)
→ There should be a variable in left hand side ✓

★ CSL (Type 1 grammar)
$|u| \le |v|$ non-contracting
$aS \to bAB$
$|aS| = 2$
$|bAB| = 3$
$2 \le 3$, ∴ Type 1

- Unrestricted Grammar (Type 0) :- only restriction is that there should be atleast one variable on the left hand side, i.e. $u \in (V \cup T)^*$, it may contain $|u| > |v|$, & v other than T^*V+T^* or VT^*+T^*.
- CSG (Type 1) :- to be in type 1, the grammar should be in type 0 & $|u| \le |v|$, i.e. it still may contain $u \in (V \cup T)^*$.
- CFG (Type 2) :- To be in type 2, the grammar should be in type 1 & $u \in V$, which mean that there should only be a single variable on left hand side

$aA \rightarrow aB|C$:- Type 0

| Java, C → CSL |

Type 2 :-

$u \in V$

only single variable is allowed in left hand side (no Terminal allowed)

Type 3:-

$v \in T^*V$
$\in T^*$
$\in VT^* + T^*$

$S \rightarrow abA | aabaC | B | aab | aA$

→ To be in Type 3, it should be in Type 2, & to be in Type 2 it should be in Type 1, & to be in Type 1 it should in Type 0.

☆☆ If we have :-
$A \rightarrow \epsilon$ (contraction is allowed).

L(G) =

$S \rightarrow aSb$ $S \rightarrow aSb \rightarrow aab$
$S \rightarrow aA$
$S \rightarrow a$

- $S \rightarrow aSb|\epsilon$
  $L(G) = \{a^n b^n | n \geq 0\}$
- $S \rightarrow aS|\epsilon$
  $L(G) = \{a^n | n \geq 0\}$
- $S \rightarrow aA$
  $A \rightarrow aAb|\epsilon$
  $A \rightarrow a^n b^n$
  $S \rightarrow aA$ $\{a^{n+1} b^n, n \geq 0\}$
- $S \rightarrow aA|aB$
  $A \rightarrow bS|CD$
  $S \rightarrow aB$
  $B \rightarrow S$

Two methods:-
1. Brute Force Method
2. Standard ~~method~~ grammar substitution
3. [G → M → L] → for regular grammar only

☆ every derivation starts with S & ends with w.

★ The set of all terminal strings obtained by production rules is the language defined by the grammar.

Date:
Page No.

$S \to AB$
$A \to aA \mid \epsilon$
$B \to Bb \mid \epsilon$

$S \to AB \to aAB \to$ ~~aABbB~~ $\to aS$
$\hookrightarrow aA \to a$ [using $B \to \epsilon$].

★ $S \to AB \to aAB \to aA \to a$

Sentence

Sentential form

→ every sentence is a sentential form.

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

$S \to AB \to aAB \to aA \to a$ or it can also be written as $S \overset{*}{\to} a$

★ for a given language there can be many grammars,
but for a given grammar, there can be only one language.

★ ★ $G_1 \equiv G_2$ iff $L(G_1) = L(G_2)$
★ ★ $r_1 \equiv r_2$ iff $L(r_1) = L(r_2)$
★ ★ $M_1 \equiv M_2$ iff $L(M_1) = L(M_2)$

$(a+b)^* \to S \to aS \mid bS \mid \epsilon$
$S \to Sa \mid Sb \mid \epsilon$
$S \to a \mid b \mid SS \mid \epsilon$

?

$T^* + T^*V$
or
$T^* + VT^*$
→ Type III

$S \Rightarrow w_1 \Rightarrow w_2 \Rightarrow w_3 \cdots \Rightarrow w$

★★ sentential forms:-
here, S, w_1, w_2 which contains both variables & terminals are called sentential forms.
sentence:- w contains just the terminals & hence is in the language defined by grammar & is called the sentence.

★★ To check:- $L(G_1) = L(G_2)$ → Undecidable (only when G_1 & G_2 are not regular grammars)
if G_1, G_2 are both regular, then it is decidable.

Date:
Page No.

Machines

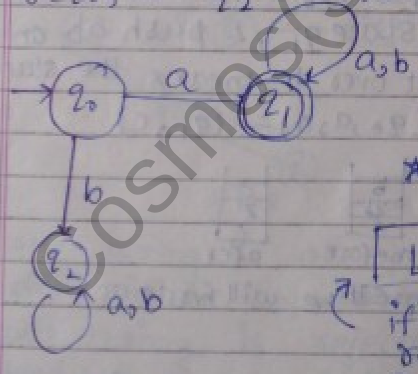
Set of commands by which we can decide which is a member of machine & which is not the member of machine.

1. Finite Automata
2. PDA
3. ~~LBA~~ TM — LBA, HTM, STM
4. ~~HTM~~

★ each cell can hold can only one symbol

q_0 is the starting state.
"δ command" is software for the machine.
by changing δ, we can change to make the machine to do diff. things.
~~everymo~~

$\delta(q_0, a) = q_1$ $\delta(q_1, a) = q_1$
$\delta(q_0, b) = q_2$ $\delta(q_1, b) = q_1$



◎ → final state

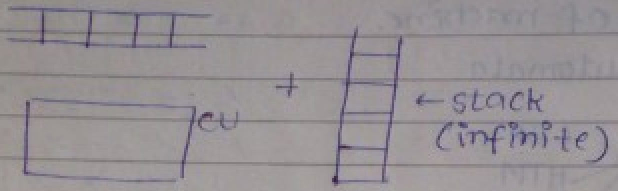
★ we can make n final states, where $n \geq 0$.

$L(M) = a(a+b)^*$

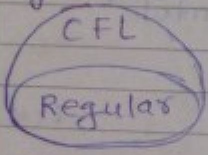
if we start with a, we will reach final state q_1 then it doesn't matter whether a or b will come, it will remain in q_1.

Date:
Page No.

Push Down Automata :-



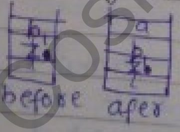
e.g. it can accept $\{a^n b^n | n \geq 0\}$



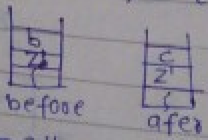
- push
- pop
- do nothing
- replace

↳ $\delta(q_0, a, b) = (q_1, ab)$
if we are in q_0, & we get a on the string & b at top of stack, then we go to q_1

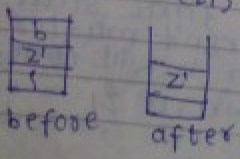
$\delta(q_0, a, b) = (q_1, ab)$ → means that if presently the initial state is q_0 & input is b & if the top of the stack have b then go to state q_1, & push ab on the stack by overwritting it over b, means the stack will be:



if $\delta(q_0, a, b) = (q_1, c)$
then



If we want to pop ~~some th~~ we will write as:-
$\delta(q_0, a, b) = (q_1, \epsilon)$

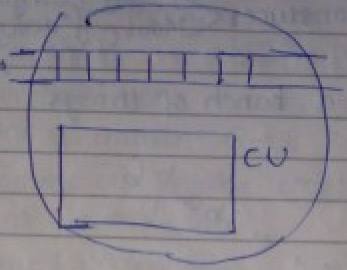


★ If some problem can't be solved by Turing Machine then the problem can never be solved.

?

Turing Machines

Infinite memory



★ In Turing Machine, we can move the head both Left & Right, whereas in FA & PDA, we can move the head only to the right.

★ Turing Machine has made to analyse the power of computer & not the speed.

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

- Deterministic & non-deterministic machines

→ Deterministic

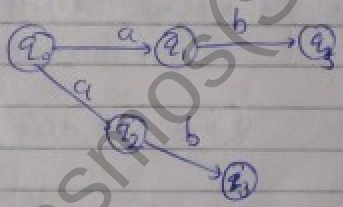
$\delta(q_0, a) = q_1$

there is no choice

→ Non-Deterministic

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

the machine has the choice to go to either q_1 or q_2.



dfa :- deterministic finite state acceptor

nfa :- non-deterministic finite state acceptor.

The string is accepted if any path in non-deterministic approach, we reach a final state, but string is rejected when all paths lead to states which are not final states.

→ δ (called as state transition function)

→ Δ (called as output function)

$\Delta(q_0, a) =$ '001', means if we are at q_0 & we get a, then 001 will be the O/P.

Date:
Page No.

- Finite Automata
  - Acceptor
    - dfa
    - nfa
  - Transducer
    - Mealy } → digital design
    - Moore }

→ Transducer can do a bunch of things

np

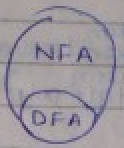
$S \to aSbc \mid \epsilon$

a

Date: 18.03.12

REGULAR LANGS & FINITE AUTOMATA

1. DFA & NFA-Theory
2. DFA design L→M, M→L (Impt.)
3. Regular Expressions : r→L, L→r, $r_1 = r_2$, identities in reg expression, Ardin's theorem, M→r
4. Regular Grammar : G→L, $G_1 = G_2$ 1. Standard grammar + substitution 2. G→M→L
5. Algorithms in Regular langs. NFA→DFA
6. Regular & Not-Regular
7. Closure & Decidability (V.Imp.)
8. Applications, Limitations & Variations
9. FSM



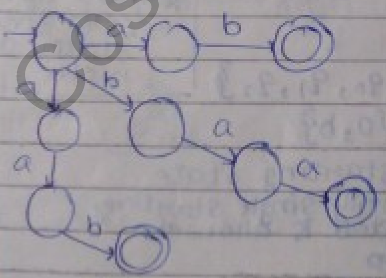
- Every DFA is a NFA
- every NFA can be converted into equivalent DFA.

But for pdpa & npda:-
- every pdpa is a npda.
- but every npda can't be converted into equivalent dpda. (undecidable).

★ Every finite language is regular.
The language is regular iff we can design an finite automata for it.

Finite → Regular (Reverse is not true)
Finite Automata → Language is regular

L = {ab, baa, aab}



ab, baa, aab
a(b+ab) + baa

★ Every finite language is regular, but every regular language is not finite.

Imp ★ We dont require memory for a regular expression.
i.e. for a regular expression, we dont require memory for it.

- ε-nfa (null move in nfa)
- nfa → dfa
- dfa → min. dfa
- mealy → moore
- moore → mealy

(mealy → moore and moore → mealy: these contains both δ & Δ)

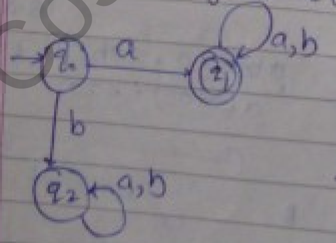
$\Delta(q_0, a) = \text{"001"}$ → Mealy [O/P depends upon the present state & present I/P]
$\Delta(q_0) = \text{"001"}$ → Moore [O/P depends upon the present state only]

★ We can convert Mealy into Moore & vice versa.

DFA & NFA Theory

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

Q:- finite non-empty set of internal states
Σ:- input alphabets (set)
δ:-
q_0:- starting state (there can only be one starting state)



$Q = \{q_0, q_1, q_2\}$
$\Sigma = \{a, b\}$
q_0:- starting state (only single starting state is allowed)
$F = \{q_1\}$

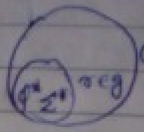
★ There can be zero final states as well.

F:- final state (there can be zero or more final states.)

→ if there is zero final state, then F is ϕ.

Date:
Page No.

Chomsky hierarchy:-



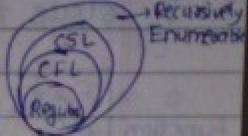
Imp

★ $L_{reg} = \{\phi, \Sigma^*, a^*, a^*b^*, \ldots\}$

★ $L_{CFL} = \{\phi, \Sigma^*, a^*, a^*b^*, \ldots, a^nb^n, \ldots\}$

Chomsky heirarchy consists of following:

(1) Regular
(2) CFL
(3) CSL
(4) Recursively Enumerable



dFA:- $\delta_{Q \times \Sigma} \rightarrow Q$

e.g. $\delta(q_0, a) = q_1$

in DFA, if we have n symbols, then every state should have n no. of arrows going out from that state.

nFA:- $\delta_{Q \times (\Sigma \cup \{\epsilon\})} \rightarrow 2^Q$

this means that null moves are allowed in nfa.

e.g. $\delta(q_0, \epsilon) = q_2$

for our example on previous page:-

$2^Q = \{\phi, \{q_0\}, \{q_1\}, \{q_2\}, \{q_0, q_1\}, \{q_1, q_2\}, \{q_0, q_2\}, \{q_0, q_1, q_2\}\}$

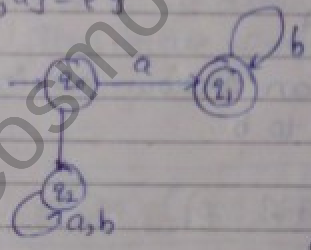
this means that we can go to any no. of states from present state in NFA, e.g. we can go to q_1, q_2 from q_0

DFA	NFA
① No choice	① Choice
② Dead configuration is not allowed.	② Dead configuration is allowed.
③ Null move is not allowed	③ Null move is allowed.

→ Dead configuration

$\delta\{q_1, a\} = \{\}$

e.g.



this is nfa, as there is no guideline to go to any state when we are at q_1 & we get an 'a'

aa will be rejected because we dont know where to go from q_1 when we get an 'a'.

Date:
Page No.

1. function method
2. state diagram
3. state transition table

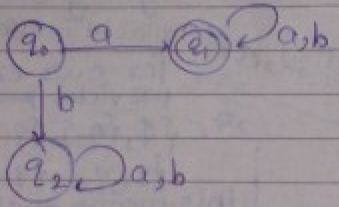
Function Method

$F = \{q_1\}$

$\delta(q_0, a) = q_1 \quad \delta(q_1, a) = q_1 \quad \delta(q_2, a) = q_2$

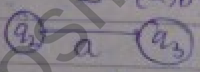
$\delta(q_0, b) = q_2 \quad \delta(q_1, b) = q_1 \quad \delta(q_2, b) = q_2$

State diagram

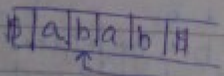


1. First check if there is a null move.
i.e. check like this

2. Secondly check if there is all arrows coming out of a state corresponding to each symbol. (i.e. check if there is some dead configuration or not.)
e.g.
$\Sigma = \{a, b\}$



as we dont have an arrow coming out of q_2 corresponding to b.



from the state diagram drawn above

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

↑ when we have moved from a to b in the tape when we receive a at state q_0.

★ The only way to accept a string in dfa, nfa, is to be at final state at the end of string. (in case of pda, the machine will accept (i) if we are at final state at the end of string (ii) if the stack gets empty at the end of string)

★ NFA rejects when:-

→ when it encounters a dead configuration.

→ when it doesn't encounter a final state at the end of string.

★ In DPDA:-

- we allow dead configuration. (as $Q \times \Sigma \times \Gamma$ is very big set.)

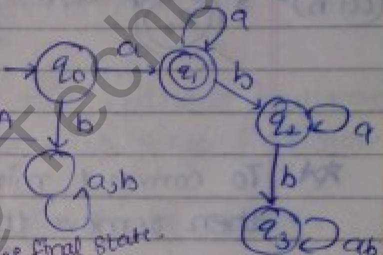
★ the only diff. b/w DPDA & NPDA is the choice.

δ^*: Extended State Transition

$\delta^*_{Q \times \Sigma^*} \to Q$

$\delta^*(q_0, aabb) = q_3$

DFA

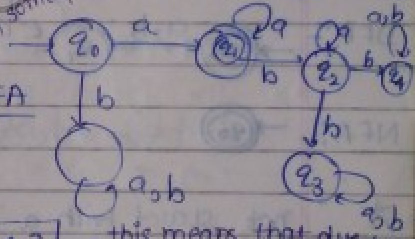


in case of nFA

$\delta^*(q_0, aabb) = \{q_3, q_4\}$ → we will be in some final state.

this means that there will be no choice, & definitely we

NFA



DFA $\boxed{L(M) = \{w \in \Sigma^* \mid \delta^*(q_0, w) \in F\}}$

NFA $\boxed{L(M) = \{w \in \Sigma^* \mid \delta^*(q_0, w) \cap F \neq \phi\}}$ → this means that due to choices, we might be in one final state or the other on reception of same string

i.e. the string on arriving at its end must reach some final state from the sets of final state

Theorem:-

$\boxed{\overline{L(M)} = L(\overline{M}) \text{ (for DFA only)}}$ → not for nfa, because we are interchanging final & non-final states, but we are doing nothing for dead configuration.

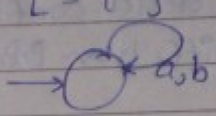
Imp. ★ $\overline{L(M)}$ is obtained by exchanging final states to non-final states & vice-versa (including starting state)

★ Only way to accept 'ε' is to make the initial state as final state.
Imp

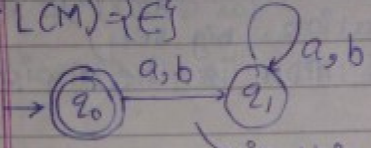
$\overline{L(M)} = \{ w \in \Sigma^* \mid \delta^*(q_0, w) \notin F \} \rightarrow$ DFA

$\overline{L(M)} = \{ w \in \Sigma^* \mid \delta^*(q_0, w) \cap F = \phi \} \rightarrow$ NFA

$L = \{\phi\}$
or
$L = \{\ \}$ $F = \{\ \}$

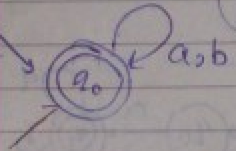


$L(M) = \{\epsilon\}$



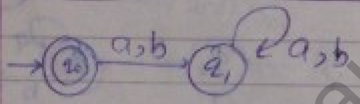
→ in this case, it will accept only ε, if a or b comes, it will reach q_1 & the string will be rejected

for $(a+b)^*$



★★ To convert min. dfa into min. nfa, then remove the trap state.

DFA:-



NFA:-

★ " not start with a or not end with b "
↓ equivalent to
start with a & end with b

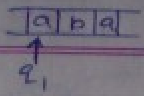
(works only for DFA)

KLEANE'S Theorem :-

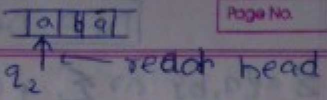
A language L is regular iff ∃ an nfa M which accepts it.

this means we can build an nfa which accepts the languge L.

★ <u>Null move</u> :- This means don't move the read head forward.



$\delta(q_1, \epsilon) = q_2$



A language L is regular iff ∃ an dfa M which accepts it.

A language L is regular iff ∃ a regular expression r such that L(r) = L.

A language L is regular iff ∃ a regular grammar G such that L(G) = L.

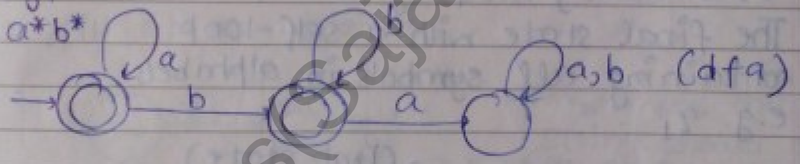
∃ → atleast one	∃! → exactly one

→ $a^* \equiv a + a^*$ & not $a^* = a + a^*$

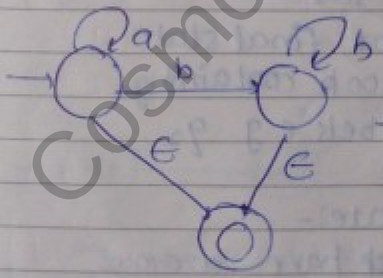
A language L is regular iff ∃ an nfa without null-move
iff ∃ an nfa with single final state

• but in dfa we can't have a single final state which is regular.

e.g.
a^*b^*



but in nfa



→ regular

A language is regular iff ∃ a transition system which accepts it.

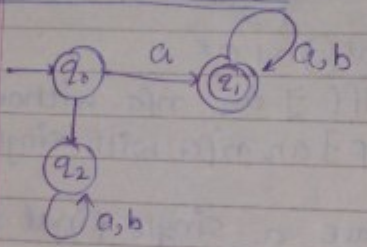
☆☆ in minimal DFA, there is only one trap state.

DFA Design

$\Sigma = \{a, b\}$ or $\Sigma = \{0, 1\}$

1. start with "a".
2. end with "b".
3. start with "a" & end with "b".
4. start with "a" & end with "a".
5. not start with "a" or not end with "b".

① Start with "a".



Permenantly Accepting state
" Rejecting state
Temporary Accepting "
" Rejecting "

Permenantly Accepting state :-

- The final state with a self-loop containing all symbols in alphabets. e.g. q_1.

Permenantly Rejecting state :- (Trap state)

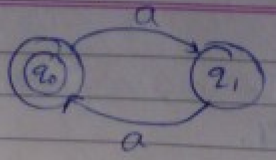
- The state which is not final state & that contains self-loop containing all symbols in alphabets. e.g. q_2.

Temporary Rejecting state:-

- The state which have arrows going out from it & they are not self-loops & rejects the null move is temporary rejecting state. e.g. q_0.

Imp ★★ In DFA, if we have n symbols, then we must have (n+1) minimal states atleast.

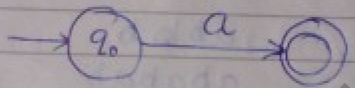
even no. of a's



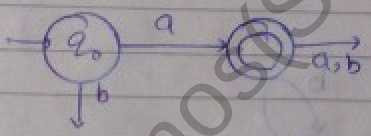
- Temporary Accepting State:-
In above, the q_0 is accepting only when even no. of a's arrive, ∴ it is temporary accepting state.

(1) Start with "a"

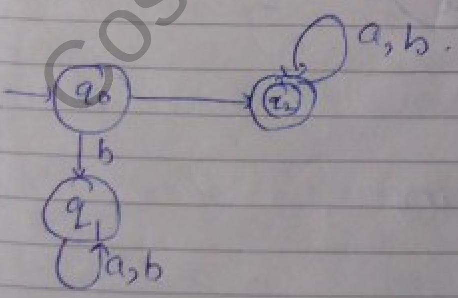
- We have atleast 1 symbol "a", ∴ we must have atleast 2 states.



- Now, fill up the blanks.



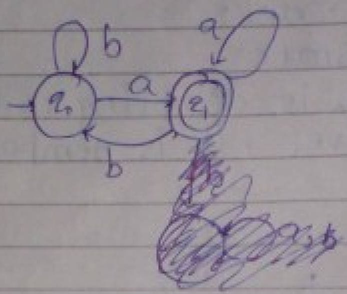
Now, we see that the string should start with a, so when we have a 'b' at q_0, therefore, we should have a permanent trap there



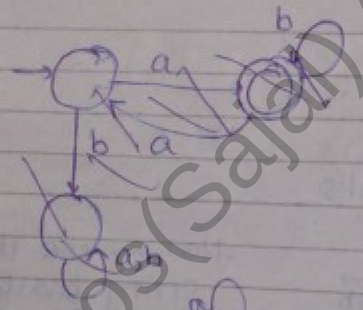
Now, we want the string to start with "a" & nothing else, there we put a permanent accept at q_2.

2 End with "a"

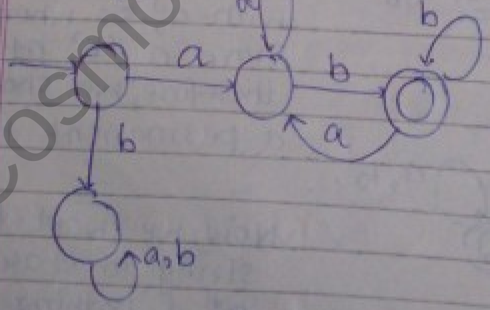
- Atleast two states.



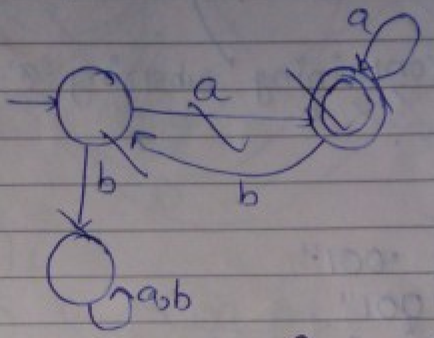
3 start with "a" & end with "b"

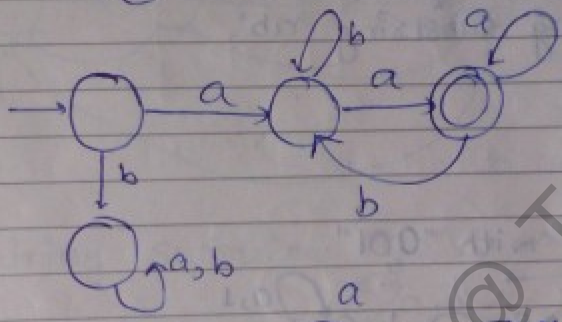


abbb
ababab



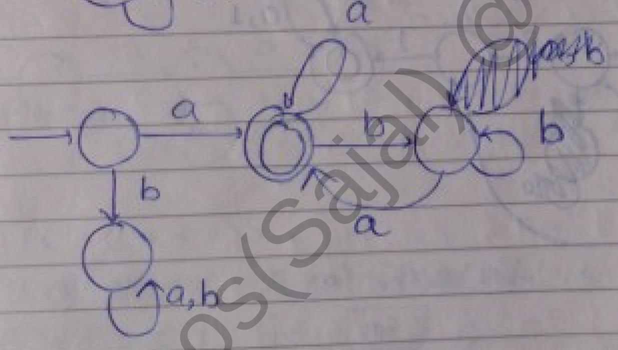
4. start with "a" & end with "a"





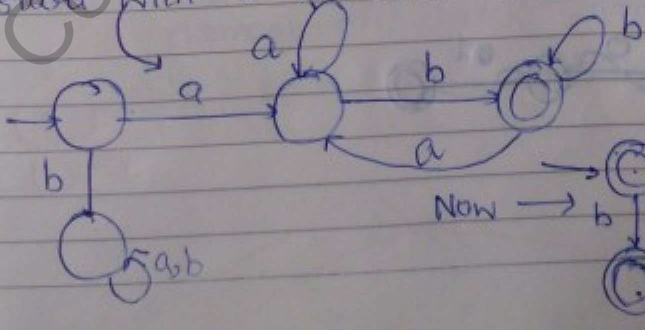
wrong

or because 'a' will not be accepted.

