Ch#1 Introduction 9hrs: 8 marks (10)

1. Assignment (Tutorial)

→ History of computation and Alan Turing as the father of computation

Ans The ~~this~~ history of computation dates back thousands of years. Humans have used various tools and machines to perform calculations:

i) Abacus: The first calculating device introduced by Chinese businessmen.

ii) Analytical Engine: In 1837, the first conceptual mechanical computer with programmable capabilities Analytical Engine was developed by Charles Babbage, who is also known as father of computer Science.

iii) Turing Machine
In 1936, Alan Turing developed a theoretical model that defined the principles of modern computing.

Alan Turing is widely recognized as the 'father of computation' and a key figure in the history of computation due to his significant work in the field of

theoretical computing and his contributions during World War II. He developed a theoretical computation model known as Turing Machine that laid the foundation for modern computers and played a crucial role in breaking the German Enigma code, accelerating the war effort.

In 1936, Turing published a paper that introduced the concept of universal Turing machine, a hypothetical machine capable of performing any computation that laid the foundation for understanding the limits and capabilities of computation.

During World War II, Turing coorked at Bletchley park, the British code-breaking center, where he played a vital role in deciphering the German Enigma code.

He contributed to the development of early electronic computers like Automatic Computing Engine (ACE)

→ TOC : describes model of computation that reflects real world computers.

→ 3 areas.

i) automata : mathematical automatic machine, eg: TM, FA

ii) Computability: problem are soluble or not?

iii) Complexity : time & memory space complexity

→ Set: Collections of well defined data.

Assignment: Types of sets.

(i) Singleton set: It has only one member.
Eg: $A = \{a\}$

(ii) Empty set: It hasn't any member. Eg: $A = \emptyset$

iii) Disjoint set: Two sets are disjoint set if they has no common data.
Eg: $A = \{1, 2, 3\}$, $B = \{4, 5\} \Rightarrow A \& B$ are disjoint set.

iv) Overlapping set: Two or more sets are overlapping set if they has some common member
Eg: $A = \{1, 2\}$, $B = \{1, 3, 5\}$
so, A & B are overlapping set.

Cardinality $= |V| = 5$

Date:

Page: 22 April

v) Subset : A set is called subset if another set if it has element of another set.

$A = \{1, 2, 3\}$, $B = \{1, 2, 3, 4\}$

So, $A \subseteq B$ (A is subset of B)

vi) Power set: no. of subsets can be created by a set.

Eg: $A = \{1, 2\}$

$P(A) = \{\emptyset, 1, 2, (1, 2)\} = 2^n = 4$

ordered pair like $\{(1,2), (1,4)\}$

Assignment: Types of relation

Date:
Page:

BC) Function and Relation

→ Relation is ordered pair like {(1,2), (3,4)}.
of set

R = {(1,2), (3,4), (1,4)}

domain range.

→ Types of relation:

(i) Reflexive : domain = range, related to itself.
R = {(1,1), (2,2)}

(ii) Symmetric : mirror image. R = {(1,2), (2,1)}

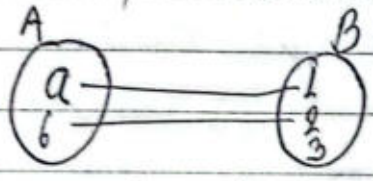
(iii) Transitive : R = {(1,2), (2,3), (1,3)}

iv) equivalence : It is reflexive, symmetric and transitive as well.

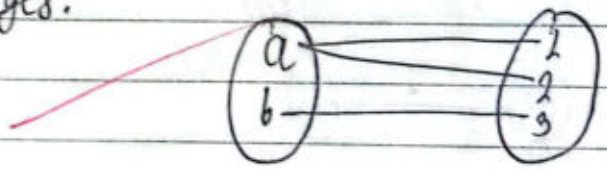
→ Function: It maps domain & range.
Function shows relations between domain and range.

Types:

a) One to One // Injective function
- every domain has only one range

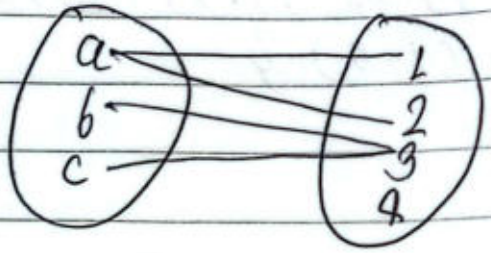


b) One to many : One domain has many ranges.