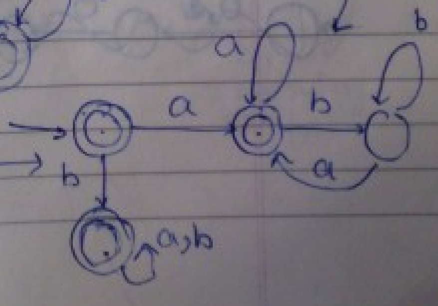


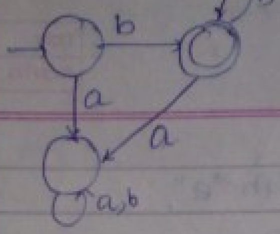
5. not start with "a" nor end with "b"

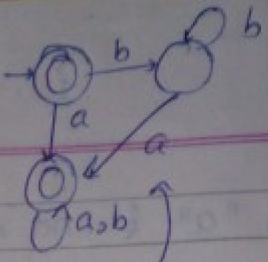
Take complement → start with "a" & end with "b"



Now →



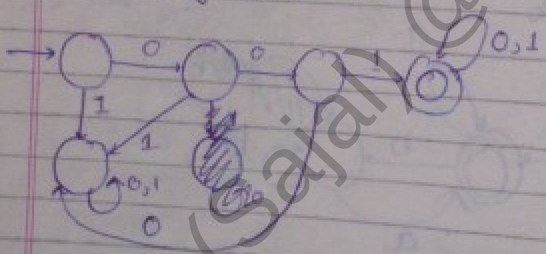




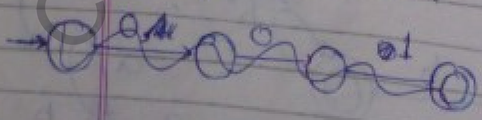
containing substring 'a'.

6. Starting with "001".
7. Ending with "001".
8. Containing substring "ab".
9.

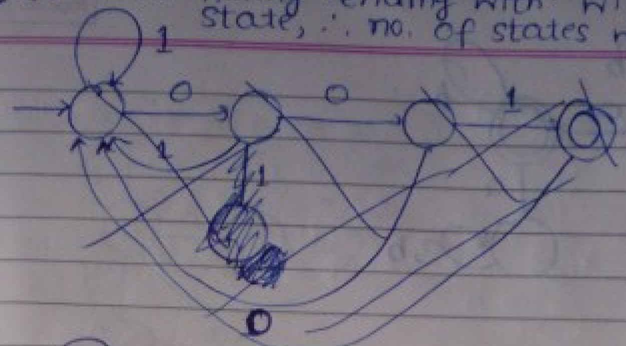
6. Starting with "001"

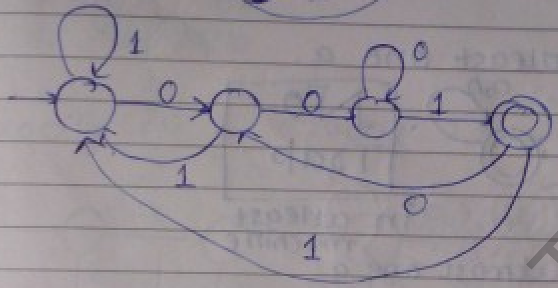


7. Ending with "001".



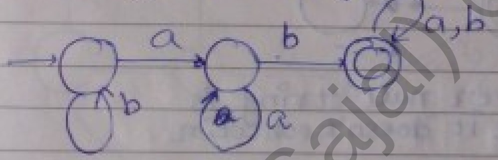
☆☆ Trap state will be there when we have question saying 'start with' will have $(n+2)$ states.
☆☆ Those having "ending with" will have no trap state, ∴ no. of states will be $(n+1)$.





8. Containing substring "ab"

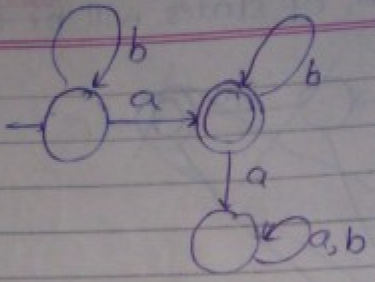
★ In substring ones, if there are n



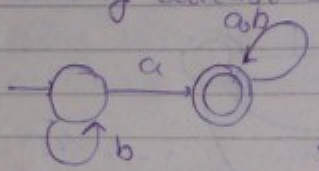
★ no. of states in minimal fa :-
$\{a^{nk} \mid k \geq 0\}$ is $\boxed{n+1}$

	DFA	NFA
9. Containing exactly one 'a'.	$(n+2)$	$n+1$
10. " atleast one 'a'.	$(n+1)$	$n+1$
11. " atmost one 'a'.	$(n+2)$	$n+1$

(NFA: by removing traps)



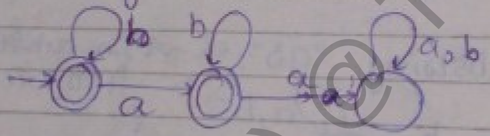
(10) containing atleast one a.



No Trap

in atleast machine

(11) containing atmost one a

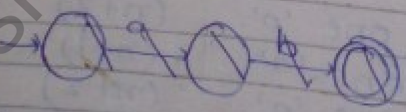


In this case even null string is accepted because it doesn't contain any a's.

(12) Exactly 1 "a" & exactly 1 "b" $(m+1)(n+1)+1$

(13) Atleast 1 "a" & atleast 1 "b" $(m+1)(n+1)$

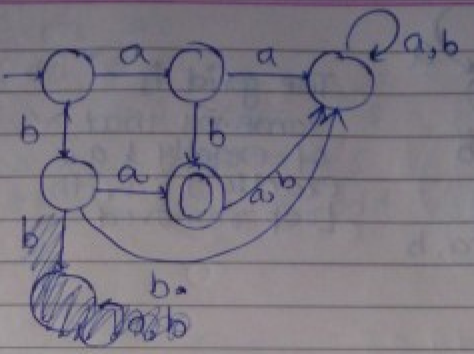
(14) Atmost 1 "a" & atmost 1 "b".



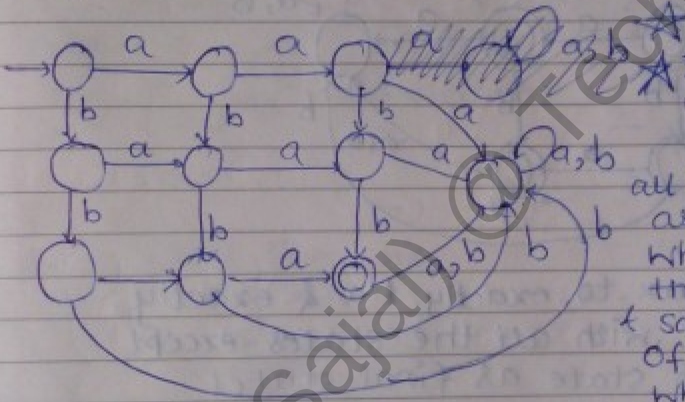
(12) Exactly 1 "a" & exactly 1 "b".

surreshot V.V. Imp. → Grid Machines

Date:
Page No.



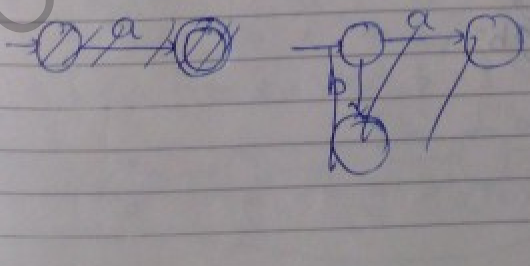
Exactly 2a & exactly 2b



★ In case of "or" we just make all the states as final states which are on the same x-axis & same y-axis of that state which is final in "&" case.

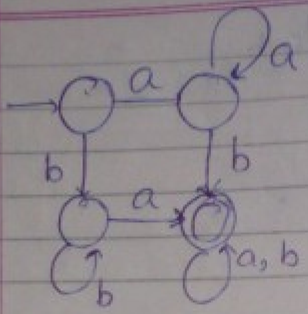
★ for exactly m a's & exactly n b's, we will have $(m+1)(n+1) + 1$

13. atleast 1 "a" & atleast 1 "b"



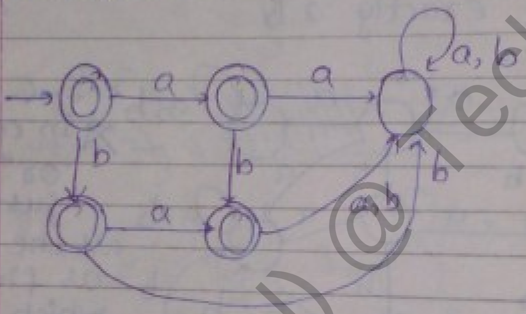
Date:
Page No.

★ If there is "exactly", then we need a trap.



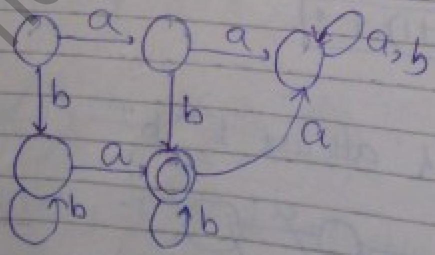
The grid is same as that of exactly 1 a & exactly 1 b with trap removed.

14. Atmost 1 "a"



similar to exactly 1 a & exactly 1 b, with all the states except trap state as final states

15. Exactly 1 a & atleast 1 b



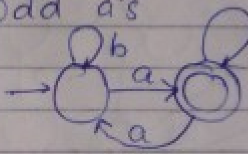
2755122
2755122

16. Even a's
17. Odd a's
18. Even a's & even b's
19. Even a's & odd b's
20. Even a's or odd b's

(16) Even a's

L = {w | $n_a(w)$ mod 2 = 0}
2 states are required

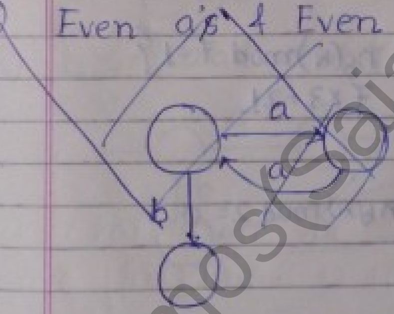
(17) Odd a's



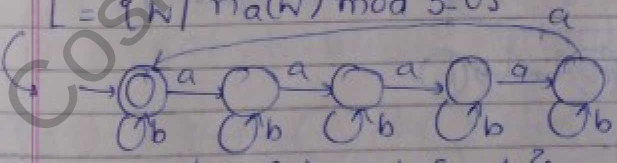
L = {w | $n_a(w)$ mod 7 = 0}
7 states are required

(complement of even a's)

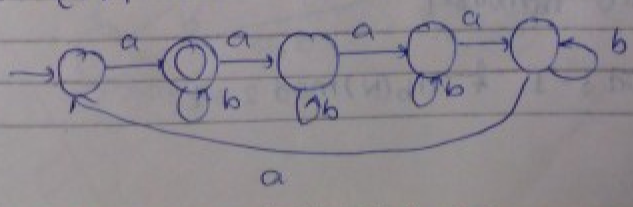
(18) Even a's & Even b's



L = {w | $n_a(w)$ mod 5 = 0}

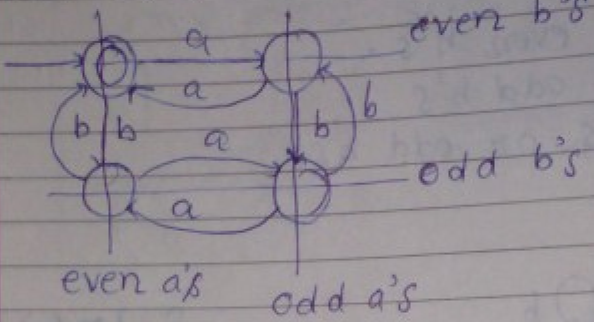


L = {w | $n_a(w)$ mod 5 = 1}

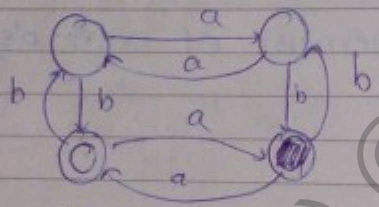


Date:
Page No.

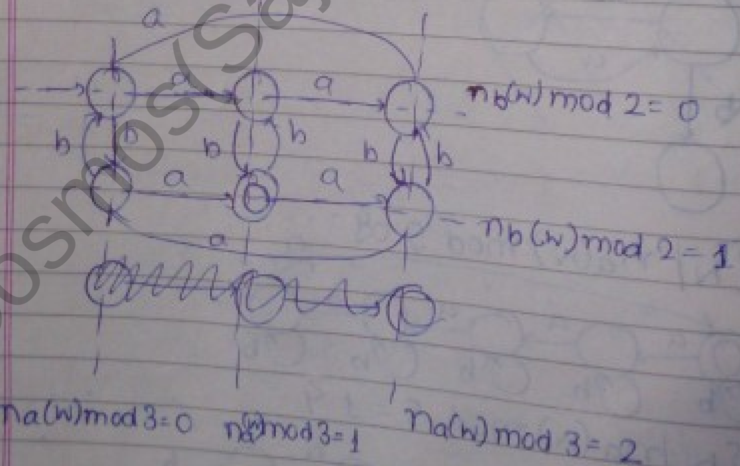
18. Even a's & even b's



19. Even a's & odd b's



★★ $L = \{w \mid n_a(w) \bmod 3 = 2 \ \& \ n_b(w) \bmod 7 = 1\}$
↳ So total states will be $7 \times 3 = 21$



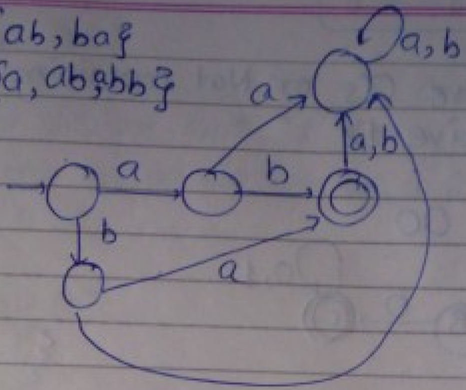
$n_a(w) \bmod 3 = 1 \ \& \ n_b(w) \bmod 2 = 1$.

★ In finite language, we need atleast (n+2) states

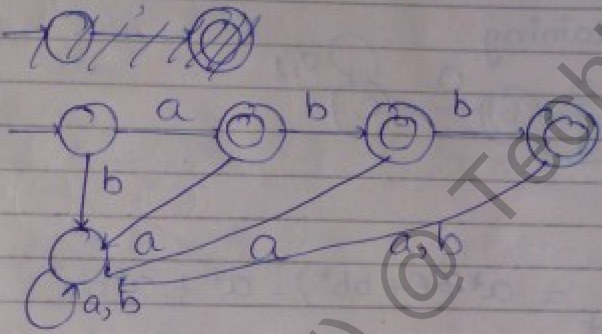
21. L = {ab, ba}

22. L = {a, ab, bb}

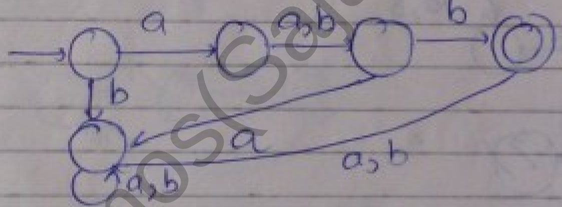
(21) →



(22)



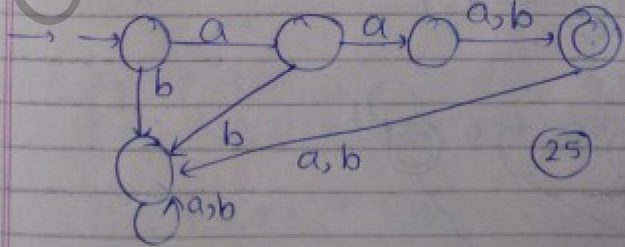
(23) L = {aab, abb}



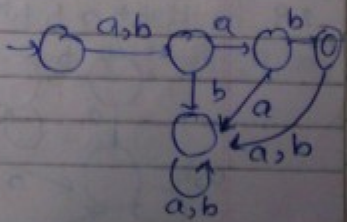
(24) L = {aab, aaa}   aa(b+a)

(25) L = {aab, bab}   (a+b)ab

(24) →



(25)



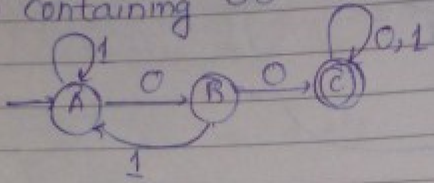
Containing Machines:—
No trap state

$\epsilon + rr^* = r^*$

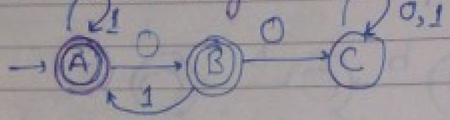
(26) No consecutive 0's or Not containing "00"

(27) No consecutive 1's

(26) → containing "00"



not containing

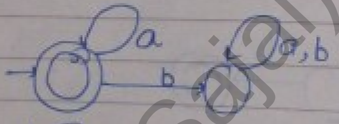


(27) a^*

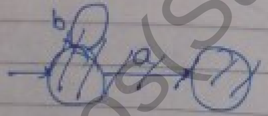
(28) $a^*b^* = a^*(\epsilon + bb^*) = a^* + a^*bb^*$

(29) bb^*abb^*

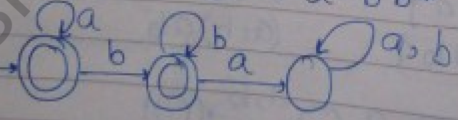
27



28

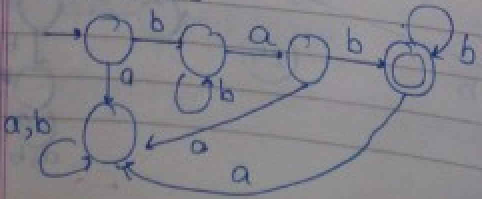


$a^*bb^* = a^* + a^*bb^*$



29

bb^*abb^*



Date:
Page No.

$b^*b = bb^*$

☆☆ So if we are asked to design b^*b, we should design bb^*, if we go on to design b^*b, we would design an nfa.

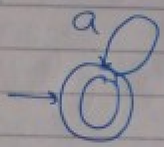
$\Sigma = \{a\}$

30. a^*

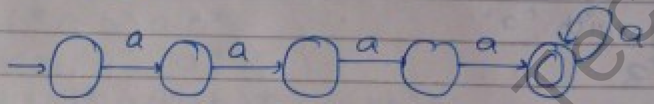
31. $aaaaa^*$

32. $(aaa)^*$

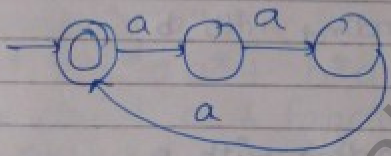
30.



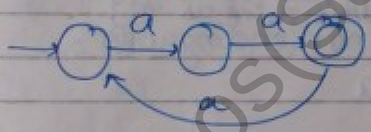
31.



32.



33. $(aaa)^*aa$



34. $(aaa)^*aa = a(aaa)^*a = aa(aaa)^*$

35. $(aa+aaa)^*$

ε	a	aa	aaa	aaaa	aaaaa	aaaaaa
	✗	✓	✓	✓	✓	✓
		2a				

So, after 2a all strings are accepted.

∴ the total no. of states are [3] = {ε, a, aa}

$(a+aaa)^*$, so the least no. of states req. is [1].

aga

Date:

Page No.

Regular Expression

1. $L \to r$
2. $r \to L$
3. $M \to r$
4. identities $r_1 \equiv r_2$
5. Arden's theorem

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

① → Starting with 'a' :-
$a.(a+b)^*$

② → ending with 'a'
$(a+b)^*a$

③ → starting with 'a' & ending with b
$a(a+b)^*b$

④ → Starting with 'a' & ending with a
$a(a+b)^*a + a$ [as $a(a+b)^*a$ will have smallest string as 'aa', but we can have smallest string as 'a']

⑤ → not starting with 'a' or not ending with b
$\epsilon + b(a+b)^* + (a+b)^*a$

⑥ → starting with 'aa' & ending with 'aa'.
$aa(a+b)^*aa + aa + aaa$

⑦ starting with "001"
$001(0+1)^*$

⑧ ending with "001".
$(0+1)^*001$.

Date:
Page No.

9. containing substring "ab"

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

10. ~~At~~ Exactly 1 "a"

$b^* a b^*$

11. Atleast 1 "a"

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

12. Atmost 1 "a"

$b^* + b^* a b^*$

13. Exactly 2 "a"

$b^* a b^* a b^*$

14. Atleast 2 "a"

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

15. Atmost 2 "a"

$b^* + b^* a b^* + b^* a b^* a b^*$

16. Exactly 1 a & Exactly 1 b.

$ab + ba$.

17. Even a's

~~$(aa)^*$~~

$\epsilon + (b^* a b^* a b^*)^* + bb^*$ — इसमें b^* इसलिए नहीं किया क्योंकि इससे ϵ भी बनता, but ϵ $(b^* ab^* ab^*)^*$ में count हो गया था।

18. Odd a's

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

or

$b^* a (b^* a b^* a b^*)^* + b^*$ → इसमें bab नहीं आएगा, तो इसको और $b^* a b^*$ को include करने के लिए हमने $+ b^*$ किया है।

19. No 2 consecutive 0's.

~~$(0+1)^*$~~ $(01+1)^* . (\epsilon+0)$ → this will make sure that string will also ends with 0.

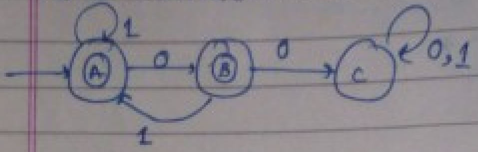
$(01+1)^*$ → this will cause the string to end with a '1'.

≡ $(\epsilon+0)(10+1)^*$

20. No 2 consecutive 1's

$(0+10)^*(\epsilon+1)$ — we are putting ϵ as without it, the string will end with 1.

No 2 consecutive 0's



regular expression for A :- $(1+01)^*$

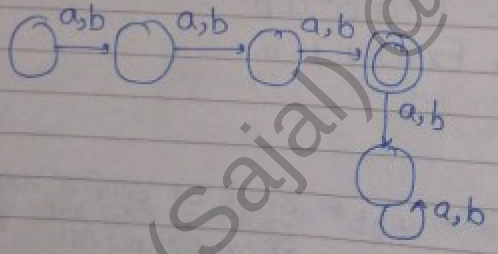
" " " B :- $(1+01)^* 0$

A followed by 0,
means we can reach B by reaching A & then getting a '0' as the input.

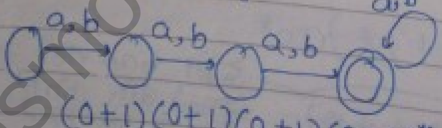
∴ regular expression = $(1+01)^* + (1+01)^* 0$

Length Machines

→ $L = \{w \mid |w| = 3\}$ $(n+2)$ $(a+b)(a+b)(a+b)$ or $(0+1)^3$

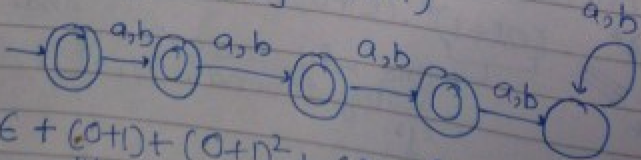


→ $L = \{w \mid |w| \geq 3\}$ $(n+1)$



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

→ $L = \{w \mid |w| \leq 3\}$ $(n+1)$



$\epsilon + (0+1) + (0+1)^2 + (0+1)^3$

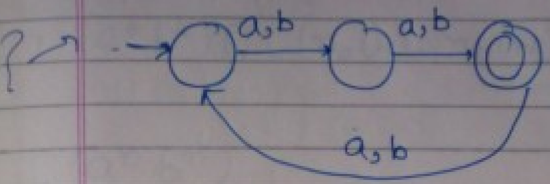
|||

$(\epsilon + 0 + 1)^3$

To shorten the atmost regular expression, we write

$(\epsilon+0+1)^n$

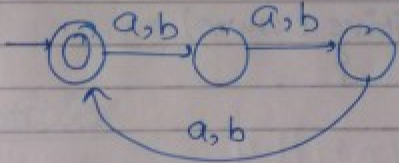
→ $L = \{w \mid |w| \bmod 3 = 2\}$



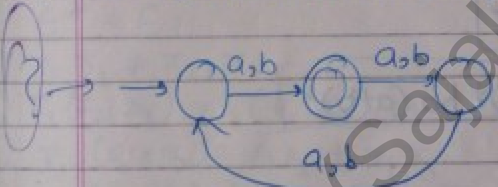
(0+1)

→ $L = \{w \mid |w| \bmod 3 = 0\}$

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



→ $L = \{w \mid |w| \bmod 3 = 1\}$



Regular Expression to Language

→ A regular expression can contain :-

$\phi, \epsilon, x \in \Sigma$

$+, \cdot,$

$r_1 + r_2$

$r_1 \cdot r_2$

$L(r_1 + r_2) = L(r_1) \cup L(r_2)$

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

$L(r^*) = (L(r))^*$

$() > * > \cdot > +$ → precedence

$\boxed{aa^* = a^+}$ $\boxed{a^*a = aa^*}$

$aaa^+ = aaaa^*$

→ $r = \phi$

$L = \{ \}$

→ $r = \epsilon$

$L = \{\epsilon\}$

→ $r = a$

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

→ $r = a + b$

$L(r) = \{a, b\}$

→ $r = a^*$

$L(r) = \{\epsilon, a, aa, aaa, \dots\}$

→ $r = aa^* = a^+$ (positive closure)

$L(r) = \{a, aa, aaa, \dots\}$

→ $r = a^* + b^*$

$L(r) =$

$\boxed{a^*b^* = a^* + b^* + a^*b^*}$

$\boxed{(a+b)^* = a^*b^* + (a+b)^*ba(a+b)^*}$

→ $a^* = aa^* + \epsilon$

$= (aa)^* + (aa)^*a$

→ $b^*ab^* + (a+b)^*a(a+b)^*a(a+b)^*$

more than 1 a

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

→ (

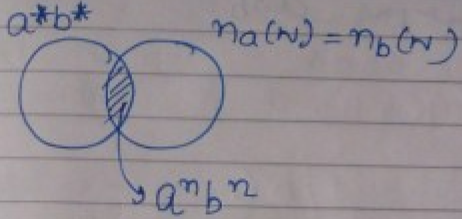
☆ In ~~most~~ cases

Date:
Page No.

→ $L(00^*) \cap L(0^*)$
$L(00^*)$

→ $L(a^*b^*) \cap L(n_a(w) = n_b(w))$ → this contains both $a^n b^n$ & $b^n a^n$, $n \geq 0$

$L(a^n b^n)$



Identities

① $\phi^* = \epsilon$
② $\epsilon^* = \epsilon$
③ $r_1 . r_2 \neq r_2 . r_1$
④ $r_1 . (r_2 . r_3) = (r_1 . r_2) . r_3$ e.g. $a(b a^* c^*) = (a.b). a^* c^*$
⑤ $r_1 . (r_2 + r_3) = r_1 r_2 + r_1 r_3$
⑥ $r_1 + (r_2 . r_3) \neq (r_1 + r_2) . (r_1 + r_3)$
⑦ $r . \epsilon = r$
⑧ $r . \phi = \phi$
⑨ $r . r^2 = r^2 r$
⑩ $\epsilon + r r^* = r^*$
⑪ $p(qp)^* = (pq)^* p$ { e.g. $ab^* (b a^* a b^*)^* = (ab^* b a^*)^* ab^*$
⑫ $(a+b)^* = (a^* + b^*)^* = (a^* b^*)^*$

$(a^* + b^*)^* = (\epsilon + a + aa + aaa + \dots + \epsilon + b + bb + bbb + \dots)^*$
$(a^* b^*)^* = (\epsilon + a^* + b^* + a^* b^*)^*$
$(r + s + t)^* = (r s^* + s r^* + r^* s^* t)^*$

Date

24.03.12

1. $(r)^* = r^*$ [e.g. $(a)^* = a^*$]

★ when r implies just a single symbol.

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

$\epsilon (0+1)^* \subseteq 0^*(0+1)^*$

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

↑ even length string — ↑ odd length string.

★ Universal set + Some set = Universal set

$(0+1)^* 0^* 1 0^* \not\equiv (0+1)^*$

as this contains atleast one '1'

★ For these two to be equivalent, then anything other than $(0+1)^*$ should have * above it.

$(0+1)^* \subset (0+1)^* 1^* 2^*$

★ as the 2nd one contains powers of 2 also, which can't be generated by the left one.

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

as U+A = U

U:- Universal set

A:- Some set.

$(01)^*0 + 0^* = (01)^*0 + 0^+ + \epsilon$ [0^+ can't produce 'ϵ'. So we add 'ϵ' as well.]

★ $1^*1 = 1.1^*$

Identities

- $r + r = r$
- $r.r = r^2$
- $r.\epsilon = r$
- $r + \epsilon = r$ [iff r contains ϵ string.]
  in other case wher ϵ isn't in r.
  $\therefore r + \epsilon \neq r$
- $\epsilon^* = \epsilon$
- $\phi^* = \epsilon$
- $\phi^+ = \phi$
- $\epsilon^+ = \epsilon$
- $\epsilon + rr^* = r^*$
- $\epsilon + rr^+ \neq r^*$
  $\epsilon + rr^+ = \epsilon + rr.r^*$ which doesn't contains r.
- $(r+s)^* = (r^* + s^*)^* = (r^* s^*)^*$
  $(r^* + s^*)^* \neq (r^* + s^*)$
  as $(r^* + s^*)$ will contain only runs of r & runs of s & not runs of rs or sr.
- $(pq)^* p = p(qp)^*$
- $\phi.r = r.\phi = \phi$
- $r + \phi = r = \phi + r$
- $r^* r^* = r^*$
- $(r^*)^* = r^*$

Simplify:

1. $(1+00^*1) + (1+00^*1)(0+10^*1)^*(0+10^*1)$

Ans:- $0^*1(0+10^*1)^*$

Soln:-

$(1+00^*1)\,[\epsilon + (0+10^*1)^*(0+10^*1)]$

$= (1+00^*1)\,[(0+10^*1)^*]$

$= (\epsilon + 00^*)1\,(0+10^*1)^*$

$= 0^*1(0+10^*1)^*$

Date:
Page No.

(2) $\lambda + 1^*(011)^*(1^*(011)^*)^*$

$= 1^*(011)^*(1^*(011)^*)^*$

$= (1^*(011)^*)^*$

$= (1+011)^*$

here, all binary strings having '0' will be followed by '11'.
& not (no two consecutive zeroes)
as 010 has no 2 consecutive '0's, but is not generated by this regular expression

$(1+01)^*$
every 0 is followed by 1, & not (no 2 consecutive zeroes) as 0 has no two consecutive zeroes but is not generated by the given regular expression.

(3) $(00+01+10+11)^* = ((0+1)(0+1))^*$
all even length strings.
$(00+01+10+11)^+$
all even length strings except 'λ'.

(4) $r = (00+01+10)^* \subset (00+01+10+11)^*$
as 11 is not produced by r.
every ~~lengt~~ string produced by r ~~contains~~ has even length & not r produces every even length string.

(5) $(0+1)^*00(0+1)^* \subset (0+1)^*0(0+1)^*0(0+1)^*$

↑ contains atleast two consecutive zeroes.

↑ contains atleast 2 zeroes.

★ Language can't be converted to machine by automata,
but regular expression can be converted to machine by automata. [because regular expression is formal definition.]

Containing atleast 2 zeroes.

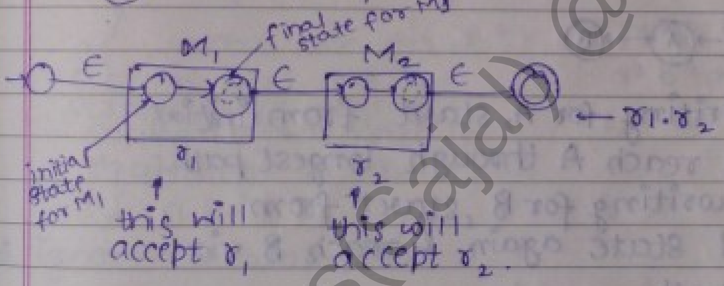
6. $1^*0\,1^*0\,(0+1)^* = (0+1)^*0\,(0+1)^*0\,(0+1)^* = (0+1)^*\,0\,1^*\,0\,1^*$

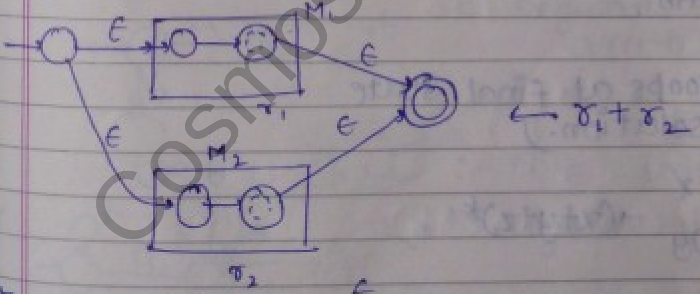
7. $(0+1)(0+1)^*(0+1) = (0+1)^*(0+1)(0+1) = (0+1)(0+1)(0+1)^*$
strings of length $2x+2$

Converting Machine to Regular Expression:-

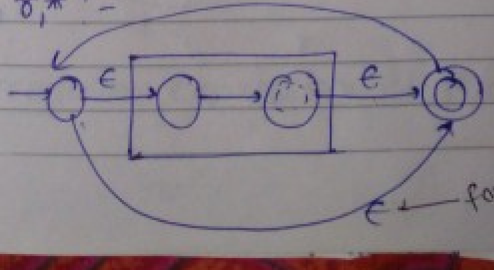
★ Regular Expression is formal definition of the language.

for ϕ :-





for r_1^* :-

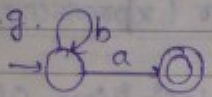


$r_1^* = \epsilon + r_1 + r_1.r_1 + r_1.r_1.r_1 + \ldots$

Machine To Regular Theorem :-

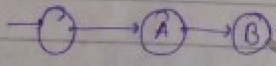
Guidelines :-

1. Minimal
   Take the longest path from initial state to the final state.
   e.g.



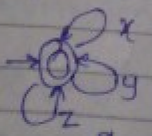
the longest path will be b^*a.

2. Several final states :-
   $r_A + r_B + r_C$



for writing for A, start from initial state & reach A through longest path. & for writing for B, start from initial state again & reach B via longest path.

3. Several loops at final state (loop resolution.)

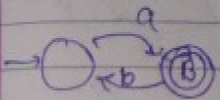


$\rightarrow (x+y+z)^*$

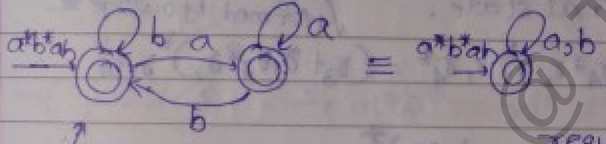
4. 

Final state on left :- no choice
Final state on right :- Choice

* Now, we will take (ab)*
when final state is on left, the expression has no choice



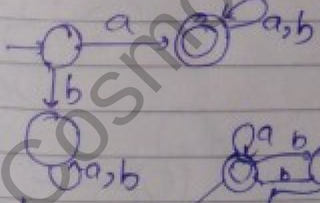
but when final state is on right side we have a choice.
(ab)*a
or a(ba)*

5. 

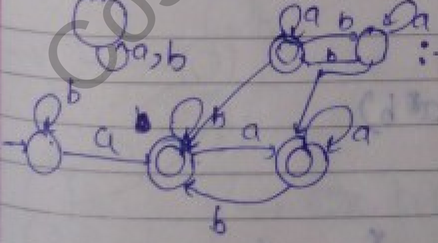
as we can see that at this point all the strings are accepted so, we convert into its equivalent machine.

regular expression :- a*b*ab(a+b)*

6. Ignore the trap state.



∴ regular expression :- a(a+b)*

7. 

:- this is unreachable.

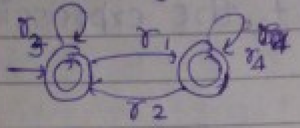
Ignore the unreachable state.

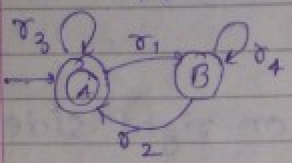
∴ regular expression :- b*a(a+b)*

Date:
Page No.

Examples:-

(8) Converting n-states to 1-state.





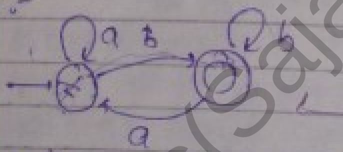
$r_A = (r_3^* + r_1 r_4^* r_2)^* \equiv (r_3 + r_1 r_4^* r_2)^*$ because r_3^* will be formed by outside *.

When B is only final state.

$\rightarrow r_B = (r_3^* + r_1 r_4^* r_2)^* r_1 r_4^* = (r_3 + r_1 r_4^* r_2)^* r_1 r_4^*$

$r_B = r_3^* r_1 (r_4 + r_2 r_3^* r_1)^*$

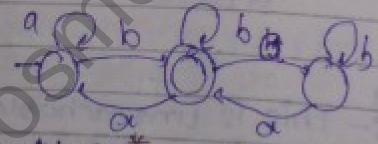
examples:-



a. $(a+bb^*b)^* bb^*$

b. $a^*b(b^* + aa^*b)^*$

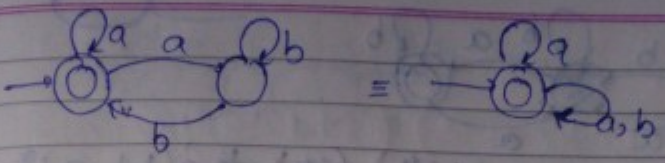
$a^*b + bb^* + bb^*aab$

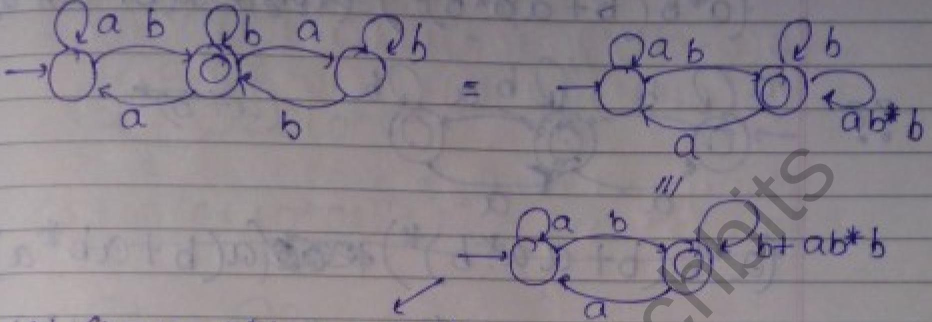


$a^*b(b^* + ab^*a + aa^*b)$

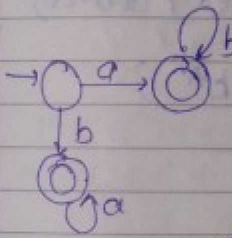
$= (a + bb^*a)^* bb^*$

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

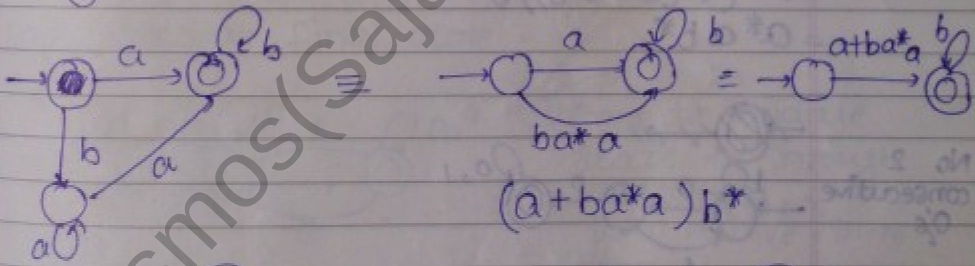




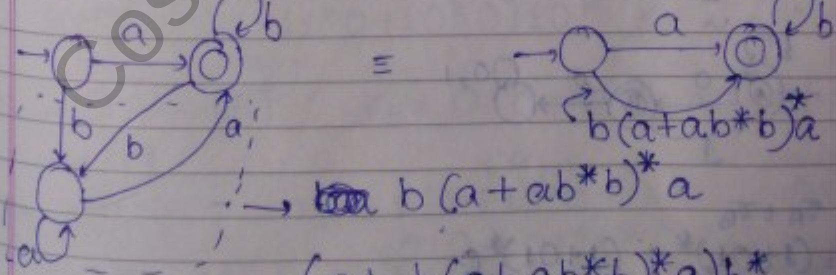
$a^*b\,(b + ab^*b + aa^*b)$



$\rightarrow ab^* + ba^*$



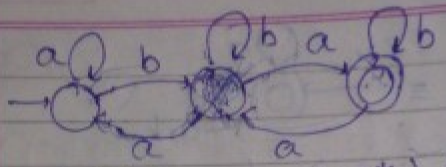
$(a + ba^*a)\,b^*$



$\rightarrow b\,(a + ab^*b)^*\,a$

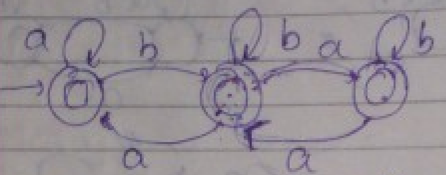
$(a + b(a + ab^*b)^*a)\,b^*$

??



$(a^*b(b+aa^*b)^*)\ a(b+ab^*a)^*$

??



$(a+b)^*$

$(a^*b(b+aa^*b)^*)\ (a(b+ab^*a)^*)$

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

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

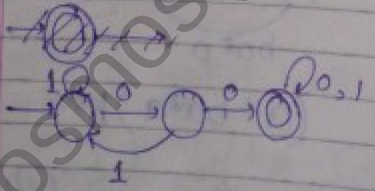
$a^*b(b+aa^*b+ab^*a)^* ab^*)$

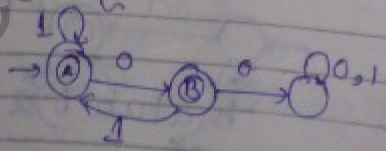
1. $a^*(a+ab^*b)b^*$

$= a^*(a(\epsilon+b^*b))b^*$

$= a^*ab^*$

No 2 consecutive 0's



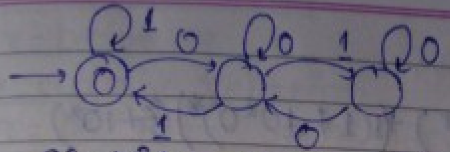


$r_A + r_B$

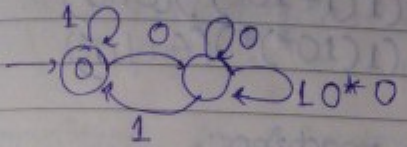
$= (1+01)^* + (1+01)^*0$

$= (1+01)^*(\epsilon+0)$

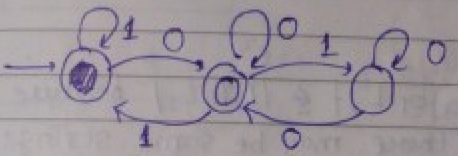
$= (1+01)^* + 1^*0(11^*0)^*$



Simplify the 3 states to 2



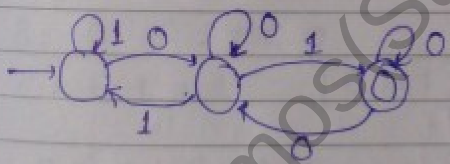
$(1 + 0(0 + 10^*0)^*1)^*$



Come to middle state & then resolve everything i.e. go through every loop.

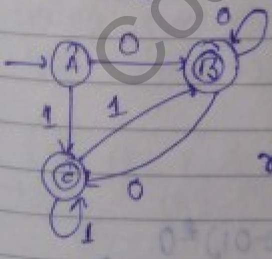
$1^*0\,(0 + 11^*0 + 10^*0)^*$
$= 1^*0\,(0 + 1(1^* + 0^*)0)^*$

→ Last state as final state :-



Come to middle state through all states & then resolve them

$1^*0\,(0 + 11^*0 + 10^*0)10^*$



$r_B = 0(0 + 01^*1)^* + 1(1 + 10^*0)^*10^*$ (final state for C.)

$r_C = 1(1 + 10^*0)^* + 0(0 + 01^*1)^*01^*$ (final state for B.)

$r_B + r_C =$

$(0(0+01^*1)^*)(\epsilon+01^*) + (1(1+10^*0)^*)(\epsilon+10^*)$

$= (0(0+01^*)^*)(\epsilon+01^*) + (1(1+10^*)^*)(\epsilon+10^*)$

$= (0(01^*)^*)(\epsilon+01^*) + (1(10^*)^*)(\epsilon+10^*)$

Mostly used identity in machines:-

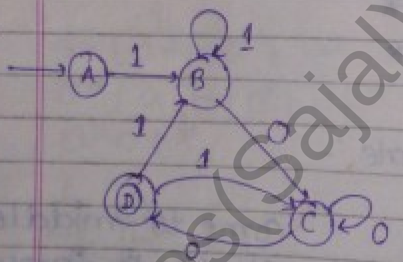
$\boxed{\epsilon + rr^* = r^*}$

★ $L_1 . L_2 \neq L_2 . L_1$

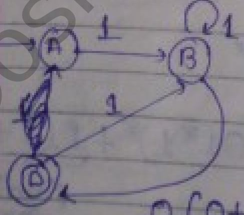
★ $|L_1 . L_2| \neq |L_2 . L_1|$

★ $|L_1 . L_2| \leq |L_1| . |L_2|$ } → $|L_1 . L_2| \leq |L_1| |L_2|$ because there may be some strings which are common in both L_1 & L_2.

Q.



|||



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

or ~~....~~ $1.1^* 0(0+01)^*0 (1.1^*0(0+01)^*0)^*$

$= (1.1^* 0(0+01)^* 0)^+$

Arden's Theorem:-

$r = p + rq$

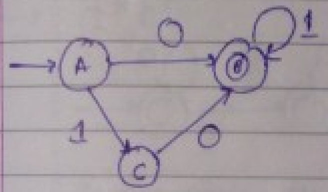
recurssive relation

$\boxed{r = pq^*}$ ⟶ soln. for $\boxed{r = p + rq}$

$pq^* = p + (pq^*)q$
$= p(\epsilon + q^*q)$
$= pq^*$

Condition for Arden's Theorem:-

$\boxed{\epsilon \notin L(q)}$



1. set up an equation for every state.

eqn. for ~~state~~ the incoming strings.

A: $r_A = \epsilon$
B: $r_B = A.0 + B.1 + C.0$
C: $r_C = A.1$

$r_A = A$
$r_B = B$
$r_C = C$

$B = 0 + B.1 + C.0$ $[A = \epsilon \ \& \ \epsilon.0 = 0]$

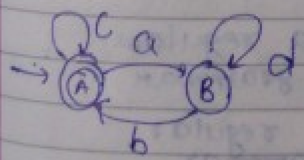
$C = \epsilon.1 = 1$ $[\epsilon.1 = 1]$

$\therefore B = 0 + B.1 + 10$

$B = (0 + 10) + B.1$

using arden's theorem

$r_B = (0 + 10)1^*$



$A = \epsilon + A.c + B.b$

$B = A.a + B.d$ (with $A.a$ marked p and d marked q)

$B = A.ad^*$

now, $A = \epsilon + A.c + A.ad^*b$

$A = \epsilon + A(c + ad^*b)$

$\therefore A = \epsilon.(c + ad^*b)^*$
$= (c + ad^*b)^*$

Regular grammar for L → L is regular.
Right linear Regular grammar for L → L is regular.

Date:
Page No.

Regular Grammar

Grammar is unique, i.e.

☆ if grammar is given, only one language can be generated. but not vice-versa.