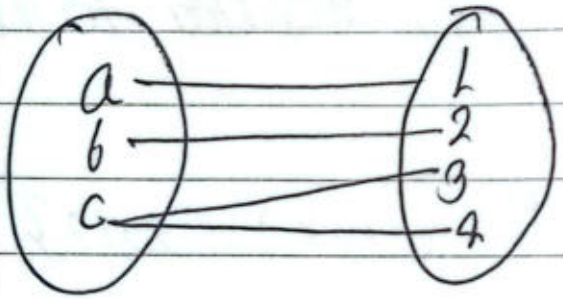


Date:
Page:

c) many to many:
- many domains have many ranges.

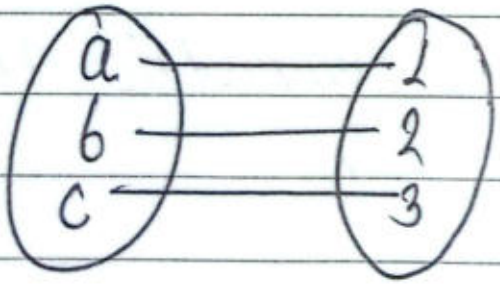


d) Onto function (Surjective)



range = codomain

e) One to one Onto (bijective)



(Σ) finite sequence of consecutive symbol

a) Alphabet: symbols that can be letters/digits.

eg: $\Sigma = \{0,1\}, \{a,b\}$

b) string or word: finite sequence of consecutive symbol. eg: $\Sigma = \{0,1\}$

W = '00', 01, '10', ...

c) length of word: no. of symbols in word.

$W = 0011$, $|W| = 4$

d) empty string: zero length (e, ε, λ, ∧)

e) substring: string z is substring if that appears consecutively in W. eg: '01' is substring of '0011'

f) closure of alphabet:
set of all strings over Σ including empty string. eg: $\Sigma \{0,1\}$

$\Sigma^* = \{ \varepsilon, 0, 1, 00, 01, 110, \ldots \}$

g) Language: Subset of all set of possible string over alphabet. $\Sigma = \{0,1\}$

eg: $L = \{01, 10, 0011, \ldots\}$ // equal no. of 0s & 1s.

h) grammar: formal system for accepting or rejecting strings // rules.

→ Proof of Techniques

1. Proof by Contradiction

In this method we define negative statement (opposite) to the original statement. If the assumption comes false then original statement is true.

Steps: → assume opposite statement
→ show assumption leads contradiction

Eg: Proof if n^2 is an even integer then n is an even

Soln: Step1: we assume the statement is false, suppose n^2 is an even integer then n is odd.

Hence, $n = 2k+1$ for some integer k

$n^2 = (2k+1)^2 = 4k^2+4k+1$
$= 2(2k^2+2k)+1 = 2r+1$
$[r = 2k^2+2k]$

This shows n^2 is odd, which is contradiction to our assumption. Hence, given statement is true.

2. Mathematical Induction

Only +ve integer n defined problem are solved by this method.

Problem: P(n)

→ We have to check no. of n to check whether the statement is true or false. It says that if we have sequence of statements and if we check one statement then other statements are not need to be check.

→ Proof: for all $n \geq 1$, prove that $1^2+2^2+3^2+4^2+\cdots+n^2 = \frac{n(n+1)(2n+1)}{6}$

Soln: we have, $P(n) = 1^2+2^2+3^2+4^2+\cdots+n^2$

$= \frac{n(n+1)(2n+1)}{6}$

Step1: for $n = 1$

$P(1) = \frac{1 \cdot 2 \cdot (2\times1+1)}{6} = 1$ (true)

Step2: for $n = k$

$P(k) = 1^2+2^2+3^2+4^2+\cdots+k^2 = \frac{k(k+1)(2k+1)}{6}$

Step3: for $n = k+1$

$P(k+1) = 1^2+2^2+3^2+4^2+\cdots+k^2+(k+1)^2 = \frac{(k+1)(k+2)(2k+3)}{6}$

$P(K+1) = \frac{K(K+1)(2K+1)}{6} + (K+1)^2 = \frac{(K+1)(K+2)(2K+3)}{6}$

$= (K+1)[K(2K+1) + 6(K+1)] = (K+1)(K+2)(2K+3)$

$= (K+1)(2K^2+7K+6) = (K+1)(2K^2+7K+6)$

Hence proved

→ proof for every positive integer n, prove that $7^n - 3^n$ is divisible by 4.