ϵ	$S \to \epsilon$
a	$S \to a$
a^*	$S \to aS \mid \lambda$ → stopper.
x^*y	$S \to xS \mid y$
yx^*	$S \to Sx \mid y$
a^*ba	$S \to aS \mid ba$
a^*+b^*	

↳ $G_1 \to L_1$ $\quad S_1 \to L_1$

$G_2 \to L_2$ $\quad S_2 \to L_2$

$\to L_1 \cup L_2$ $\quad S \to S_1 \mid S_2$

$\therefore S_1 \to aS_1 \mid \lambda$

$S_2 \to bS_2 \mid \lambda$

$S \to S_1 \mid S_2$

☆ $S \to aS \mid \epsilon \equiv S \to Sa \mid \epsilon$ $\quad$ $S \to aS \mid b$ will generate a^*b

but $S \to aS \mid b \not\equiv S \to Sa \mid b$ $\quad$ $S \to Sa \mid b$ will generate ba^*

↑ Right Linear Regular Grammar $\quad$ ↑ Left Linear Regular Grammar

☆ Language is regular iff ∃ a regular grammar

iff ∃ a right linear regular grammar,

because we can convert every left linear regular grammar into right linear regular grammar.

Right Linear $\xleftrightarrow[\text{interconvertible}]{}$ Left linear.

Date:
Page No.

★ $S \rightarrow aS | Sb | \epsilon$

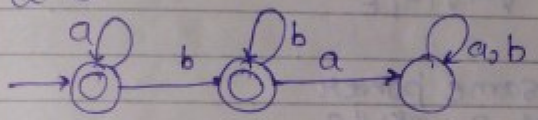
this is not a regular grammar as

$V \rightarrow T^*V + T^*$
$V \rightarrow VT^* + T^*$ } regular grammar

In a regular grammar, only one of them can exists, i.e. either only right linear regular grammar or only left linear regular grammar, but not both.

but this is a context free grammar
this grammar generates a^*b^*, which is a regular language, but every regular language is a CFL.

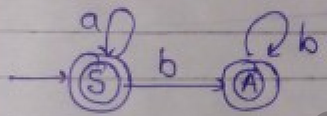
→ a^*b^*



Converting the machine to grammar.

1. Remove trap states & unreachable state.

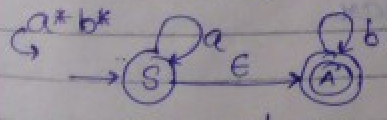
DFA:-



$S \rightarrow aS | bA | \epsilon$
$A \rightarrow bA | \epsilon$

$A \rightarrow b^*$
$\therefore S \rightarrow aS | bb^* | \epsilon$
$S \rightarrow a^*S | bb^* | \epsilon$
$S \rightarrow a^*(\epsilon + bb^*)$ $[bb^* = b^+,\ b^+ + \epsilon = b^*]$
$S \rightarrow a^*b^*$

NFA:-



$S \rightarrow aS | \epsilon . A$
or $S \rightarrow aS | A$
$A \rightarrow bA | \epsilon$

Date:
Page No.

→ $S \to abS \mid \epsilon$ $(ab)^*$

$S \to aA \mid \epsilon$
$A \to bS$

→ $S \to abcS \mid \epsilon$
$S \to aA \mid \epsilon$ $(abc)^*$
$A \to bB$
$B \to cS$

★ $V \to TV \mid \epsilon$ has same power as
$V \to T^*V \mid \epsilon$ & $V \to VT^* \mid \epsilon$

& $S \to TV$
$S \to T$
$S \to \epsilon$
} → same power as $S \to T^*V \mid \epsilon$

$V \to TV$

$\lambda \; \epsilon \; \phi$
$\lambda = \epsilon$
$L(\epsilon) = \{\epsilon\}$
$L(\phi) = \{\}$

★ Grammar that accept nothing:-
$S \to aS$

$(ab)^*$ $S \to abS \mid \epsilon$
a^*b^* $S \to aSb \mid \epsilon$, $S \to aS \mid Sb \mid \epsilon$ — not regular, as it contains both ~~left linear~~ T^*V & VT^* at same time.
$(a+b)^*$ $S \to aS \mid bS \mid \epsilon$ → regular
$S \to Sa \mid Sb \mid \epsilon$

?? $S \to a$ $S \to aS$
$S \to b$ $S \to bS$

We can create L_1^* from:-

$(a+b)^*$ $S \to a | b | SS | \epsilon$ ⟶ this is not regular

which of the following is incorrect.

?? $G: S \to aa | bb | SS | \epsilon$

(a) $L(G)$ is ambigous.
(b) $x(\epsilon) \in L(G)$ & $y \in L(g) \Rightarrow xy \in L(g)$ correct
(c) $L(G)$ is accepted by dpda. correct
(d) None of these correct

★★ Imp. If any grammar contains
$S \to SS | \epsilon$,
then the ~~gramma~~ language is ambiguous.

✓I. A CFG can create Regular language.
✓II. A Regular can " " "
✓III. A CFG can create any " "
✓IV. A Regular " " " " "
{ using Chomsky Heirarchy (The Types of grammar)

✓V. A Regular grammar can create a CFL.
✗VI. A Regular grammar " " any CFL.

$(aa + ba + bc)^*$

Grammar:- $S \to aa | ba | bc | SS | \epsilon$
or $S \to aaS | baS | bcS | \epsilon$.

$S \to S_1 \| S_2$ $S_1 \to aS_1 \| \epsilon$ $S_2 \to bS_2 \| \epsilon$	$L_1 \cup L_2$

↑ regular grammar

$S \to AB$ $A \to aA \| \epsilon$ $B \to bB \| \epsilon$	$L_1 . L_2$ (e.g. $a^* b^*$)

↑ context free language. (as R.H.S. contains 2 variables)

Date: Page No.

★★

$S \to S_1 | S_2$

G_1 : $S_1 \to$

G_2 : $S_2 \to$

→ both are regular

↓ but the complete is not regular, because G_1 & G_2 might not be both right linear or left linear.

★ Closure property :-

If L_1 is regular & L_2 is regular, then $L_1 . L_2$ has to be regular. (but we can't say anything about grammar)

★ Regular is not closed under subset, under superset, & under infinite union.

e.g.

$\{ab\} \cup \{aabb\} \cup \{aaabbb\} \cup \ldots$

though these are individually regular, but

$\{a^n b^n, | n \geq 1\}$

which is not regular.

★ $\supseteq$, $\subseteq$, infinite union

(not covered under closure property.)

L_1^*

$S \to SS | \epsilon$

$S \to L_1$

★ If L_1 is regular, L_2 is regular, $L_1 \cap L_2$ is surely regular.

$G \rightarrow L$

$G^c \rightarrow L^c$ (only when we can construct a dfa machine for G.)

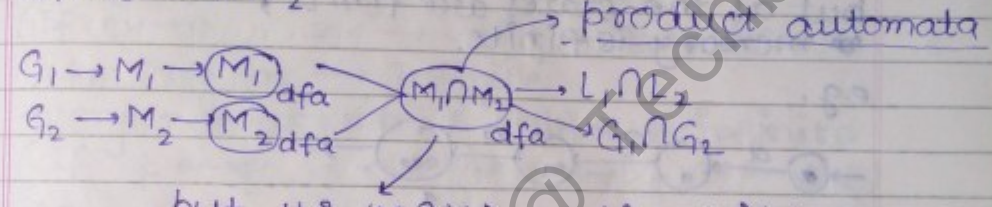
$G \downarrow M_{nfa} \longrightarrow M_{dfa} \longrightarrow M^c \longrightarrow G^c \longrightarrow L^c$

☆ The Machine formed by grammar should be DFA only then we can complement the grammar.

→ $G_1 \rightarrow L_1$

$G_2 \rightarrow L_2$

$G_1 \cap G_2 \rightarrow L_1 \cap L_2$



but this $(M_1 \cap M_2)$ machine will have several trap states

☆ $M_1 \times M_2 \longrightarrow L_1 \cap L_2$ $[M_1 \times M_2 = M_1 \cap M_2]$

Q. $S \rightarrow aABab$ $\quad S \rightarrow aa^*ac\,adb^*ab$

$A \rightarrow aA \mid ac$ $\quad A \rightarrow a^*ac$

$B \rightarrow Bb \mid ad$ $\quad B \rightarrow adb^*$

Language = $\{a^m c\,a\,d\,b^n ab \mid m \geq 2, n \geq 0\}$

$a^m b^{m+1}, m \geq 0$

$S \rightarrow S_1 b$

$S_1 \rightarrow aS_1 b \mid \epsilon$ ⟶ this is not regular as this is equivalent to (RHS) T^*VT^*

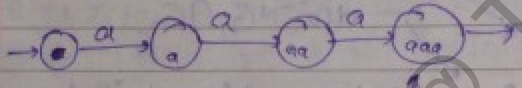
$\{a^m b^{m+1} \mid m \leq 10\} \rightarrow$ this i

Date:
Page No.

$\{a^m b^m bbb \mid m \geq 0\} \equiv \{a^m bbb b^m \mid m \geq 0\}$
the grammar for these are:-
$S_1 \rightarrow aS_1b \mid bbb.$

☆☆ $\{a^m b^m \mid m \leq 10\}$
↓
regular
the Finite automata has memory in form of states.
but as the states are finite, the memory is finite.

e.g.



but as no. of states are finite,
∴ $a^m b^m, m \geq 0$ is not regular as the max. no. of states req. is infinite, but dfa has finite states, so it is not regular.

Linear Grammar

Right Linear	Left Linear	Not Regular
$T^*V + T^*$	$VT^* + T^*$	

The R.H.S. should have atmost one variable.
It allows:-
$VT^* + T^*$
$T^*V + T^*$
$T^* V T^*$ → this isn't allowed in regular.

Date: 2

Grammar

★ Regular Language is a proper subset of Linear Grammar.

→ $a^n b^n$:- $S \to aSb | \epsilon$

→ $n_a(w) = n_b(w)$:- $S \to aSb | bSa | SS | \epsilon$

Identify Regular, CFL, CSL

(Follow these steps stepwise)

1. Finite → Regular (If the language is infinite, then go to step 2)
every finite language is regular.
(If $p \to q$, $\therefore \sim q \to \sim p$) contrapositive.
Non regular → Infinite.
every non-regular is always infinite.
(If $p \leftrightarrow q$, $\therefore \sim p \leftrightarrow \sim q$, $q \leftrightarrow p$, $\sim q \leftrightarrow \sim p$.)
e.g.
equivalent { Every regular language has regular expression. ($p \to q$)
Every regular expression has regular language ($q \to p$)
~~Every non~~ ($\sim p \to \sim q$) & ($\sim q \to \sim p$).
$p \to q$
converse :- ~~q→p~~ $q \to p$
inverse :- ~~q→p~~ $\sim p \to \sim q$.

★★ The condition in language must be finite for a regular language. e.g. $\{a^m b^n | m \le 5, n \le 10\}$ → regular
$\{a^m b^n | m \ge 0, n \ge 10\}$ → irregular

2. Comparison

$\{a^m b^{2m+3} | m \ge 0\}$
not regular.

$\{a^m b^n | m = n, m, n \ge 0\}$
not regular.

$=, <, \le, >, \ge, \ne$ → these operators should have variables on both sides. → comparison operators.

If there is a comparison, then the language is not regular.

Date:
Page No.

☆ $\{a^m b^n \mid m=n \ \& \ m,n \le 10\}$
this is regular as this is finite.

The PDA can do more than one comparison but not on more than 2 variables.

e.g.
$\{a^m b^n c^p \mid m=n \ \& \ m=p\}$
this can't be done via PDA
but
$\{a^m b^m c^n d^n \mid m,n \ge 0\}$
but this can be done via PDA.

3. String Matching

$L = \{ww^R \mid w \in (0,1)^*\}$ → even pallindrome
The string matching can be done via PDA, but not FA.

The PDA can't perform ww, as once pushed into stack, the variables will come in reverse order.
So, (ww) is CSL.

$\{a^m b^n \mid m,n \ge 0\}$ → this is regular (equivalent to $a^* b^*$) as there is no comparison or string matching.

$\{a^m b^n c^p \mid m,n \ge 6 \ \& \ p \ge 6\}$
this is regular as well according to the same concept discussed above.

4. Comparison —
- one comparison on one variable at a time (non-regular CFL)
- two or more comparison on same variable at a time (CSL)

?? → $1(0+1)^*1 + 0(0+1)^*0$
this is regular.

Date:
Page No.

5. String Matching → ww^R :- CFL
→ ww :- CSL

FACTS:-

① Mod machines are always regular as they are always finite.
e.g.
$\{w \mid |w| \bmod 3 = 2 \ \& \ n_a(w) \bmod 6 = 2\}$
as the no. of states are finite, ∴ the mod machines are regular.

② $L_1 \cup L_2 \cup L_3$
If L_1, L_2, L_3 is regular, then the whole union is regular according to closure property.

$L_1 = \{w_1 a w_2 \mid w_1, w_2 \in [0,1]^*\}$ → regular
$L_2 = \{w_1 a w_1 \mid w_1 \in [0,1]^*\}$ → not regular
as there is comparison b/w the two strings, ∴ L_2 is CSL as the string matching involves ww.

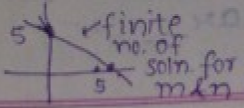
③ $\Sigma \in \{a\}$
$\{a^{n^2} \mid n \geq 5\}$ → regular (as no comparison.)
$\{a^{2n} \mid n \geq 0\}$ → regular (as no comparisons)
$\{a^{c_1 n + c_2} \mid n \geq 0\}$ → regular (as " ")

(exception to finite language rule)

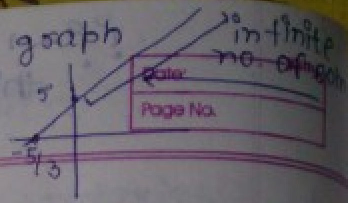
☆ The power of a should be in linear form i.e. should be in form of $c_1 n + c_2$, & not $c_1 n^2 + c_2$ or $c_1 n^3 + c_2$, & so on...

for such questions draw the graph

$m+n=5$
$\Rightarrow \frac{m}{5}+\frac{n}{5}=1$



$n=3m+5$
$\Rightarrow -3m+n=5$
$\Rightarrow \frac{m}{-5/3}+\frac{n}{5}=1$



Date:
Page No.

★★

④ PDA can't perform :-

~~$\{w \mid n_a(w) \times n_b(w) ...$~~

$\{w \mid n_a(w).n_b(w) \geq 5\}$ → not regular, or CFL.

This is not regular as the no. of soln for $n_a(w)$ & $n_b(w)$ will be infinite

but

Exception: $\{w \mid n_a(w).n_b(w) = 5\}$ is regular, as $n_a(w).n_b(w)=5$ is finite, ∴ it is regular.

← This is regular because the no. of solutions will be finite

$\{w \mid n_a(w)+n_b(w)=5\}$ is regular, because $n_a(w)+n_b(w)=5$, which is finite.

~~$\{w \mid m+n$~~ =

$\{a^m b^n \mid m+n=5\}$ is regular.

← This is regular as the soln for $m+n=5$ will have finite no. of soln for m & n.

★★ sum & multiplication is regular, but division & subtraction is not regular as division & subtraction is infinite as $\frac{m}{n}$ has infinite soln. for m, n.

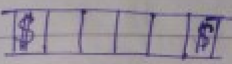
$\{a^m b^n \mid m+n=5$ → regular
$\mid m \times n = 5$ → regular
$\mid m-n=5$ → not regular
$\mid m/n=5$ → not regular
$\mid m = 5-n$ → regular
$\equiv m+n=5$

✓ Imp. $\{a^m b^n \mid n = 3m+5\}$ → CFL & not regular as no. of m's & n's will become infinite.

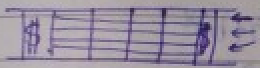
★ Any power which becomes non-linear, then it becomes CSL.

e.g. $\{a^m b^n \mid n = m^2, m \geq 0\}$ → not CFL, but CSL.

Turing Machine



The machine has various tracks, it is in the form of :-



★ Imp.

1. Check whether the language is finite.
2. Check whether the power of symbols are linear or not. If power is non-linear then it is a CSL.

Pumping Lemma

If L is regular, then it will surely satisfy pumping lemma for regular.

L is regular ⟹ L satisfy P.L. (for regular), but not that if L satisfy P.L., then L is regular.

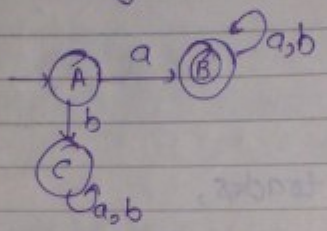
Ma M.N

L is regular ⟺ No. of M.N. is finite

If L doesn't satisfy P.L. ⇒ L is not regular.

No. of M.N. states = no. of states in minimal dfa

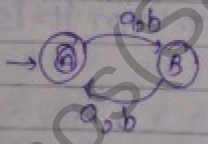
1. Starting with "a".



{ϵ} → reg. expression of A
reg. expression of B :- a(a+b)*
reg. expression of C :- b(a+b)*

∴ {ϵ}, {a(a+b)*}, {b(a+b)*}

2. $L = \{w \mid |w| \bmod 2 = 0\}$



reg. expression of A :- {(a+b)(a+b)*}
reg. expression of B :- {(a+b)(a+b)}*(a+b)

Pumping Lemma :-

① If L is regular ∃ a dfa M that accepts L.

So, $w \in L(M)$

$|w| \geq N$ [N:- no. of states in dfa]

then

$w = xyz$

$|y| \geq 1$

so, if $xyz \in L(M)$

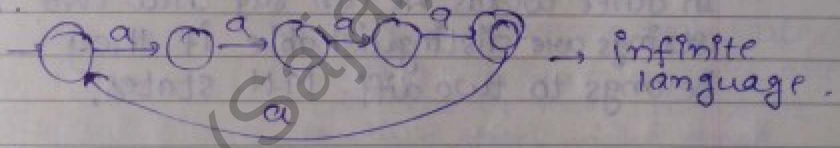
then the machine will also accept

$xy^iz \in L(M)$, where $i = 0, 1, 2, 3, \ldots$

In other words, an N state machine which can accept (N-1) symbols, then to accept more symbols, then there must be a loop.

★ If there is a dfa which has 5 states & it accepts $|w| = 6$, then that language is infinite, as there will be a loop.

e.g.



→ every infinite language satisfies pumping lemma, but every finite language also satisfies pumping lemma.

because

in finite language, $|w| \geq N$ is false, so the finite language trivially satisfies pumping lemma.

→ If $|w| < N$, then we can't say that language is finite or not because pumping lemma say nothing about $|w| < N$, but only $|w| \geq N$.

Date

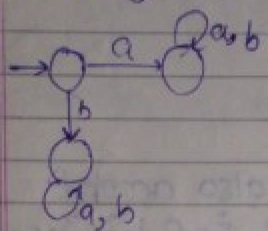
5.03.12

★ A Transition diagram is that which contains more than one starting state.

Date:
Page No.

Distinguishability

Acc. to Myhill Nerode Theorem:- Myhill Nerode
L is regular ⇔ No. of classes is finite



✓(a) aab, ε (distinguishable)
(b) aab, bbb
(c) bbb, ba
(d) NOT

M.N. equivalence classes:- ε, a (a+b)*, b (a+b)*

Two strings are equivalent if they take the system to same state

★ x & y is distinguishable iff.
$\delta^*(q_0, x) \neq \delta^*(q_0, y)$
i.e. two strings are distinguishable when the two strings takes the F.A. into 2 different states.
In other words we can say that two strings are distinguishable if they belongs to two diff. M.N. states.

★ Two strings are related iff
$\delta^*(a, x) = \delta^*(a, y)$
i.e belonging to same M.N. state

★ M.N. relation is right invariant relation

Right Invariant
If u & v are equivalent strings then concatenating strings w to them creates two strings uw & vw which will also be equivalent.

Algorithms

1. NFA → DFA (subset construction algorithm)
2. DFA → min. DFA
3. ε-closure (nfa with ε-move to nfa w/o ε move)

for every nfa, there is an equivalent DFA for it.

Date:
Page No. 43

★ NFA → DFA (or Transition → DFA)

δ	a	b
→ q_0	$q_0 q_1$	q_2
q_1	q_0	q_1
(q_2)	-	$q_0 q_1$

There is no ∈-move defined in this NFA.

★ This algo. can only be applied when there is no ∈-move in NFA, but if there is an ∈-move in NFA, then 1st apply ∈-closure algorithm.

★★ imp. NFA with N states ≡ DFA with atmost 2^N states.

Conversion:-

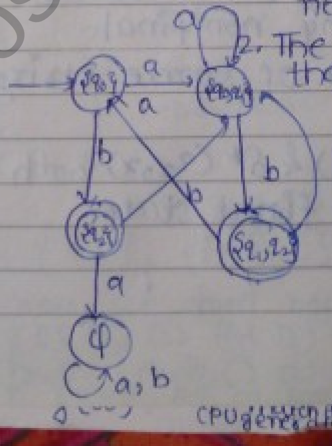
δ	a	b
→ $\{q_0\}$	$\{q_0,q_1\}$	$\{q_2\}$
$\{q_0,q_1\}$	$\{q_0,q_1\} \cup \{q_0\}$	$\{q_1\} \cup \{q_2\}$
($\{q_2\}$)	$\{\ \}$	$\{q_0,q_1\}$
φ	φ	φ
($\{q_2,q_1\}$)	$\{q_0\} \cup \{φ\} = \{q_0\}$	$\{q_0,q_1\}$

For every new state, we go on acc. to that states.

★ The φ will be the trap state.

1. Start with q_0, & if there are two or more choices in the nfa, then make those states as 1 union of states. e.g. when a is input at q_0, the next state will be a union of $\{q_0\}$ & $\{q_1\}$, ∴ $\{q_0,q_1\}$ will be the combined new state.
2. The final states will be those that contains q_2

DFA:-



✓ Imp

M → nfa
M' → equivalent dfa.

$M \rightarrow (Q_1, \Sigma_1, \delta_1, q_{01}, F_1)$

≡dfa $M' \rightarrow (Q_2, \Sigma_2, \delta_2, q_{02}, F_2)$

- $\Sigma_1 = \Sigma_2$
- $Q_2 \subseteq 2^{Q_1}$ → Q_2 contains states which are subsets of power set 2^{Q_1}.
- $|Q_2| \leq 2^{|Q_1|}$
- $q_{02} = \{q_{01}\}$
- There is no connection b/w F_1 & F_2.

~~$F_2 \subseteq F_1$~~

~~$\{F_1 \ldots F_2\}$~~

$F_2 \subseteq Q_2 \subseteq 2^{Q_1}$

$\therefore F_2 \subseteq 2^{Q_1}$

☆ The NFA → DFA algorithm the formed DFA may or may not be minimal

Algorithm 2

(DFA → minimal DFA)

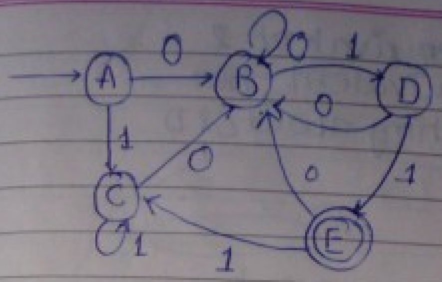
* Two or more states in a dfa might be doing the same job, these states are equivalent states.

* Two states are equivalent states if both goes to any final states or both goes to any non final states on occurence of same string

$q_1 \equiv q_2$ iff $\delta^*(q_1, x)$ & $\delta^*(q_2, x)$ both ends up in either final state or non-final state.

Imp: Actually, these two states can end up in diff. non-final states or diff. final states, even though then the states are equivalent.

Date:
Page No.



$Q = \{A, B, C, D, E\}$

1 Start with π_0

Create 1st partition
take all non-final states together & all final states together.

$\pi_0 = \{\{E\}, \{A, B, C, D\}\}$

π_0 :- Behaviour towards zero length string, so, behaviour of non-final states towards 0 length string is same, similar the case with final states.

π_1 :- Behaviour towards atmost 1 length string is same for both final states & non-final states

- Comparing A & B

$A \xrightarrow{0} B$ } Their behaviour is same for '0'.
$B \xrightarrow{0} B$

$A \xrightarrow{1} C$ } Since C, D are in same block in π_0, so their behaviour is same for '1'.
$B \xrightarrow{1} D$

- Comparing A & C

$A \xrightarrow{0} B$
$C \xrightarrow{0} B$
$A \xrightarrow{1} C$
$C \xrightarrow{1} C$
} same behaviour

- Comparing A & D

$A \xrightarrow{0} B$ } → same block
$D \xrightarrow{0} B$
$A \xrightarrow{1} C$ } → diff. block
$D \xrightarrow{1} E$
} diff. behaviour.

★★ [as we see '0' has similar behaviour, but '1' has diff behaviour, not equivalent]

$\therefore \pi_1 = \{\{E\}, \{A, B, C\}, \{D\}\}$

now, we dont need to check for (B & D) & (C & D) as (A & D) are not in same block but (A, B, C) are in same block, ∴ B/C & D can't be in same block.

??

★ If A & D are not equivalent & A&C are not equivalent, then we can't say anything about C & D.

Write π_2

$\pi_2 = \{\{E\}, \{D\}, \{A, C\}, \{C\}\}$

- Compare A & B

$A \xrightarrow{0} B$
$B \xrightarrow{0} B$
$A \xrightarrow{1} C$
$B \xrightarrow{1} D$

} As C & D are in diff. blocks in π_1, ∴ A & B are not equivalent.

- Compare A & C

$A \xrightarrow{0} B$
$A \xrightarrow{1} C$
$C \xrightarrow{0} B$
$C \xrightarrow{1} C$

} As these are in same block in π_1, ∴ A & C are equivalent.

∴ $\pi_2 = \{\{E\}, \{D\}, \{A, C\}, \{B\}, \{D\}\}$

→ Now, write π_3

- Compare A, C

?? → $A \xrightarrow{0} B$
$C \xrightarrow{0} B$
} These two are in same block in π_2

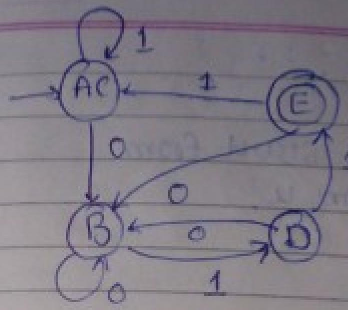
$A \xrightarrow{1} C$
$C \xrightarrow{1} C$
} These two are in same block in π_2

∴ $\pi_3 = \{\{E\}, \{A, C\}, \{B\}, \{D\}\}$

now,

$\boxed{\pi_2 = \pi_3}$

★ All trap states will always be equivalent.



→ Minimal DFA

★ No. of Minimal states in DFA is 4.

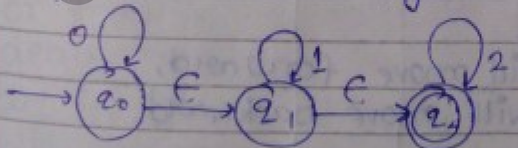
| 1 ≤ No. of states in Min. DFA ≤ N |

★ If we have non-reachable states, then these states will remain as such in minimal DFA.
If we have any non-reachable state, then first remove them & add them to dfa at the end.

Algorithm 3

ε - Closure :-
(NFA with ε move ⟶ transition system without ε move)

★ If we have to convert NFA with ε-move to DFA, we will use algo 3 followed by algo 1.



$u \xrightarrow{\epsilon} v$

★ To remove the ε move.

Step 1:- Duplicate every arrow start from v as if it starts from u.

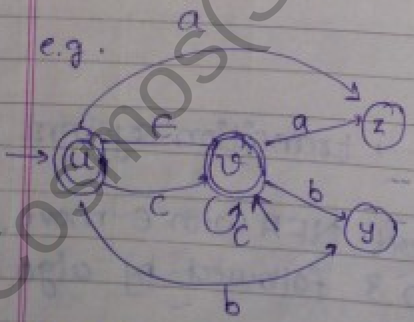
e.g.:

v has an arrow having gone to z when getting a at v, ∴ we duplicate arrow having a at u & going to z.

Step 2:- If u is the starting state, then make v as starting state as well.

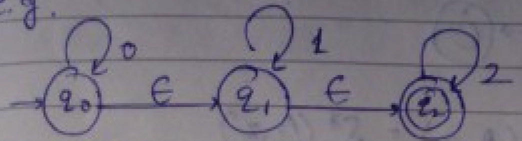
Step 3:- If v is the final state, then make u as the final state as well.

e.g.

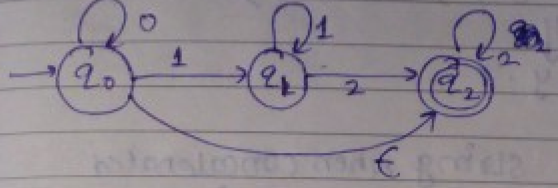


★ Starting state will move forward, & final state will move backward.

e.g.

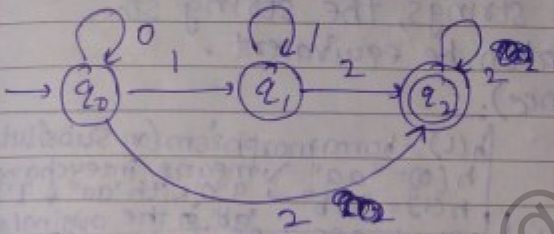


as we see that there are consecutive null moves from q_0 to q_2, i.e. $q_0 \xrightarrow{\epsilon} q_1 \xrightarrow{\epsilon} q_2$



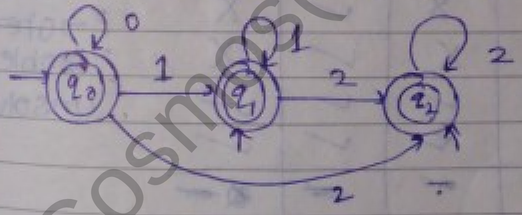
(Step 1) ∴ make a null move from $q_0 \to q_2$

when we remove null moves b/w (q_0 & q_1) & (q_1 & q_2)



(Step 1)

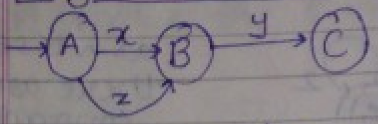
Step 2 & 3: As we can see q_0 is initial state, & there is ϵ-move b/w $q_0 \xrightarrow{\epsilon} q_1$ & $q_0 \xrightarrow{\epsilon} q_2$, ∴ make q_1 & q_2 as ~~final~~ initial states, & using step 3, make q_0 & q_1 as final states.



☆☆ IN NFA, if all states are final, then it may or may not accepts Σ^*, as it may contains dead configuration,
but in DFA, if all states are final, then it will accept Σ^*.

String equivalence

☆ Right invariant:



$x \equiv z$ iff $\delta^*(A,x) = \delta^*(A,z)$

$$\Downarrow$$

$$x.y \equiv z.y$$

i.e. the right string when concatenated with equivalent strings, the string so formed will also be equivalent (right invariance).

h(L):- homomorphism (or subsitution)
h(0) = "aa" } means interchange a '0' with 'aa' & '1' with
h(1) = "ab" } 'ab' in the original string.

Closure Table

	REG	DCFL	CFL	CSL	REC	RE
U	✓	×	✓	✓	✓	✓
∩	✓	×	×	✓	✓	✓
L^c	✓	✓	×	?	✓	×
.	✓	×	✓	✓	✓	✓
*	✓	×	✓	✓	✓	✓
$L_1 - L_2$	✓	×	×	×	✓	×
$L_1 \oplus L_2$	✓	×	×	×	✓	×
L∩R	✓	✓	✓	✓	✓	✓
L∪R	✓	×	✓	✓	✓	✓
h(L)	✓	×	✓	–	–	–
$h^{-1}(L)$	✓	✓	✓	–	–	–
L^R	✓	×	✓	–	–	–
INIT(L)	✓	✓	✓			
L/a	✓	✓	✓			
CYCLE	✓	✓	✓			
MIN(L)	✓	×	×			
MAX(L)	✓	×	×			
HALF(L)	✓	×	×			
ALT(L)	✓	×	×			

(U, ∩, L^c, ., *: used when all of the languages is reg.)
($L_1 - L_2$ to L∪R: atleast one language is reg.)

× → may or may not
e.g. under DCFL union of two DCFL may or may not be DCFL.

? → open problem/ unsolved.

e.g. $\Sigma = \{0,1\}$ $h(0) = aa$
$L = \{0, 1, 01\}$ $h(1) = bb$
$\therefore h(L) = \{aa, bb, aabb\}$
$h_{\Sigma \to \Gamma^*}$

If $L = \{aa, bb, aabb\}$
then $h^{-1}(L) = \{0, 1, 01\}$

Date:
Page No.

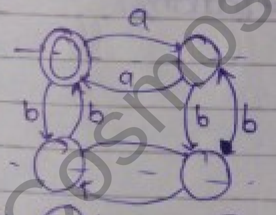
★ Subset is not under closure, because as $(a+b)^*$ is regular & every other language is subset of $(a+b)^*$, ∴ subset is not under closure.

★ Complement of a DCFL is always a DCFL.

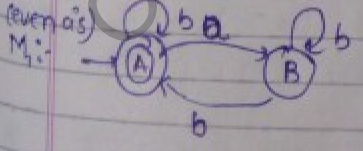
★ $L_1 - L_2 = L_1 \cap L_2^c$
both ∩ & complement must be closed for $L_1 - L_2$ to be closed.

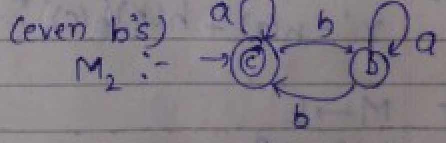
★ ∪ If L_1 & L_2 are both regular, then $L_1 \cup L_2$ is regular.
$L_1 \cup L_2$
↓ ↓
$r_1 + r_2$

Product Automata :-



even a's & even b's.





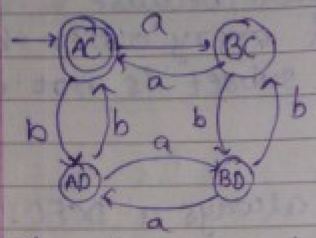
$M_1 (Q_1, \Sigma_1, \delta_1, q_{01}, F_1)$ $M_2 (Q_2, \Sigma_2, \delta_2, q_{02}, F_2)$
$M_1 \times M_2 (Q_1 \times Q_2, \Sigma, \delta_{12}, (q_{01}, q_{02}), (F_1, F_2))$ • $[\Sigma_1 = \Sigma_2]$

• $Q_1 \times Q_2 = \{AC, AD, BC, BD\}$

• starting state $= (q_{01}, q_{02}) = (A, C)$
• ending state $= (A, C)$

Algorithm

$\delta_{12}((A,C),a) = (\delta_1(A,a), \delta_2(C,a))$



δ_1	a	b
A	B	A
B	A	B

δ_2	a	b
C	C	D
D	D	C

L(M)

δ_{12}	a	b
→ (AC)	BC	AD
AD	BD	AC
BC	AC	BD
BD	AD	BC

Map this to given answer

★ If some language is closed under NAND then it is closed under all operations.

★ $r = a + bc^*$ $h(a) = 01$
$L \to r$ $h(b) = 11$
$h(L) \to ?$ $h(c) = 10$

$r' = h(a) + h(b)(h(c))^*$

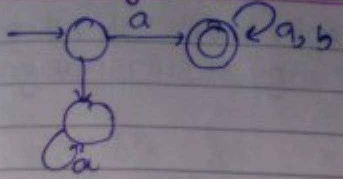
★ $M \to L$

$M^R \to L^R$

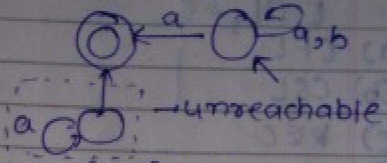
$\overline{M} \to \overline{L}$

• exchange the starting state & final state

L = starting with 'a' :-



L^R :- ending with 'a'
interchange starting state & final state.



step1 :- interchange starting state & final state.
step2 :- reverse all the arrows.
Result :- Machine will accept L^R.

★ INIT(L) = set of all prefixes of all symbols in L
e.g. $L = \{a, ab, bab\}$
prefix(a) = ϵ, a
prefix(ab) = ϵ, a, ab
prefix(bab) = ϵ, b, ba, bab
∴ INIT(L) = $\{\epsilon, a, ab, b, ba, bab\}$

→ $INIT(a^*b) = a^*b^*$
→ $INIT(a^nb^n) = \{a^mb^n \mid m \geq n\}$ as the prefix will always contains greater than or equal.
e.g. aabb
prefixes :- a $(m>n)$, aa $(m>n)$, aab $(m>n)$, aabb $(m=n)$

→ INIT(L)
where $L = \{w \mid n_0(w) = n_1(w)\}$ 1010
∴ $INIT(L) = (0+1)^*$ ~~ε, 1, 10, 101~~, 1010
as

→ $L = (0+1)^*$
$INIT(L) = (0+1)^*$

→ $L = (0+1)^+$
$INIT(L) = (0+1)^*$

★ ALT
take equal length string
[0110, 101] → [01101100, 100110]
[1010, 010]

Date:
Page No.

Usage Of Closure Table

1. L_1 is CFL & L_2 is CFL
   $L_1 \cap L_2 = ?$
(a) CFL
(b) CSL
(c) REC
(d) RE

★ if $L_1 \cap L_2$ has a ⊗ in CFL, then move towards right till we get a '✓' mark.
→ moving towards right from CFL, we get the tick mark at CSL, ∴ $L_1 \cap L_2$ is CSL.

★ when there is two different languages, then push the lower level language to the level of higher language & then resolve it.
e.g. CFL ∩ CSL
(L_1) (L_2)
push L_1 to CSL level & then solve.

★ For '−'
dont use the table & above two rules directly.
First convert '−' into ∩ & L^c.
e.g.
RE − REC
= RE ∩ $(REC)^c$
= RE ∩ REC (as complement operation on REC produces REC language.)
= RE ∩ RE → push REC to RE

sureshot

Date:

Page No.

??

** Dont use the closure property when the languages are given.

e.g.

$L_1 = a^n b^n$

$L_2 = (a+b)^*$

$L_1 \cup L_2 = (a+b)^*$ → regular

using closure property:-

$L_1 \cup L_2$

CFL ∪ Reg.

= CFL ∪ CFL

= CFL → weak answer

- as every regular can be said as a CFL.

??→* DCFL ∪ Regular

CFL ∪ Reg → CFL

DCFL is upgraded to CFL because Union is not defined under DCFL, so upgrading Regular to DCFL will make no difference, so we have upgraded DCFL to CFL & then used (LUR) case.

Decidability :-

Problem	REG	CFL	CSL	REC	RE
1. Membership	✓	✓	✓	✓	✗
Emptiness	✓	✓	✗	✗	✗
Finiteness	✓	✓	✗	✗	✗
Equivalence	✓	✗	✗	✗	✗
Regularity	✓	✗	✗	✗	✗
Ambiguity	✓	✗	✗	✗	✗
$L=\Sigma^*$?	✓	✗	✗	✗	✗
Disjointedness	✓	✗	✗	✗	✗

- ✗ means the problem can't be solved ever.

Regular :-

1. Membership :-
Given a language & $w \in \Sigma^*$, then does $w \in L$ or $w \notin L$.

2. Emptiness :-
Given a language L, whether the $L=\phi$ or $L \neq \phi$.

3. Finiteness :-
We are given language L, we have to check L is finite or infinite.

4. Equivalence :-
Given 2 languages, we have to check whether these 2 languages are equivalence.

5. Regularity

Given a language L, we have to decide whether the language L is regular.

6. Ambiguity

Given a language L, the language is ambigous ~~because~~ if all its grammars are ambigous.

Ambigous:-

$S \to aS|a|\epsilon$

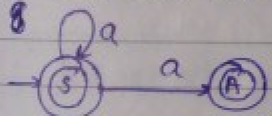
We can derive a through two diff. derivations,

- $S \to a$
- $S \to aS$
  $S \to \epsilon$
  $S \to a$

★ Every regular language ~~grammar~~ is unambigous.

$S \to aS|a|\epsilon$

to create nfa :-



convert the grammar as follows:-

$S \to aS|aA|\epsilon$

$A \to \epsilon$.

Ambigous CFL:-

$\{a^n b^n c^m\} \cup \{a^n b^m c^m\}$

Take e.g. if we create a grammar saying

$S \to S_1|S_2$

S_1 generating $a^n b^n c^m$

S_2 generating $a^n b^m c^m$

but the strings in the form $a^n b^n c^n$ will be derived both from $a^n b^n c^m$ & $a^n b^m c^m$, therefore producing ambiguity.

??

?? ☆ Ambiguity can be solved by left recursion.

Imp☆☆ Subset of regular is not regular.

7. $L = \Sigma^*$?

- Undecidable
  - Semi-Decidable (the problem may or may not be decidable) it will tell that ~~member~~ string belongs to language, but when string is not a member, then it may hangup.
  - Not even Semi-decidable: it may hang up for those string also that are members of the language

Membership

- Through machine, we can prove that whether given string belongs to the language or not. FA, PDA, LBA & HTM can never hang & hence will always tell whether the string is a member of the language or not. But Turing Machine hangs up & hence membership is undecidable.

Emptiness

In dfa, the language $L = \phi$ iff there is no path from initial state to final state. (final state is unreachable).

Finiteness

A language is finite iff ∃ no cycle in the ~~graph~~ directed path. i.e. the graph should not contain any cycle ~~fo~~ in the directed path from initial state to final state.

** Presence of '*' in the regular expression doesn't mean that the language is infinite.
e.g. $a^*.\phi = \phi$ which is a finite language.

* Absence of '*' surely means that language is finite.

Equivalence

Two languages are equivalent iff $L_1 \oplus L_2 = \phi$.
$L_1 = L_2$ iff $L_1 \oplus L_2 = \phi$.
This means, that there is no string which belongs to either L_1 or L_2 but not both.

$$L_1 \oplus L_2 = (L_1 - L_2) \cup (L_2 - L_1)$$
$$= (L_1 \cap L_2^c) \cup (L_2 \cap L_1^c)$$
$$= ((L_1 \cap L_2^c)^c \cap (L_2 \cap L_1^c)^c)^c$$

We have algorithm for this using complement (converting final state to initial state & vice versa) & intersection (product automata).

$L = \Sigma^*$?

$L = \Sigma^*$ iff $\bar{L} = \phi$
This can be done via making the machine for L, then complement the machine & check for the emptiness of the language.

Disjointedness ($L_1 \cap L_2 = \phi$)

Make the product automata for L_1 & L_2 & check for emptiness.

Date:
Page No.

Membership for CFL:-

The algorithm for this is CFK whose complexity is $O(n^3)$.

Applications

1. Lexical Analysis
   This part of compiler works on a fa machine.
2. Spell Checker
   e.g. Complate   compl(a+b+...+z)*
   Complete
3. Search Replace.
4. Grep. Command.
5. Neural Networks.
6. Storage
   (finite memory)
   e.g.
   C state store '01'.

7. FSM (Finite State Machine)
   Sequential Circuits.

Limitations

1. String Maching
2. Comparisons.

Variations

(1) 2-dfa & 2-nfa, (but this has the same power of the dfa & nfa) it can move the read head to both left & right

¢	a	b	a	$

↑ read head

$\delta(q_1, b) = (q_2, L)$

move read head to left & go to state q_2.

② FA + 1 stack ≡ pda

③ FA + 2 stack ≡ pda + 1 stack ≡ Turing Machine

we can give the command as follows :-

$\delta(q_0, q_1, a,$

④ nfa + 1 stack ≡ npda ≡ cfl

⑤ counter automata :-

CA → fa + 2 counter ≡ Turing Machine

Regular (fa) < (fa + 1 counter) < CFL (pda)

(fa + 1 counter) can accept $a^n b^n$, but not ww^R (as) $a^n b^n$ cant be accepted by fa, & ww^R is accepted by pda.

fa < fa + 1 counter < fa + 1 stack (pda) < fa + 2 counter, fa + 2 stack (TM)

fa + 3 counter, fa + 3 stacks → any no. of stacks or counters can be added, but the machine so formed will only be equivalent to TM.

★ (fa + 1 counter) can work with 1 symbol only. $a^n b^n$ can be checked by this.

★ if we remove the write capacity from turing machine, it will degenrate into fa.

Mealey & Moore Machine :-

output alphabet

$(Q, \Sigma, \Gamma, \delta, \Delta, q_0)$

↓ output to be given

★ Γ will contain the set of alphabets to be given as outputs.

★ δ for mealy & moore machine is equivalent to the δ of dfa.
i.e. $\delta: Q \times \Sigma \to Q$

★ Diff. b/w Mealy & Moore

Mealy :-

$\Delta: Q \times \Sigma \to \Gamma^*$

output is function of both current state & input (current input).

Moore :-

$\Delta: Q \to \Gamma^*$

O/P depends only on the current state & not on the current input.

★★ Mealy Machine ≡ Moore Machine

★ $M_1 \equiv M_2$ iff
$\forall w \in \Sigma^*$
$\Delta_1^*(q_0, w) = \Delta_2^*(q_0, w)$
means that for every input string, the two machines must give the same output.

★ M_{state}, N_{output} Mealy Machine ≡ Moore machine having no. of states $\leq (MN+1)$

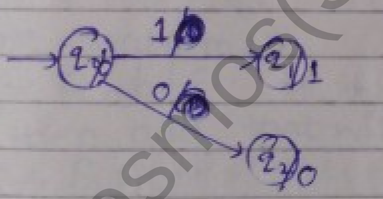
★ In worst case, every state will give both 1 & 0 as output, so if we have m states & n outputs, then there will be mn states. but if in the beginning we dont need any output, we need an extra state which is doing similar transitions to that of initial state with giving ε at the begining.
∴ total no. of max. states = mn + 1

δ	a	b	Δ o/p
q_0	q_1,0	q_2,11	1
q_1	q_2,11	q_3,10	0
q_2	q_0,1		11
q_3			10

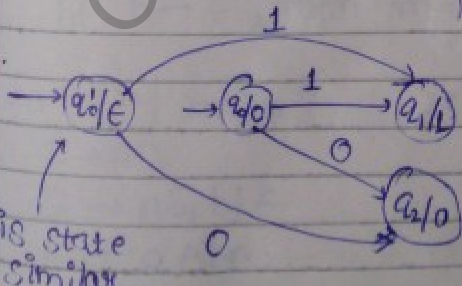
[Moore Machine ≡ Mealy Machine]

no. of states in Moore Machine ≤ m

★ there can be maximum m states in equivalent moore machine, there can be states lesser than m because in moore machine, we can have some equivalent states.



if we want no output at initial state, then we will add a state which is doing operations similar to q_0 with an addition that it wont give any output in the beginning.



this state is similar to q_0. (with the addition that it will output anything in the beginning).

☆ $\{a^m b^{2m} c^n \mid n \geq m, m,n \geq 0\}$

this is CSL, as there are 2 comparisons.

CFL's & PDA

☆ If a grammar is in chomsky normal form (CNF) with $|w| = n$, the no. of steps = $\boxed{2n-1}$.

☆ Max. height of any derivation tree = $\boxed{\lceil \log_2 n \rceil + 1}$.

☆ Any CFG can be converted to PDA only when CFG is in Grabbeth Normal Form (GNF).

Standard CFL's & their grammars & properties

1. $a^n b^n$
2. $n_a(w) = n_b(w)$ [comparison w/o ordering]
3. Pallindrome

(1) $a^n b^n$

• $\{a^n b^n, n \geq 0\}$ $\quad n_a(w) = n_b(w)$

Grammar: $S \to aSb \mid \epsilon$

• $\{a^n b^n, n \geq 2\}$

Grammar :- $S \to aSb \mid aabb$

• $\{a^m b^n, m = 2n\}$
or $\{a^{2n} b^n, n \geq 0\}$

$S \to aaSb \mid \epsilon$

• $\{a^n b^{2n}, n \geq 0\}$

$S \to aSbb \mid \epsilon$

• $\{a^n b^{2n}, n \geq 2\}$

$S \to aSbb \mid aabbbb$

—) :D :P ;)

Date:
Page No.

★ $S \rightarrow aSbb \mid \epsilon \mid a \mid b$

$S \rightarrow a^n S b^{2n}$

then we will just put the stoppers in place of S.

$\therefore S \rightarrow a^n (\epsilon + a + b) b^{2n}$

- $\{a^m b^{2m+2} \mid m \geq 0\}$

  $S \rightarrow aSbb \mid bb$

- $\{a^m b^{2m+2} \mid m \geq 1\}$

  $S \rightarrow aSbb \mid abbbb$

- $\{a^{2m+3} b^m \mid m \geq 0\}$

  $S \rightarrow aaSb \mid aaa$

- $\{a^m b^n \mid m \leq n\}$

  first create $m=n$

  $S_1 \rightarrow aS_1b \mid \epsilon$

  then add that will increase no. of b's

  $S \rightarrow S_1 B$

  $B \rightarrow bB \mid \epsilon$

  If $B \rightarrow \epsilon$, then $m=n$.

- $\{a^m b^n \mid m \geq n\}$

  $S \rightarrow AS_1$

  $S_1 \rightarrow aS_1b \mid \epsilon$

  $A \rightarrow aA \mid \epsilon$

- $\{a^m b^n \mid m < n\}$

  $S \rightarrow S_1 B$

  $S_1 \rightarrow aS_1b \mid \epsilon$

  $B \rightarrow bB \mid b$

Date:
Page No.

- $\{a^m b^n \mid m n \geq 1\}$ → regular $(a^+ b^+)$

  $\{a^m b^n \mid m > n+3\}$
  for this make the grammar for $m = n+3$ & then simply change the stopper from ϵ to aaa.

- $\{a^m b^n \mid m \neq n\}$
  create $m>n$ & $n>m$ & mix them up.

  $S \rightarrow AS_1 \mid S_1B$
  $S_1 \rightarrow aS_1b \mid \epsilon$
  $A \rightarrow aA \mid a$ → $m>n$
  $B \rightarrow bB \mid b$ → $n>m$

- $\{a^n b^n\} \cup \{a^n b^{2n}\}$
  $S \rightarrow S_1 \mid S_2$
  $S_1 \rightarrow aS_1b \mid \epsilon$
  $S_2 \rightarrow aS_2bb \mid \epsilon$

- $\{a^m b^n \mid n \leq m \leq 2n\}$ $S \rightarrow aSbb \rightarrow aaSbbb \rightarrow aa(bbb)$
  $S \rightarrow aSb \mid aSbb \mid \epsilon$
- $\{a^m b^n \mid n \leq m \leq 3n\}$
  $S \rightarrow aSb \mid aSbb \mid aSbbb \mid \epsilon$
- $\{a^m b^n \mid n \leq m \leq 4n\}$
  $S \rightarrow aSb \mid aSbb \mid aSbbb \mid aSbbbb \mid \epsilon$

(2) $n_a(w) = n_b(w)$

Imp: $S \rightarrow aSb \mid bSa \mid \epsilon \mid SS$

(a) aabb
(b) abab
(c) bbaa
(d) abba → not generated by $S \rightarrow aSb \mid bSa \mid \epsilon$

Imp.

Palindrome

① Even Palindrome

$S \rightarrow aSa \mid bSb \mid \epsilon$ → this creates ww^R.

② Odd Palindrome

$S \rightarrow aSa / bSb / a / b$

↓

this creates:-

$waw^R \cup wbw^R$

To cover all palindromes:-

$\{w \mid w = w^R\}$ (odd palindrome + even palindrome)

Date 01.04.12

If $A \overset{*}{\Rightarrow} w$ in G, then there is a leftmost derivation of w.

Date: Page No.

CFL

→ $n_a(w) = n_b(w)$ → This grammar is similar to (()), ()()() → set of properly nested structures in programming language when a represents "(" & b represents ")"

- $S \to aSb \mid bSa \mid SS \mid \epsilon$ → ($n_a(w) = n_b(w)$)
- $S \to aSb \mid SS \mid \epsilon$ → $L = \{ w \mid n_a(w) = n_b(w)$ & $n_a(v) \geq n_b(v)$ where v is the prefixes of w.

balanced paranthesis

$S \to (S) \mid)S(\mid SS \mid \epsilon$

(S) → properly balanced;)S(→ improperly balanced.

In balanced paranthesis, the no. of left brackets should be equal to or greater than no. of right brackets in all prefixes.

Palindrome:-

$S \to aSa \mid bSb \mid c$ → # palindrome

in this case c always comes in the middle, so c tells that when to start poping from stack & performs comparison for palindrome.

e.g. aab(c)baa

this tells that after c, start popping from stack.

A given grammar is linear if it is context-free grammar & it contains atmost 1 variable on the R.H.S

Grammar

$S \to abB$

$A \to aaBb \mid \epsilon$

$B \to bbAa$

first substitute A in B:- (because in S, there is derivation of B & not A)

$B \to bbaaBba \mid bba$ [by substituting $A \to \epsilon$]

$A \to aaBb \mid \epsilon$

Now, we have no A at any other part.

$\therefore B \to (bbaa)^n bba (ba)^n$

- Derivation Tree:-
(1) Root is labeled S.
(2) Every leaf has a label from $T \cup \{\lambda\}$.
(3) Every interior vertex (that is not a leaf) has a label from V.
(4) If a vertex has label $A \in V$ & its children are $a_1, a_2, \ldots, a_n$, then it must have a production $A \to a_1 a_2 \ldots a_n$.
(5) A leaf labeled λ has no siblings i.e. a vertex with a child labeled λ can have no other children.

★ A tree satisfying 3, 4, 5 & 2a. Every leaf has label $V \cup T \cup \{\lambda\}$ is a partial derivation tree.

Check this

$A \to aa\,bbA\,a\,b \mid \epsilon$

$A \to (aabb)^n A (ab)^n \mid \epsilon$

$B \to bb(aabb)^n a^n a \mid (aabb)^n a^n$

$S \to ab\, bb(aabb)^n a^n a$

$\therefore S \to abB \to ab\,(\underbrace{bb}_{P}\underbrace{aa}_{Q})^n \underbrace{bba}_{P}\,(ba)^n \quad [n \geq 0]$

Now,

$$(PQ)^n P = P(QP)^n$$

$\therefore S \to ab\,bba\,(abba)^n (ba)^n \quad [n \geq 0]$

Parsing & Ambiguity

- Parsing is done by compiler to check whether the statement belongs to language.

$w \in \Sigma^*$

$\therefore w \in L(G)$ iff $\exists$ a derivation for w using the productions of Grammar.

- The algorithm which decides whether the word belongs to language or not depending upon the productions of grammar, by derivations & the algorithm should be performed in finite amount of time & should tell whether the word belongs to language or not. This algo is called Membership Algo.

Parsing Algorithms :-

① Brute-Force Parsing (complexity :- $O(K^n)$)

In this parsing, the compiler will check whether the word belongs or not without any particular algo, i.e. it performs & try all type of derivation.

→ Ambiguity:- A terminal string $w \in L(G)$ is ambiguous if there exist two or more derivation trees for w (or there exist two or more leftmost derivations of w.)

• A CFG is ambiguous if there exists some $w \in L(G)$ which is ambiguous.

Date:
Page No.

★★ $S \to \epsilon$

though is ok, but is not allowed in compiler, because it gives a hope of contraction, e.g.

if we want to check whether "aabbbb" belongs to language or not, & by derivation we get to "aabbbbb", if null production is not there, we can surely say that aabbbb doesn't belongs to L, but if we have ϵ production, then there is a hope of contraction.

★★ ~~Presence of~~

Absence of null production assures ϵ doesn't belong.

But presence of null production says nothing.

i.e. $S \to B$ } grammar
$A \to \epsilon$

though null production, but null can't be generated by the language.

★★ In brute force, in worst case we need 2n steps to derive a word, when $|w| = n$.

IS ~~V/~~

In worst case, the sentential form will increase by 1, & in worst case, after n rounds, we can have n variables on RHS, & again after n rounds these variables are converted to n terminals.

$$|P| + |P|^2 + \ldots + |P|^{2n} = \frac{|P|\left[|P|^{2n} - 1\right]}{|P| - 1} \simeq |P|^{2n}$$

★ For every context free grammar, there is an equivalent grammar G_2 in Chomsky normal form.

Date
Page No.

Chomsky Normal Form:-

$V \to VV$
$\to T$

[i.e. no ~~normal~~ null production.]

& if there is $S \to \lambda$, then must be in L(G), we assume that S doesn't appear on the R.H.S. of any production, only after this it is in CNF.

★ Complexity:- $O(n^3)$

★ Best complexity for any algorithm that can have similar power of CFG is $O(n^3)$ itself.

★ Every LRK grammar can only generate DCFL.
LRK :- $O(n)$

~~Ambiguity~~:-

Complexity:- $O(n)$

Properties of LL(k) & LR(K) grammar:-

1. LL(k) & LR(K) grammar is unambigous.

L L (K) → no. of symbols shown to the compiler including current symbol.
L (first) → left to right parsing (means the i/p is read from left to right)
L (second) → Left most derivation [i.e. e.g. $S \to aABC$ in this case we will subsitute left most variable, i.e. $S \to aABC$, $A \to a$, $B \to b$, $C \to c$ ∴ $S \to aaBC$ & not $S \to aABc$ or $S \to aAbC$]

2. L R (K) → no. of symbols to be shown to compiler that doesn't include current symbol.
L → left to right parsing
R → Right Most derivation.

$S \to a|b \to LL(1)$
$S \to aa|ab \to LL(2)$

through K, we can precisely determine which production to use.
e.g. $S \to aa|ab$
if we show only 1 symbol to compiler, i.e. I$|a|#| in this case compiler have ambiguity whether there is 'a' or 'b' after a, i.e. it gets confused

Date:
Page No.

whether to use production S→aa or S→ab.

★ S → aSb | SS | ε

It is not LL(K)
because whatever we will give value to K, the compiler will always gets confused whether to use production S→aSb or S→SS
e.g.
if we give K=4
but if we give aabb, means it will follow production aSb,
but if we have aabbaabb, then it will have to go to SS also, so there will always remain an ambiguity.

★ Top down Compiler (Recursive Descend) → for LLK
Bottom up " (Shift reduce) → for LRK

Top down creates from root to word (LMD)
Bottom up creates from word to root (RMD).

(2) LL(K) ⇒ LL(K') | LR(K) ⇒ LR(K')
K' ≥ K
but converse is not true.
i.e. K < K' (not possible).

(3) DCFL ⇔ LR(K)
for every DCFL, we are guaranteed to have an LR(K) but it is not unique, i.e. we can have a grammar other than LR(K), but it is sure that it will have an LR(k) grammar.

{ab, aab} → prefix property ??

(2)

But every LR(K) can only create a DCFL.

(4) ~~LR(0)~~
LR(0) means that it has to show only the current symbol, as K doesn't include current symbol.
LL(1) means that it has to show one symbol which is current, as K include current symbol.

(5) LR(0) is available to those DCFL's having prefix property.
LR(1) is available to all DCFL.

Prefix property:-
In Language there is no proper prefix for any word.
e.g.
$L = \{ab, ba\}$
proper prefix of ab $= \epsilon, a$
proper " " ba $= \epsilon, b$
as, we can see that L doesn't contain ϵ, a, b, so L is LR(0).

but if $L = \{a, ab, ba\}$ is LR(1) as it doesn't follow prefix property.

** $L = \{a^n b^n, n \geq 1\}$ is LR(0)

(7) DCFL w/o prefix property ⟶ DCFL with prefix property.

Date:
Page No.

★ If L doesn't have prefix property, then L# will have.

$L\$ = \{w\$ \mid w \in L\}$

e.g. $L = \{a, ab, ba\}$ → LR(1)

$L\$ = \{a\$, ab\$, ba\$\}$ → LR(0)

prefix property.

Ambiguity

1. Grammar Ambiguity
2. Language Ambiguity

(atleast one w with 2 or more derivation tree.)

- G is ambiguous iff ∃ w ∈ L(G) with 2 or more derivation tree. (2 or more LMD/2 or more RMD)
- G is unambigous iff ∀ w ∈ L(G) has exactly 1 derivation tree. (all w should have exactly 1 derivation tree.) (exactly 1 LMD/RMD)

$S \to AB$

$A \to aA \mid \epsilon$

$B \to bB \mid \epsilon$

$S \to AB \to AbB \to aAbB \to aAb \to aab$

[this neither LMD nor RMD]

(LMD) $S \to AB \to aAB \to aaAB \to aaB \to aabB \to aab$

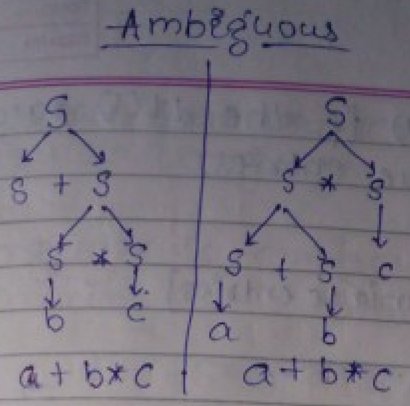
(RMD) $S \to AB \to AbB \to Ab \to aAb \to aaAb \to aab$.

★ If the word belongs to unambiguous grammar, it will surely have exactly '1' LMD.

$S \to S+S \mid S*S \mid a \mid b \mid c$ (Ambiguous)

★ For ambiguous grammar, we can have more than '1' derivation tree.

Ambiguous

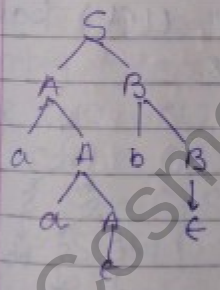


★ Having multiple derivations is not the problem, but having multiple LMD/RMD is the problem.

★ LMD is unique iff RMD is unique
iff Derivation is unique.

★ if n bit string takes x sec
then n+ bit string takes .

★ Diff. b/w Derivation & Derivation Tree :-



S → AB → aAB →

In derivation tree, we can produce so many derivations,
but only one type of word can be formed using derivation,
i.e. we can form S → AB → ab & S → AB → aB → a etc.
but we can have only one such word in derivation.

★ The grammar is LL(1) if the LL(1) parse table contains unique entries.

e.g. $S \to a$, $S \to b$ (P_1, P_2)

S	ε	a	b
	–	P_1	P_2

→ LL(1) [unique entries]

but if $S \to aa \mid ab$ (P_1, P_2)

	ε	a	b
S	–	P_1, P_2	–

↳ LL(1) parse table.

↳ grammar is not LL(1) [as it doesn't contain unique entries]

LL(2) parse table for $S \to aa \mid ab$ (P_1, P_2)

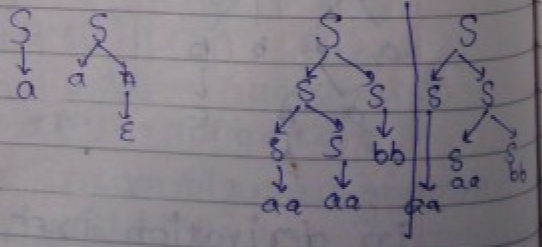
	ε	a	b	aa	ab
S	–	–	–	P_1	P_2

→ LL(2) grammar

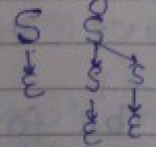
LL(2) parse table

★ The grammar is LL(K) if the LL(K) parse table contains unique entries.

★ $S \to aA \mid a \mid \epsilon$
(Ambigous grammar)

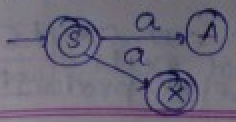


★ $S \to aa \mid bb \mid SS \mid \epsilon$
(Ambigous grammar)



★ If there is $S \to \epsilon$, then there is a chance of ambiguity.

$S \to aA \mid a \mid \epsilon$



Date:
Page No.

★★ Regular Grammar can be ambiguous but not regular Language.

★ Language is ambigous iff every grammar G producing L is ambiguous.

e.g. $S \to S+S \mid S*S \mid a \mid b \mid c$

(1) Grammar is ambiguous.
(2) Language is unambiguous.

★ Regular Language can never be ambigous.

★ L is unambiguous iff there exists atleast one grammar for it which is unambiguous.

★★ example of Ambiguous Language:-

$\{a^n b^m c^m\} \cup \{a^m b^m c^n\}$
$\{a^n b^n c^m d^m\} \cup \{a^n b^m c^m d^n\}$ } → Unremovable ambiguity.

- Ambiguous Language will have union.
- The two sets of language will not have empty intersection.
- The two sets of language can't be generated by same set of grammar.

$S \to S_1 \mid S_2$
$S_1 \to L_1$
$S_2 \to L_2$
} Two different Grammars.

- These languages are ambiguous because we can derive $a^n b^n c^n d^n$ from both the sets of languages.

★ Removal of null production can create unit & useless production, but removal of unit production can create only useless production but not null production, but removal of useless production can't ~~create either.~~

★ When Language is ambiguous we can also say that it is Inherently Ambiguous Language. (There is some problem in language itself).

Algorithms

1. Removal of ϵ production.
(have to remove it, otherwise the compiler can't run.)

If $|w| = n$, then min. height of derivation tree $\lceil \log_2 n \rceil + 1$. (Chomsky Normal form)

★ If we have CFG which creates λ-free language, it has equivalent grammar in Chomsky Normal form/Grebeth Normal form.

Removal Of Null Production

$S \rightarrow AB a C$
$A \rightarrow BC$
$B \rightarrow b \mid \epsilon$ (remove this)
$C \rightarrow D \mid \epsilon$ (remove this)
$D \rightarrow d$ (this is not reachable)

By removing all Null productions, we generate a language ~~L(G)~~ a grammar G_1, such that $L(G_1) = L(G) - \lambda$
i.e. the new language doesn't contain the null variable.

Step 1:-
1st round:- add nullable variables i.e. those that contains ϵ production.
{B, C}

2nd round:- add those variables containing only B, C
{A, B, C}

3rd round:- add those variables containing only A, B, C on R.H.S.
{A, B, C} → set of nullable variables

In elimination of Null production, we first find out all nullable variables defined by Step 1, now we will show how to calculate step 2. Acc. to step 2, $A \to \alpha$, where $\alpha \in$ {set of nullable variables}, then include this production & hence $D \to \alpha$ is selected, now as explained below in step 2 :- $S \to ABaC | BaC | AaC | ABa | aC | Aa | Ba | a$, now we will also include $A \to BC$, because if we want $S \stackrel{*}{\Rightarrow} bdbad$, ~~then it can only be done by~~ $S \to ABaC \to BCBaC \to bDBaD \to bdbad$ ∴ include $A \to BC$, now we can't do any derivation w/o $B \to b$, $C \to D$,

Step 2 :- $S \to ABaC$
put A to null, B to null & C to null,
i.e. one at a time,
then 2 at a time
& then all A, B, C to null at the same time

$S \to ABaC | BaC | AaC | ABa | aC | Aa | Ba | a$
$A \to BC | B | C$
$B \to b$
$C \to D$
$D \to d$

∴ ~~$S \to ABaC | BaC$~~
$S \to BaC | ABaC | ABa | aC | Aa | Ba | a$

BaC, AaC, ABa: Removing 1 variable at a time.
aC, Aa, Ba: Removing 2 variables at a time.
a: Removing all 3 variables at a time.

Removal Of Unit Production

$S \to Aa | B$
$B \to A | bb$
$A \to a | bc | B$

$S \to AB$
$A \to a$
$B \to C | b$
$C \to D$
$D \to E$
$E \to a$

$S \to AB$ is not a unit production.
The unit productions are $B \to C$, $C \to D$, $D \to E$

$B \to C \to D \to E$

∴ $S \to AB$
$A \to a$
$B \to a | b$
but we can see that
$B \stackrel{*}{\Rightarrow} a$ $D \stackrel{*}{\Rightarrow} a$
$C \stackrel{*}{\Rightarrow} a$ $E \to a$
∴ add these
$C \to a$, $D \to a$, $E \to a$

[Draw the unit graph]

1. $S \stackrel{1}{\Rightarrow} B$
2. $S \Rightarrow A$
3. $B \Rightarrow A$
4. $A \Rightarrow B$

but A can't go to S, & neither B can go to S.

∴ remove unit production
$S \to Aa$
$B \to bb$
$A \to a | bc$
we have 1. as $S \stackrel{*}{\Rightarrow} B$, so add it
$S \to Aa | bb$
& we also have 2. as $S \stackrel{*}{\Rightarrow} A$, so add all RHS to S.
∴ $S \to Aa | bb | a | bc$

we have 3. as $B \overset{*}{\Rightarrow} A$, ∴ add all RHS of A productions to B.

- ∴ $B \rightarrow bb | a | bc$

we have 4 as $A \overset{*}{\Rightarrow} B$, ∴ add all RHS of B (in grammar) productions to A.

- $A \rightarrow a | bc | bb$

Removal Of Useless Production

Useful Variable: $U = \{V | S \overset{*}{\Rightarrow} xVy \overset{*}{\Rightarrow} w\}$ This means V is participating in some derivation.

Useful variable is that which participates in atleast one derivation of the grammar.

Useless Variable:- (1)

$S \rightarrow aS | A | C$
$A \rightarrow a$
$B \rightarrow aa$
$C \rightarrow aCb$

Step 1:- Useless variable which doesn't create any terminal.

1st round:- add those variables which creates terminal.
∴ {B, A}

2nd round:- In this round check whether any variable goes to either variable A or variable B or both.
then add it.
We see that $S \rightarrow A$
∴ {B, A, S}

Now, we have C as useless variable.

★ Now, remove anything that have C on L.H.S. or R.H.S.

$S \to Sa \mid a$
$S' \to aS' \mid \epsilon$
$S \to aS'$

$S \to aS \mid A$
$A \to a$
$B \to aa$

* Now, remove those variables which are not reachable by S. [B is not reachable by S.]

$\therefore S \to aS \mid A$
$A \to a$

↑ It may contain ε.

Removal Of Left Recursion

(Main culprit for Ambiguity)

$A \to A\alpha_1 \mid A\alpha_2 \mid A\alpha_3 \ldots \mid A\alpha_m \mid \beta_1 \mid \beta_2 \ldots \mid \beta_n$

Step 1: Put all A's on the right & remove all the stoppers

$A' \to \alpha_1 A' \mid \alpha_2 A' \mid \alpha_3 A' \mid \ldots \mid \alpha_m A'$

Now put ε as the stopper.

$A' \to \alpha_1 A' \mid \alpha_2 A' \mid \alpha_3 A' \mid \ldots \mid \alpha_m A' \mid \epsilon$

Now attach all β's on right hand side of A.

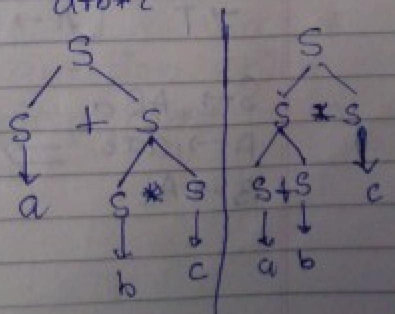
$\therefore A \to \beta_1 A' \mid \beta_2 A' \mid \ldots \mid \beta_n A'$

e.g.

$S \to S{+}S \mid S{*}S \mid a \mid b \mid c$

(where $+S = \alpha_1$, $*S = \alpha_2$, $a = \beta_1$, $b = \beta_2$, $c = \beta_3$)

a+b*c



$S' \to +SS' \mid *SS' \mid \epsilon$
$S \to aS' \mid bS' \mid cS'$

??

Date:

Page No.

Left Factoring (can cause ambiguity)

$S \to \alpha X_1 \mid \alpha X_2 \mid \alpha X_3$

when we have same ~~term~~ terminal on two or more productions of S.

Removal Of Left Factoring-

Grammar: $S \to \alpha X_1 \mid \alpha X_2 \mid \alpha X_3$ [X_1, X_2, X_3 should have nothing in common.
i.e. $X_1 \cap X_2 = \phi$
$X_1 \cap X_3 = \phi$
$X_2 \cap X_3 = \phi$]

Equivalent Grammar: $S \to \alpha S'$
$S' \to X_1 \mid X_2 \mid X_3$

~~i.e.~~

$S \to \alpha\phi 1 \mid \alpha\phi 2$

to do it,
take α as $\alpha\phi$

?? → $\therefore S \to \alpha\phi S'$
$S' \to 1 \mid 2$

Removal to CNF:-

$V \to VV$
$\to T$

[$A \to BC$
$A \to a$]

$S \to ABa$
$A \to aab$ ⟹
$B \to Ac$

$S \to [AB] B_a$
$A \to B_a B_a B_b$
$B \to A B_c$
$B_a \to a$
$B_b \to b$
$B_c \to c$

(as there can only be 2 variables on RHS.) Take AB as single variable & make B_a, B_b, B_c as variables.

★ To use CNF & GNF, first ~~perform~~ ~~null~~ remove null & unit production.

Date:
Page No.

Removal to GNF:-

$S \to AB$ → not in GNF
$A \to aA \mid bB \mid b$
$B \to b$ → GNF

$V \to TV^*$
$\to T$ → (single Terminal)

→ One Terminal followed by any no. of variables.

Step 1:- substitute the productions of ~~those~~ ~~vari~~ 1st variable to the production that is not in GNF.

∴ $S \to aAB \mid bBB \mid bB$
$A \to aA \mid bB \mid b$
$B \to b$

if we have $S \to ABa$
do it as $S \to aABB_a \mid bBBB_a \mid bBB_a$
$A \to aA \mid bB \mid b$
$B \to b$
$B_a \to b$

★ We wont be given any question which contains 1st variable which wont be in GNF.

Date:
Page No.

$|w| = n$

no. of steps in chomsky Normal form = $2n-1$

$A \rightarrow BC$

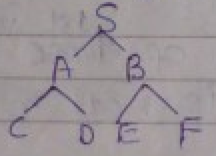
$A \rightarrow a$

$S \xrightarrow{1} AB \xrightarrow{2} BCB \xrightarrow{3} CDCB$

$(n-1)$ steps is req. to make S to be all n variables

& then n ~~variables~~ steps are required to make all variables as terminals.

Min. height in CNF : $\lceil \log_2 n \rceil + 1$



- $\lceil \log_2 n \rceil$ → for obtaining all the variables i.e. e.g. $S \rightarrow ABCD$
- 1 → 1 step for converting all variables to terminals

PDA programming

(Q, Σ, Γ, δ, q_0, Z, F)

- Γ → symbols to be put into stack.
- Z → starting stack symbol [i.e. initially the top of stack will contain Z.]

① A pda halts when the stack is empty i.e. it halts when $\delta(q, \lambda, \lambda)$; when top of stack is empty.

② $\delta(q, \lambda, z) \rightarrow$ null move, i.e. without reading the input the transition takes place.

★ null move is allowed both in dpda & npda.

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

1. Push → $\delta(q_0, a, b) = (q_1, ab)$:- replace b with ab, with a on top.
2. Pop
3. Replace
4. Do nothing

Pop → $\delta(q_0, a, b) = (q_1, \epsilon)$

☆ To make pda as nfa, then put z on the stack always.

Date:
Page No.

NPDA:-

$\delta\ Q \times \{\Sigma \cup \epsilon\} \times \Gamma \rightarrow 2^{Q \times \Gamma^*}$ (Finite subsets of $2^{Q \times \Gamma^*}$)

e.g.

$\delta(q_0, a, b) = \{(q_1, ab), (q_2, \epsilon)\}$

These two will have their individual stacks.

i.e. in npda, we must have finite no. of choices.

Diff. b/w dpda & npda

dpda	npda
① No choice	① Choice
② Conditional null move.	② Unconditional null move.

* If we allow unconditional null move, then it will cause choice in dpda.

$\delta(q, \epsilon, b) = (q_1, b)$
$\delta(q, c, b) = (q_2, b)$

If we allowed null move, then for same combination of previous state & top of the stack, we can't specify a δ with input character other than ϵ.

$\delta(q, \epsilon, b) \neq \phi$ (this means if there is a null move.)

↓ implies

$\delta(q, c, b) = \phi$
$\forall c \in \Sigma$

* In DPDA & NPDA, if there is dead configuration then string is rejected.

??

Date:
Page No.

(1) $\{a^n b^n\}$

☆ To accept null move, we have to make 1st state as final state.

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

| a | a | a | b | b | b | ϵ |

q_0

$\delta(q_0, a, Z) = (q_1, aZ)$

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

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

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

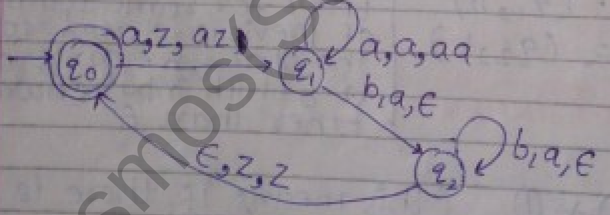
$\delta(q_2, \epsilon, Z) = (q_0, Z)$

★ We changed the state from q_0 to q_1 because at q_0 we accepted ϵ, but if we remained at q_0, a will also be accepted.

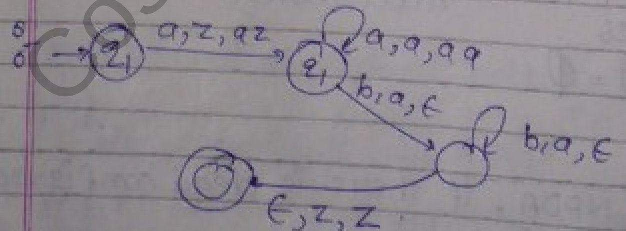
★ We changed the state from q_1 to q_2, because we are accepting a only at q_1 & accepting b only at q_2.

As we have no. of a's = no. of b's, ∴ we moved to the final state q_0.

Before this command when the last 'b' is input, then the last 'a' will be popped & hence next time, input will read ϵ & top of stack will be Z in the



$n \geq 1$



★ Any CFL can be made using machine with maximum of 3 states.

Date:
Page No.

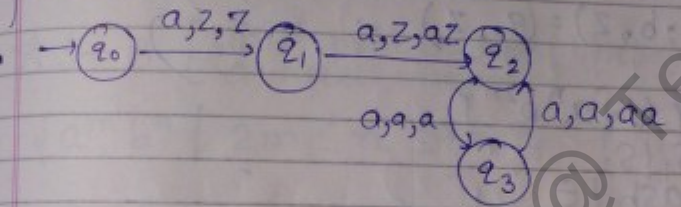
- $a^n b^{2n}$

$\delta(q_0, a, Z) = (q_1, aaZ)$
$\delta(q_1, a, a) = (q_1, aaa)$
$\delta(q_1, b, a) = (q_2, \epsilon)$
$\delta(q_2, b, a) = (q_2, \epsilon)$
$\delta(q_2, \epsilon, Z) = (q_0, Z)$

aabbbb

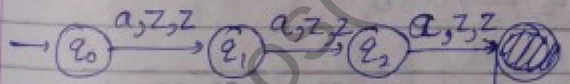
we push 4 a's inside, because when we pop, we can only pop 1 a at time with 1 input symbol, so there are 4 b's, so we need 4 a's in stack.

- $a^{2n} b^n$



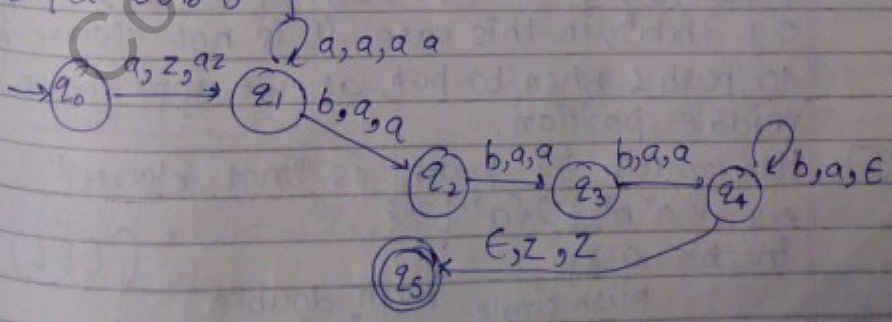
[Accepting a's at even positions.]

- $\{a^m b^n \mid m = n+3\}$
$= \{\underline{aaa}\, a^n b^n\}$
↳ regular (for any regular expression dont use the stack.)



↳ similar to $a^n b^n$

- $\{a^m b^n \mid n = m+3\}$
$= \{a^m bbb\, b^m\}$



Date:
Page No.

- $\{a^m b^n \mid m \geq n\}$ — aaaabbϵ ↑

Sol

The last command will be :-

$\delta(q_2, \epsilon, a) = (q_0, a)$ → we have reached end of string, i.e. last b.

- $\{a^m b^n \mid n \geq m\}$

The last command will be :-

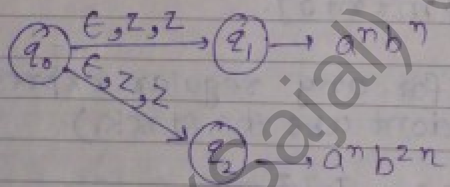
$\delta(q_2, b, Z) = (q_0, Z)$

- $\{a^n b^n\} \cup \{a^n b^{2n}\}$

$S \rightarrow S_1 \mid S_2$

$S_1 \rightarrow aSb \mid \epsilon$

$S_2 \rightarrow aSbb \mid \epsilon$



Check ~~wheth~~ whether CFL is DCFL :-

① push → pop "not clear"
push to pop is not clear. (CFL)
e.g. ww^R, in this case it is not clear when to push & when to pop, as we don't have middle position.

② How much to push is "not clear."
e.g. $\{a^n b^n\} \cup \{a^n b^{2n}\}$
~~In this cas~~ → push single "a" ; → push double "a" (CFL)

as DCFL has single stack, ∴ such language is CFL.

→ CFL (not DCFL)

3. $\{a^m b^n c^p \mid m=n \text{ or } n=p\}$
we need one comparison.
but
$\{a^m b^n c^p \mid m=n \ \& \ n=p\} \rightarrow$ CSL.

* $\{a^n b^n\} \cup \{a^n b^{2n}\}$

$\delta(q_0, \epsilon, z) = \{(q_1, z), (q_2, z)\}$

☆ $S \rightarrow A$ [unit production always means a null move]

· $\{a^m b^n \mid 2m \le n \le 3m\}$ (CFL)

$\delta(q_0, a, z) = \{(q_1, aaZ), (q_2, aaaZ)\}$

push 2a for each b — push 3a for each b.

$\delta(q_1, b, a) = \delta(q_1, \epsilon)$
$\delta(q_1, \epsilon, z) = \delta(q_F, z)$
$\delta(q_2, b, a) = \delta(q_2, \epsilon)$
$\delta(q_2, \epsilon, z) = \delta(q_F, z)$

· $N_a(w) = N_b(w)$ → CFL

{for every a, if b comes then pop that a, & if a comes push it into stack & similar with b.}

| b | a | a | b |

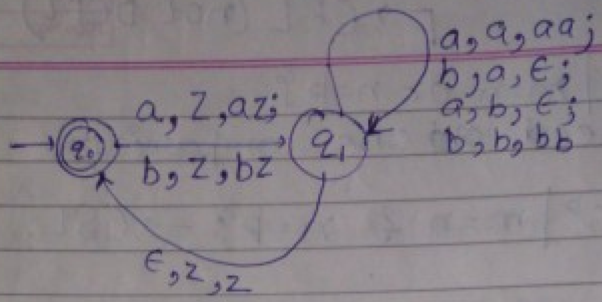
$\delta(q_0, a, z) = (q_1, az)$
$\delta(q_0, b, z) = (q_1, bz)$

$\delta(q_1, a, a) = (q_1, aa)$
$\delta(q_1, b, a) = (q_1, \epsilon)$
$\delta(q_1, a, b) = (q_1, \epsilon)$
$\delta(q_1, b, b) = (q_1, bb)$

$\delta(q_1, \epsilon, z) = (q_0, z)$ → final state.

Date:

Page No.



Palindrome (CFL)

- WW^R

| a | a | b | b | a | a |

$\delta(q_0, a, Z) = (q_1, aZ)$
$\delta(q_0, b, Z) = (q_1, bZ)$
$\delta(q_1, a, a) = (q_1, aa)$
$\delta(q_1, b, a) = (q_1, ba)$
$\delta(q_1, a, b) = (q_1, ab)$
$\delta(q_1, b, b) = (q_1, bb)$
$\delta(q_1, \epsilon, a) = (q_2, a)$ } → jump to
$\delta(q_1, \epsilon, b) = (q_2, b)$ } pop state
$\delta(q_2, a, a) = (q_2, \epsilon)$ } → pop
$\delta(q_2, b, b) = (q_2, \epsilon)$ } state
~~$\delta(q_2, a, b)$~~ =
$\delta(q_2, \epsilon, Z) = (q_0, Z)$ → acceptance

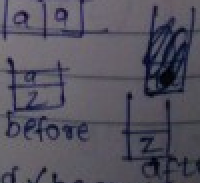
- | a | a | b | b | a | a |
  ↑ q_0

1st copy

- | a | a | b | b | a | a |
  ↑ q_1

2nd copy

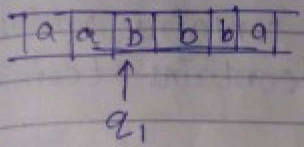
| a | a | b | b | a | a |
↑ q_2

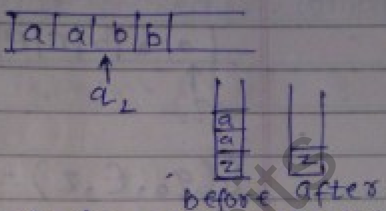


This copy is rejected (because of dead configuration)

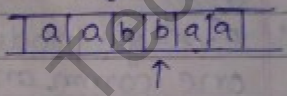
★ The two copies of stack can't communicate because if they does it, the machine will become turing machine.

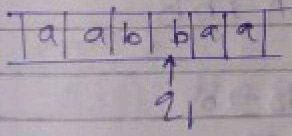
Dead configuration because we havent specified anything like $\delta(q_2, b, z)$

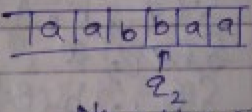
- 



this is also rejected because of dead configuration.



- ~~Odd Palin~~ |
- 



Now the 2nd copy is accepted.

- Odd palindrome $(w(a+b)w^R)$

$\delta(q_2, a, a) = (q_3, a)$
$\delta(q_2, a, b) = (q_3, b)$
$\delta(q_2, b, a) = (q_3, a)$ *
$\delta(q_2, b, b) = (q_3, b)$

Rest ~~is sim~~ commands is similar to even palindrome.

GNF → PDA

$S \to aA$
$A \to aABC \mid bB \mid a$
$B \to b$
$C \to c$

a	a	b	c	ε

↑ q_0

Stack contains variable & input contains terminals.

$\delta(q_0, \epsilon, z) = (q_1, Sz)$ ∴ put the start variable in the stack.

$\delta(q_1, \epsilon, z) = (q_f, z)$

- Every production is converted into one command each.

$\delta(q_1, a, S) = (q_1, A)$ $S \to aA$
$\delta(q_1, a, A) = \{(q_1, ABC), (q_1, \epsilon)\}$ $A \to aABC$ & $A \to a$
$\delta(q_1, b, A) = (q_1, B)$ $A \to bB$
$\delta(q_1, b, B) = (q_1, \epsilon)$ $B \to b$
$\delta(q_1, c, C) = (q_1, \epsilon)$ $C \to c$

Closure Properties

- CFL is closed under $\cup, \cdot, *$ & not in $\cap, L^c$.
- DCFL is closed under L^c & not in $\cup, \cdot, *, \cap$.
- $\{a^n b^n c^m\} \cap \{a^n b^m c^m\} = \{a^n b^n c^n\}$
  - $\{a^n b^n c^m\}$ → CFL
  - $\{a^n b^m c^m\}$ → CFL
  - $\{a^n b^n c^n\}$ → CSL (not CFL)

Intersection of two CFL's may or may not be CFL.

$(L_1^c \cup L_2^c)^c = (L_1 \cap L_2)$

let L_1^c & L_2^c be CFL, & we know that $\cup$ is closed,
let complement be closed, so LHS is closed,
but we know that $\cap$ is not closed, ∴ complement is also not closed.

Test for Emptiness

$L(G) = \phi$ iff $S \notin$ Useful variables
[In algo. of Removal of Useless Variable]

Variations Of CFL

- $\{a^m b^n c^p \mid m = n, p \geq 0\}$ → CFL
  as p doesn't depend upon m & n.
- $\{a^m b^m c^n d^n \mid m, n \geq 0\}$ → CFL
  this is CFL because we search for $a^m b^m$ & then stack will be empty & then we will check for $c^n d^n$.
- $\{a^m b^n c^p \mid m \leq n$ or $n > p\}$ → CFL
- $\{a^m b^n c^p \mid p = m+n\}$ → CFL
  ↓
  one comparison

as we will push '1' whenever a & b comes & will pop '1's whenever c comes.

- $\{a^m b^{m+n} c^n \mid m,n \geq 0\} \rightarrow CFL$

or

$\{a^m b^p c^n \mid p-m=n\} \rightarrow CFL$

aaabbbbbcc

| a |
| a |
| a |

& for next 3 b's we will pop a

| b |
| b |

& for next 2 b's we will push b

& for next 2 c's we will pop b.

- $\{a^m b^m c^n d^n\} \rightarrow CFL$

$\{a^m b^n c^n d^m\} \rightarrow CFL$

for this push all a's, then push all b's & then pop every b's for every c's, & then stack will contain only a's, & then pop every a for every d.

- $\{a^m b^n c^m d^n\} \rightarrow CSL$
- $\{a^m b^n c^p d^q \mid m+n = p+q\} \rightarrow CFL$

push one 1 for every a & b, & then pop one 1 for every c & d.

- $\{a^m b^n c^p d^q \mid m+q = n+p\} \rightarrow CFL$

or

$m-n+q = p$

push 0

- $\{a^m b^n c^p d^q \mid m+p = n+q\} \rightarrow CFL$

Date:
Page No.

- $\{a^m b^m c^n d^p \mid m > p\} \rightarrow CSL$

Turing Machine

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

Γ → tape alphabet

★ There is no stack in Turing Machine.

★ $\square$ is used to tell the starting & ending of string [it is a reserved symbol.]

$\boxed{\Sigma \subseteq \Gamma - \{\square\}}$

★ It has same tape for both i/p & o/p.

Standard Turing Machine

★ The input tape will contain all blanks except where the input symbols are.

□	□	a	a	b	b	□	□	□

(before) — head at q_0 under the first a

& not space ← because we need to program for □ input as well.

$\boxed{\delta: Q \times \Gamma \longrightarrow Q \times \Gamma \times \{L/R\}}$

$\delta(q_0, a) = (q_1, b, L)$ $\delta(q_1, \square) = (q_2, a, R)$

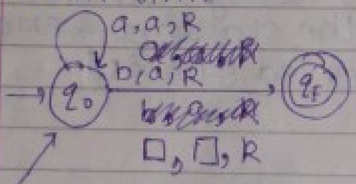
□	□	b	a	b	b	□	□	□

(after) — head at q_1; b → this is changed.

↑ String will be accepted when it becomes dead in final state.

☆ Turing machine can only accept by coming to the final state.
Once machine comes to the final state, it will halt.
Rejection:- (i) dead on non-final state.
(ii) hang.

☆ Turing machine can accept even when it hasn't parsed the complete string, because the string go on changing with time.

☆☆ 

[No left move:- Machine will never hang.]

$(a+b)^+$ [ϵ is never accepted by Turing Machine]

$$L(M) = \{ w \in \Sigma^+ \mid q_0 w \vdash^* x q_f y, q_f \in F \}$$

Configuration:-

□	a	a	b	b	□

↑ q_0

$\delta(q_0, a) = (q_1, b, R)$
$\delta(q_1, a) = (q_2, a, R)$

What is the configuration after 2 moves?

→ Configuration in the begining

$q_0 aabb \vdash b q_1 abb \vdash ba q_2 bb$

In configuration :-

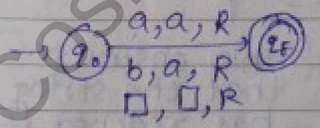
1. 1st part contains the string that is processed.
2. 2nd part contains the current state.
3. 3rd part contains the string to be processed.

→ The given machine will never hang as there is no left move.
∴ It will halt for $(a+b)^+$.
This halt will contain those string which will be accepted on final state & rejected on non-final state.
But, this machine is accepting everything.

→ Hang,
this machine is not hanging,
∴ it will hang on φ.

→ Contents of tape (in the end)
in this machine, we can see that whatever the input, only a will be overwrited, ∴ content of tape in the end will contain string of a's.

**



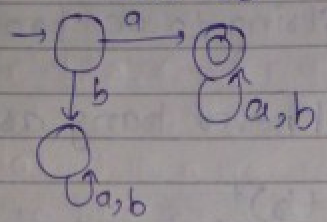
Contents of the tape :- $a(a+b)^*$ [as after getting 1st a or b, it will be overwritten to a & will reach the final state & the string will be accepted, & the rest of the string will not be processed.]

★ The machine will halt the moment it reaches a final state & the rest of the string will not be processed.

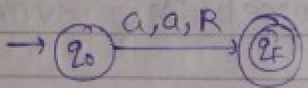
Date:
Page No.

Converting DFA to Turing Machine :-

- a(a+b)*

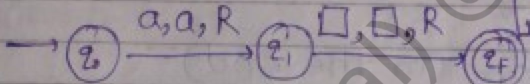


remove the trap state & remove the self loop on permanently accepting state.

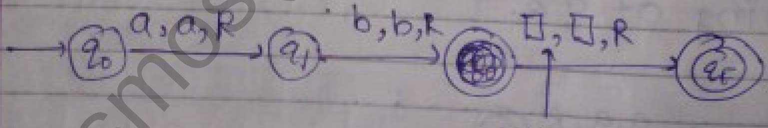


- starting with b will be rejected due to dead configuration
- & once reaching the final state, the machine will halt & whatever comes after, it will not be processed & everything will be accepted.

- To accept just 'a'

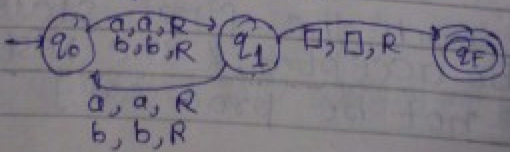


- To accept ab :-

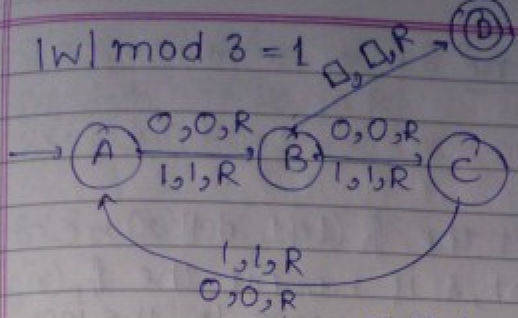


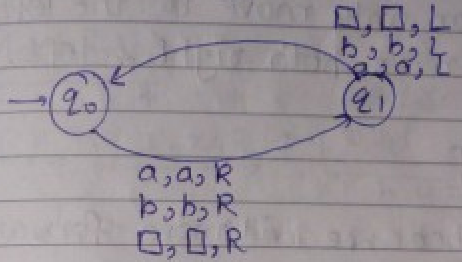
this ensures that the string has reached the end.

- Odd length string :-



- $|w| \bmod 3 = 1$

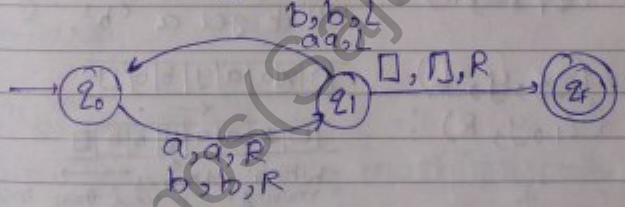


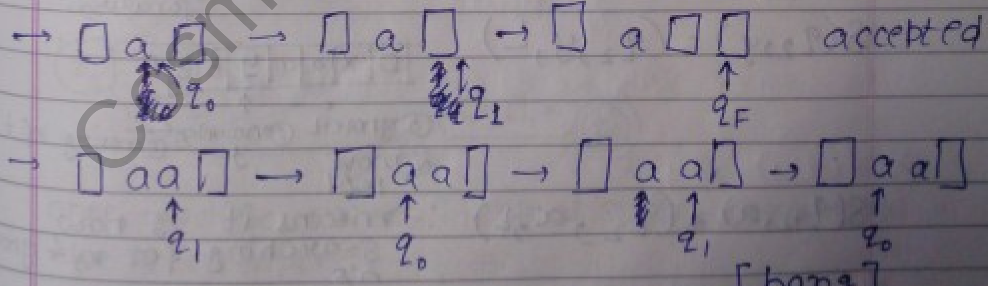
- 

This will accept nothing (φ).

This machine will hang on everything.

□a□
□b□
□aa□
□ab□
} → will hang on everything.

- 



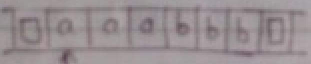
Date

07.04.12

Turing Machine

LBA

in LBA, if we hit the $\$_L$ (left dollar) we can only move to right, & if we hit the $\$_R$ (right dollar), then we can only move to the left, & in b/w we can move both right & left.



$a^n b^n$:- if we are at 1, then we will move forward till we get a b, then we move left to find a, & then move right to find its corresponding b, & then we will remove these a's & b's.

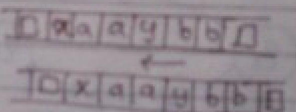
$\delta(q_0, a) = (q_1, x, R)$

$\delta(q_1, a) = (q_1, a, R)$:- means it is moving right till it get a 'b'.

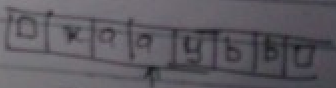
$\delta(q_1, b) = (q_2, y, L)$

$\delta(q_1, y) = (q_1, y, R)$:-

$\delta(q_2, y) = (q_2, y, L)$



$\delta(q_2, a) = (q_2, a, L)$:- means it is now searching for x, & ignoring a's

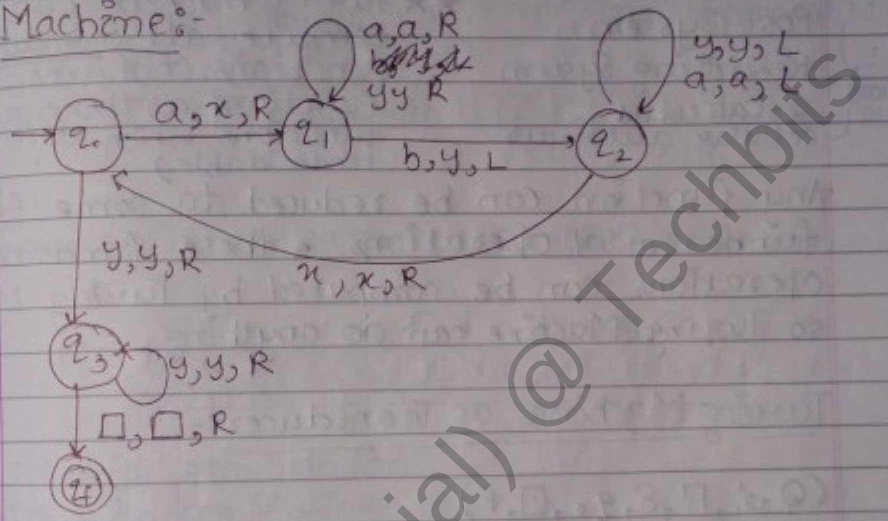


Date:
Page No.

$\delta(q_2, x) = (q_0, x, R)$
$\delta(q_0, y) = (q_3, y, R)$
$\delta(q_3, y) = (q_3, y, R)$

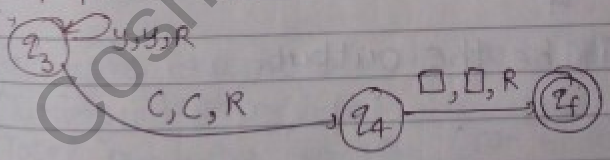
$\delta(q_3, \square) = (q_f, \square, R)$ → reached final state, machine becomes dead.

Machine:-



similarly:- $a^n b^n c$

$a^n b^n c(a+b+c)^*$
Similar to above machine, only diff. is remove q_f from above



once, machine comes in final state, everything else is accepted.

** Any NTM can be converted into DTM.

★ DTM has ~~similar~~ infinite tape on both sides.

→ PDA model :- compiler (used in compiler)
→ FA model :- Lexical Analysis
→ Turing Machine :- To check whether something can be done or not.

TM as transducer ≡ { Post System, Rewriting System, λ-calculus, cellular automata }

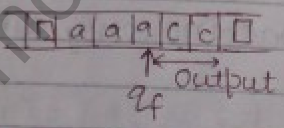
★★ Turing machine can compute all computable functions. (not all functions as there are functions which are undecidable)

Any function can be reduced to some finite fundamental operations & these fundamental operations can be computed by Turing Machine so Turing Machine can do anything.

Turing Machine as Transducer :-

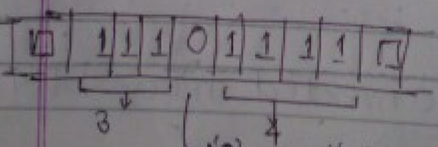
$(Q, \Sigma, \Gamma, \delta, q_0, \square, F)$

$q_0 w \vdash q_f f(w)$



∴ acc will be the output.

★ $f(x,y) = x+y$
e.g. 3+4



↳ '0' will ~~tell~~ discriminate about the inputs.

replace '0' with '1' & then make leftmost or rightmost '1' with a '□'.

$\delta(q_0, 1) = (q_0, 1, R)$
$\delta(q_0, 0) = (q_1, 1, R)$
$\delta(q_1, 1) = (q_1, 1, R)$
$\delta(q_1, \square) = (q_2, \square, L)$
$\delta(q_2, 1) = (q_3, \square, L)$:- making rightmost 1 with a '□'
$\delta(q_3, 1) = (q_3, 1, L)$
$\delta(q_3, \square) = (q_f, \square, R)$:- the pointer is at the 1st position & is at final state.

** Semidecidable + Not even semidecidable = undecidable

Variations Of TM.

1. TM with stay option
2. TM with semi-infinite tape
3. Multi-tape turing machine
4. Offline TM
5. Multidimensional TM
6. NTM
7. Universal TM

} ≡ Standard TM

Recursive Enumerable ≡ partial recursive function (Turing Machine)
Total Recursive Function (can be done by HTM).

Turing Machine ≡ Any computational model

$\sim q \vee p$ $q \rightarrow p$ → converse
$\sim p \vee \sim q$ $\sim p \rightarrow \sim q$ → inverse

Date:
Page No.

① Turing Machine with stay option

Stay means do not move left or right & stay there.

$$\delta\, Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R, S\}$$ (S = stay)

It is similar to standard TM, because stay can be performed by first moving to right & then to left.

e.g. $\delta(q_0, a) = \{q_1, b, S\}$

$\parallel$

$\delta(q_0, a) = (q_1, b, R)$
$\delta(q_1, a) = (q_0, a, L)$ } → do this for all symbols, & as
$\delta(q_1, b) = (q_0, b, L)$

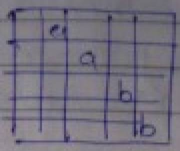
② TM with Semi-infinite Tape

An infinite tape on both sides can be folded & we can have semi-infinite tape.

④ Offline TM.-
It will have separate tapes for input & output.

⑤ Multidimensional TM -

It will have grid type tape :-



This multidimensional can be converted to linear tape.

⑥ NTM

There is an algo for converting NTM to DTM.

$\delta_{Q \times \Gamma} \rightarrow 2^{Q \times \Gamma \times \{L,R\}}$.

⑦ UTM

** Turing Machine is not reprogrammable, so it can't ~~run~~ perform multitasking.

We can give command set of any turing machine to UTM, & it can simulate it.

** UTM is a 3-state machine. (it has 3 i/o tapes) it ~~has~~ is a multitape TM.

→ In 1st tape, we give the code or command set of Turing Machine.
In other words, we give binary encoding of the command set of turing machine.

e.g. $\delta(q_0, a) = (q_1, b, R)$
$\delta(q_1, b) = (q_2, b, L)$

$Q = \{q_0, q_1, q_2\}$, ~~[illegible]~~ $\{R, L\}$, $\Gamma = \{a, b, \square\}$

$q_0 \to 1$, $q_1 \to 11$, $q_2 \to 111$; $R \to 1$, $L \to 11$; $a \to 1$, $b \to 11$, $\square \to 111$

$\delta(q_0, a) = (q_1, b, R)$

this is converted as:-

1 0 1 0 1 1 0 1 1 0 1
(↓ q_0, 0 → , , ↓ a, 0 → , , ↓ q_1, 0 → , , ↓ b, 0 → , , ↓ R)

& $\delta(q_1, b) = (q_2, b, L)$

(11) 0 (11) 0 (111) 0 (11) 0 (11)
↓ q_1 ↓ b ↓ q_2 ↓ b ↓ L

∴ tape 1 will contain

10101101101 0110110111011011

(2) On tape 2, we give the input which is given to the turing machine. (Tape 2 will contain the current input).

(3) The third tape will tell the position of read/write head.

e.g.

if the current state is q_2,

the tape will contain

| □ | 1 | 1 | 1 | □ |

So, UTM is a Multitape TM, & every Multitape TM is a Standard TM, ∴ UTM = STM.

★★ UTM can also simulate a UTM,

Less Power than STM

1. HTM ⟶ Decidable.
This machine will never hang, as hanging is a necessary evil
A problem is decidable iff it can run on a HTM.
So, there exists an algo. for a problem iff it can run on a HTM

2. LBA < ~~STM~~ HTM < STM (powerwise).
↳(CSL) (REC) ↳(RE)

3. Turing Machine with read only head ≡ 2-dfa. ≡ nfa ≡ dfa. (with or without Right, left move capability).

★ Turing Machine with read/write capacity but without Right, Left moving capability < STM

★ FA + 2 counter ≡ Turing Machine
FA < FA + 1 Counter < DPDA
(Regular) (DCFL)

★★ I $\{a^m c b^m\} \cup \{a^m d b^{2m}\}$ ⟶ DCFL {as we will know how much to pop}

II $\{a^m b^m\} \cup \{a^m b^{2m}\}$ ⟶ CFL. (as we dont know how much to push).

III $\{a^m b^m c\} \cup \{a^m b^{2m} d\}$ → CFL [clue is coming in the end, so its of no use.]

IV $\{c a^m b^m\} \cup \{d a^m b^{2m}\}$ → DCFL
if c comes, go to some state q' & push one 'a', & if d comes, go to some other state q'' & push two 'a's'.

In case I, we push one 'a', & then if c comes we will pop one 'a' for every 'b' & if d comes we will pop one 'a' for two 'b'.
∴ DCFL.

In case IV, we push one 'a' if 1st symbol is c & we push two 'a' if 1st symbol is d.

V. $\{a^m c b^m\} \cup \{a^m d b^m\} \rightarrow$ DCFL

~~In this case, we push one 'a' & then if c comes, we pop one 'a' for every 'b' & if d comes for every 1 'b', pop 'a'.~~

For every 2 'a' push 'a' & if c comes then pop 2 'a' for every b & if d comes pop

REC & RE

$\{a^n b^n\} \cup \{aa^*\} \rightarrow$ CFL [as there is a choice
$\{a^n b^n\} \cup \{cc^*\} \rightarrow$ DCFL

★★ A language L is RE iff there exists a TM which accepts L. (It will halt only when string belongs to language.) [iff it is turing enumerable]

Rejection in TM < looping / halting in non-final state

★★ A language L is REC iff there exists a [iff it is turing enumerable]

TM which accepts L & which halts on every $w \in \Sigma^*$.
This will halt on every string that may or may not belong to language.

* REC & RE both have enumeration procedure (Turing Enumerable) & Only REC have Membership Algorithm.

* A procedure is EP for language L iff ~~there exists~~ this procedure must list every member of L in finite amount of time.

- Uncountable Infinity:- b/w any two nos., there are infinite nos.
e.g. Set of R, i.e. there are infinite nos. b/w 0.1 & 0.2.
- Countabe Infinity:- b/w two nos., there are no nos.
e.g. set of Z, eg. b/w 1 & 2.

- By countable, we mean that any member of a set can be reached by some rule in a finite amount of time.

N, Z, Q → countable infinite | IR, R (real nos.), C → uncountable infinite.

* 2^Z is uncountable infinite.

Cander's Diagonalisation Theorem:-

If S is countably infinite, then 2^S is uncountably infinite.

??

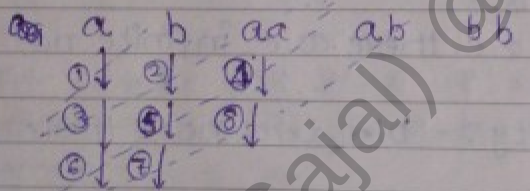
Date:
Page No.

★★ $L \subseteq \Sigma^*$, then L is countable.
every language is countable, this means all language has EP.

★ Not RE was also has an EP.
but Not RE is not turing enumerable.

a, b, aa, ab, bb, ba

if we take one string at a time & checks if it is a member, then if aa is not a member, then machine will get hang.
& other members wont get listed in finite amount of time, rather they are never listed.



using this step, even though machine gets hanged with aa but all the members are listed in finite amount of time.

★ Proper Ordering

To prove Σ^* is EP. → ??
Σ^* is CI,
& every set is a subset of Σ^*, ∴ those are also countable.

Theorems:-

① S is countable iff ∃ an EP for S.
② Every subset of countable set is also countable

Date:
Page No.

(3) Countable union of countable sets is countable.
i.e.
$L_1 \cup L_2 \cup L_3 \cup \ldots \ldots \rightarrow$ infinite sets.
→ we mean that there should not be any set b/w L_1 & L_2.

(4) Infinite Cross product of countable sets is countable.
e.g.
Z×Z×Z → countable
R×R×R → uncountable
because R is uncountable because there are infinite nos. b/w 0.1 & 0.2.

(5) If S is countably infinite then 2^S is uncountably infinite.

Diff. b/w EP & MA:-
(1) MA not only lists all the members but also lists all the non-members.

★★ L is REC iff both L & $\bar{L}$ has EP.
(L → members, $\bar{L}$ → non-members.)

★★ L is REC iff it can be enumerated in Lexicographic order.
★★ REC can be enumerated in Lexicographic order because it will never hang & will always halt to tell whether it is a member or not of the language.

	REC	RE
$\cup$	✓	✓
$\cap$	✓	✓
L^c	✓	x
$\cdot$	✓	✓
$*$	✓	✓

→ L is REC iff $\bar{L}$ is REC.
but if L is RE then L^c may or may not be RE.

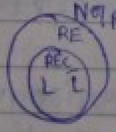
(i) ☆ L is REC ⇔ $\bar{L}$ is REC.
(ii) ☆ L is RE ⇒ $\bar{L}$ may or may not be RE.
(iii) ☆☆ If L & $\bar{L}$ both are RE ⇒ both are REC.

→ Imp Theorem

If L is RE, then $\bar{L}$ can be RE or NOT RE, but acc. to (iii), both L, $\bar{L}$ are RE, but as we know that complement is not closed under RE ∴ L & $\bar{L}$ must be REC.

then $\bar{L}$, may be in RE or in Not RE.

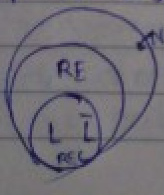
→ but acc. to closure property, complement is not closed under RE.



→ ∴ both must be REC.

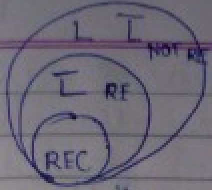
Q. If ∃ L, $\bar{L}$ ⇒

Case I:-



Case II:-

case 3:-



L is RE

which is not possible?

(1) $\bar{L}$ is REC.
(2) $\bar{L}$ is RE
✓(3) $\bar{L}$ is RE but not REC
(4) $\bar{L}$ is not RE
(5) none of these.

RE but not REC.:-

L_u:- universal language of Turing Machine

SA:- $L_u = \{T \mid e(T) \in L(T)\}$ ⟶ that recognizes its own code.

e(T):- binary encoding

NSA:- $L_d = \bar{L_u} = \{T \mid e(T) \notin L(T)\}$ ⟶ there is no such machine that accepts such language.

L_u:- RE but not REC

L_d:- not RE.

★ All strings starting with 1, will have code which starts with a 1, & hence such TM will accept its own code, but all strings starting with 0, will have code which starts with 1, & hence such TM will not accept its own code because the code itself starts with 1 & not with a '0'.

Date:
Page No.

★ Σ^* is countably infinite.
$L \subseteq \Sigma^*$ is countable.

$L_{reg}, L_{CFL}, L_{CSL}, L_{REC}, L_{re}$ → countable

sets of language are countable.

Proof:- We will prove sets of RE languages are countable.

The set of Turing Machines is countable.

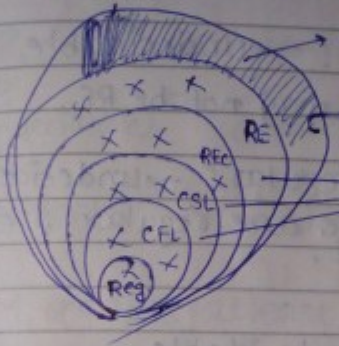
$S_{TM} = \{T_1, T_2, T_3, \ldots\}$ (Set of Turing Machines)
Now, Turing Machines can be described in ~~Turing~~ Binary String.

★ For a Binary String to be a TM, the no. of 0's should be n+1.

∴ each Turing Machine can be expressed in binary form.
Now, $\Sigma^* = (0+1)^*$ is countable, ∴ the binary string of TM is a proper subset of $(0+1)^*$.
∴ S_{TM} is countable.
∴ L_{re} is countable.
& every Language is just subset of L_{re},
∴ the subsets are also countable.
∴ All set of languages are countably infinite.
i.e. set of all reg. languages are countably infinite,
& set of all CFL are countably infinite.
& so on...

Date:
Page No.

★ $L_{not\ RE}$ is uncountably infinite.
set of all languages which are NOT RE is uncountably infinite.



★ In Any axiomatic system, there are some statements that are true but can't be proved by the same axioms.
(Godel's Incompleteness Theorem)

★ In NOT RE, two languages are so close to each other that we can't discriminate b/w two languages.

Decidability

Rice's Theorem :-

Every Non-trivial question regarding RE language is undecidable.
L_1 is RE & L_2 is RE → $L_1 \cup L_2$ is RE. (Trivial questions, trivial questions are those which always give either 'yes' or 'no')

Date:
Page No.

Non-trivial :- These are those which gives answer sometimes as 'yes' & sometimes as 'no'.

L is RE → $\bar{L}$ is RE? → undecidable
(may or may not be RE.)

L is RE → L is regular? → undecidable
as some are regular & some are not.

TM halts in 5 steps → decidable

★ There is no TM which can tell whether a turing machine which will halt on $w \in \Sigma^*$.
(Halting Problem is undecidable)

Decidable :- in finite amount of time

★ L_1 is RE, L_2 is RE → $L_1 \cap L_2$ is RE, $L_1 . L_2$ is RE, L_1^*, L_2^* is RE } Trivially Decidable.

As we dont know after how many steps will be req. to halt the machine.

1. TM halts in finite no. of steps → decidable.
2. Whether a TM makes no left move → decidable
(whether there is a '1' after 4 0's).
3. TM prints some letter ? → decidable
4. TM print "a" → undecidable

★ P1 Halting Problem.

★ P2 State entry problem:-
whether a Turing Machine comes into "q" state.

$P_1 \leq P_2$ (Reduction)

P_1 (undecidable) ⇒ P_2 (undecidable)

P_2 (decidable) ⇒ P_1 (decidable)

(5) TM makes atleast 10 moves → decidable

★ Blank Tape Halting Problem (Accept null input)
This is undecidable, because even though we initially have blank tape, then it must halt but it can create string on its own ← hence it may hang or halt.

★ RE membership or Type 0 Grammar, $w \in L(g)$ ← RE membership
This is undecidable.

★ Post Correspondance Problem

If we have two sets of strings with equal no. of strings,
(u_1, u_2, u_3, u_4)
(v_1, v_2, v_3, v_4)

then there must be $u_i^* . u_j^* . u_k^* = v_i^* . v_j^* . v_k^*$

This is undecidable

Reductions

HP ≤ SEP
 ≤ BTHP
 ≤ PCP

REM ≤ MPCP
 ≤ PCP

☆ Modified PCP :-

In this case

$u_1^* u_i^* u_j^* \dots = v_1^* v_i^* v_j^* \dots$

The first string will be the 1st member

(u_1, u_2, u_3)

(v_1, v_2, v_3)

☆☆ PCP on unary alphabet is decidable

e.g. $u_1 \; u_2 \; u_3$

(11, 111, 1111) → set 1

(111, 11, 1111) → set 2

If every corresponding element, the set 1 has smaller size element, i.e. u_i(length) < v_i(length) & u_i(length) > v_i(length) [for all i], there wont be any PCP solution.

& when some are short & some are long for u_i & v_i, then these are decidable.

Take example:-

(11, 11) → set 1

(1111, 11) → set 2

PCP → decidable

because.

$\underbrace{11^*.11^*}_{set1} = \underbrace{1111^*.11^*}_{set2}$

$(11)^2.(11)^1 = (1111)^1.(11)^1$

⇒ 1111.11 = 1111.11 (Decidable).

Complexity

Date:
Page No.

P, NP, NP-complete, NP-hard :- Decidable Problems

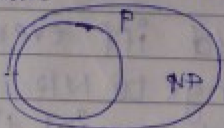
Polynomially bounded DTM → $P = \bigcup DTIME(n^k)$ (Polynomial)

Polynomially bounded NTM → $NP = \bigcup NTIME(n^k)$ (Not deterministic polynomial)

n:-

→ run on a deterministic turing machine.

→ run on a non-deterministic turing machine.



↔ This is because every DTM is an NTM.

$P \subseteq NP$

$NP = \bigcup NTIME(n^k) = \bigcup DTIME(k^n)$ ← exponential time to solve

P:- all practically usable algorithms.

NP:- all problems that can't be solved in polynomial time.

→ $EXP = \bigcup NTIME(k^n)$:- exponentially bounded NTM

★ The algorithms are NP, & not the problems.

★ $P \subseteq NP$

$P = NP$? yes / no } → unknown

★★ If $x \in P \Rightarrow x \in NP$

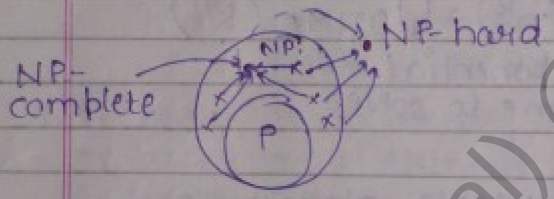
NP-complete:-

These are special problems. A problem is NP complete iff only it is NP hard & Problem must belong to NP.

NP hard:-

A problem is NP hard iff every other problem which belongs to NP is reducible to that problem in polynomial time.

P is NPH iff $\forall L' \in NP \; L' \leq_p L$



★ L is NP complete iff $L \in$ NP hard & $L \in NP$.

★ ~~L is NP hard~~

If L_1 & L_2 are NP complete problems, then $L_1 \leq_p L_2$ & $L_2 \leq_p L_1$

i.e. they must be polynomial time reducible to each other.

★ $x \in P \Rightarrow x \in NP$ } but not converse.

★ $x \in NPC \Rightarrow x \in NPH$ }

★ P problem is closed under $\cup, \cap, L^c, \cdot, *$

★ NP problem is closed under $\cup, \cap, \cdot, *$ & not L^c (?)

e.g. $L_1^c \cup (L_2 \cap L_3)$

L_1 ↑ P, L_2 ↑ P, L_3 ↑ NP

$L_2 \cap L_3$

we know that $P \subseteq NP$

∴ we move L_2 upward to NP.

∴ $NP \cap NP = NP$

& $L_1^c \cup NP$

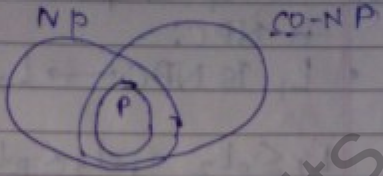
but complement of P is P.

∴ $P \cup NP$

move P to NP

∴ $NP \cup NP$

= NP.



★ if P = NP ⇒ NP = CO-NP

if P = NP ⇒ P = NP = CO-NP

★★ if NPC ∈ P

(a) P = NP (b) NP = CO-NP

(c) both (a) & (b) (d) None

★ If NP = CO-NP ⇔ NP is closed under L^c.

★ If ∃ NPC say L, & $\overline{L} \in NP$ ⇒ NP is closed under L^c.

This is correct because all NP are reducible to NPC, so if complement of NPC belongs to NP, then NP is closed under complement.

$L_1 \le_p L_2$

L_1 is decidable ← L_2 is decidable.

L_1 is recursive ← L_2 is recursive.

L_1 is RE ← L_2 is RE.

L_1 is P ← L_2 is P.

L_1 is NP ← L_2 is NP.

$L_1 \leq_p L_2$ $L_2 \leq_p L_3$
L_2 is P & L_3 is NP
then $L_1 \in P$.

★ NPC problems are reducible to each other.
★ NPH problems may or may not be reducible to each other.

$L_1 \leq_p L_2$
- L_1 is NPH $\longrightarrow$ L_2 is NPH.

★ If a problem is NPH & NP, then it is NPC.

- L_1 is NPC $\longrightarrow$ L_2 is NPH.

$L_1 \leq_p L_2$, $L_2 \leq_p L_1$
L_1 is NPH
$L_2 \in NP$.

$L_2 \in NPH$ } from
$L_2 \in NP$ }
$\therefore L_2 \in NPC$
but $L_2 \leq_p L_1$
$\therefore L_2 \in NPC \longrightarrow L_1 \in NPC$.

★ $L' \leq_p L$ (means, one L' is reducible to L)
$L' \in NP$
then (a) L is NPH
(b) L is NPC
✓(c) none of these

because all NP problems need to be reducible to L in polynomial time, only then L is NPH.
& if $L \in NP$ in case $\forall L' \leq_p L$, then L is NPC.

Examples of NP-complete :-

Bounded Tiling
↓
CNF-SAT
→ 3-SAT
→ MAX-SAT
→ Exact Cover

★ graph is bipartite ⇔ graph is bi-colourable.

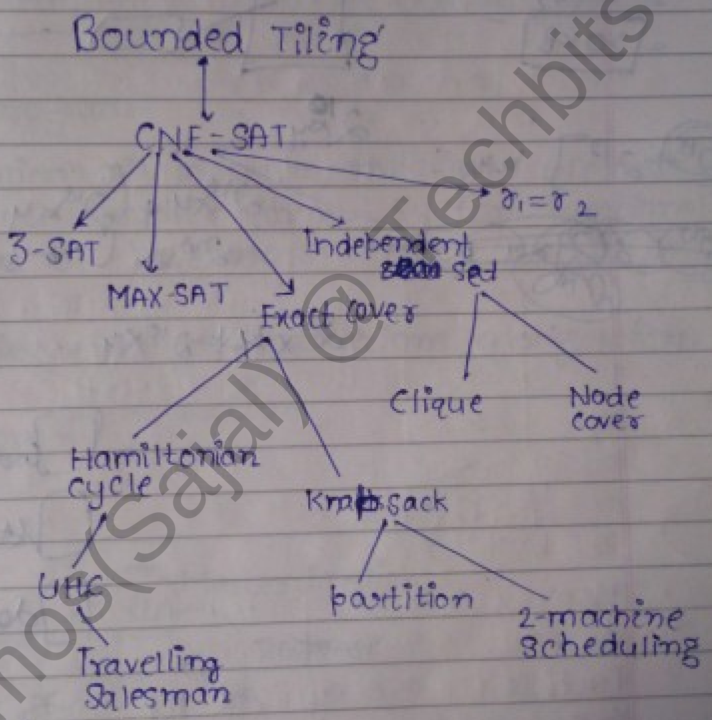
CNF-SAT

↳ $(p+q+r).(p'+q+r).(p'+q+r')=1$

There must be some value of p, q, r for which the eqn. becomes 1.

★★ 2-SAT belongs to P.
eg $(p'+q).(p+q')$.

★★ 3-SAT, CNF-SAT belongs to NP.



P	NP
2-sat	3-sat
Euler cycle	Hamiltonian cycle
Shortest path	Longest path → NP
eq. of dfa	eq. of nfa & eq. of reg. expression
binary partition	partition
2-colourability	k-colourability