Soln: we have,

$P(n) = 7^n - 3^n$

Step1: for $n=1$

$P(1) = 7-3 = 4$, (true) which is divisible by 4.

Step2: for $n=k$

$P(K) = 7^K - 3^K = 4d$, $d \in N$

Step3: for $n = K+1$

$P(K+1) = 7^{K+1} - 3^{K+1}$

$= 7^K \cdot 7 - 7 \cdot 3^K + 7 \cdot 3^K - 3^K \cdot 3$

$= 7(7^K - 3^K) + 3^K(7-3)$

$= 7 \times 4d + 3^K \times 4$

$= 4(7d + 3^K)$

which is divisible by 4. Hence proved.

→ prove that for all $n \in N$, $n(n+1)(n+5)$ is a multiple by 3.

Soln: we have,

$P(n) = n(n+1)(n+5)$

Step1: for $n=1$

$P(1) = 1 \cdot 2 \cdot 6 = 12$ (true).

which is multiple by 3.

Step2: for $n=k$, for some integer k

$P(K) = k(K+1)(K+5) = 3d$, $d \in N$

Step3: for $n = K+1$

$P(K+1) = (K+1)(K+2)(K+6)$

$= (K+1)(K^2+8K+12)$

$= (K+1)(K^2+5K+3K+12)$

$= (K+1)[K(K+5) + 3(K+4)]$

$= K(K+1)(K+5) + 3(K+1)(K+4)$

$= 3d + 3(K+1)(K+4)$

$= 3[d + (K+1)(K+4)]$,

which is multiple by 3.

Hence, proved.

3. Pigeon Hole Principle

→ it says at least one box there is two or more pigeons.

→ if (K+1) object is to be placed in K boxes then there is at least one box that has more than 1 object.

Theorem: a function f from a set of with K+1 elements to a set with K elements is not one to one.

→ Generalized:
if there are Kn+1 or more pigeon where K is positive integer and these are n pigeon holes then atleast one pigeon holes, contain K+1 pigeon.

n pigeonholes → Kn+1 pigeons
at least 1 pigeon hole → K+1 pigeons

Q. How many student are there in a class among which atleast 4 of them are born in same month.

Soln: Total no. of month (n) = 12 (Pregon holes)
atleast 4 students are in same months,
i.e. K+1 = 4
K = 3
total no. of student (pigeons) = Kn+1
= 3×12+1
= 37

Q. 25 carates of apples are deliver to a store. The apples are of 3 different sorts and all the apples in each carates of the same sort. Show that among which carates there are at least nine containing same sorts of apple.

Soln: Total no. of carates (n) = 3
total no. of carates of apple (Kn+1) = 25
K×3+1 = 25
∴ K = 8
∴ K+1 = 9
∴ atleast 9 carates containing same sorts of apple.
proved

Q. A bag contain 10 red marble, 10 white marble & 10 blue marble. What is the minimum no. of marble you have choose randomly from the back to ensure that we get 4 marbles of same color.

Soln: no. of colors (pigeon holes) = (n) = 3
no. of marbles (K+1) = 4
∴ K = 3
now, Kn+1 = 3×3+1 = 10
∴ no. of mables from randomly = 10

4. Diagonalization Principle

→ used to prove problems that are unsolvable.

let, R = {(a,a), (b,c), (b,d), (c,a), (c,c), (c,d), (d,a), (d,b)}

A = {a, b, c, d}

now,

	R	a	b	c	d
Ra ←	a	1	0	0	0
Rb ←	b	0	0	1	1
Rc ←	c	1	0	1	1
Rd ←	d	1	1	0	0

D = 1010 (a, b, c, d)

from figure,

Ra = {a}
Rb = {c, d}
Rc = {a, c, d}
Rd = {a, b}

[1, যদি এই]

Complement of diagonal (D') = {0, 1, 0, 1}
= {b, d}

if we compare each of the set Ra, Rb, Rc, Rd with digonal set D, we can see D is different from each row. Thus, complement of Diagonal is different from each row. (unsolvable).

Chomsky's Hierarchy:

→ Hierarchy of grammars

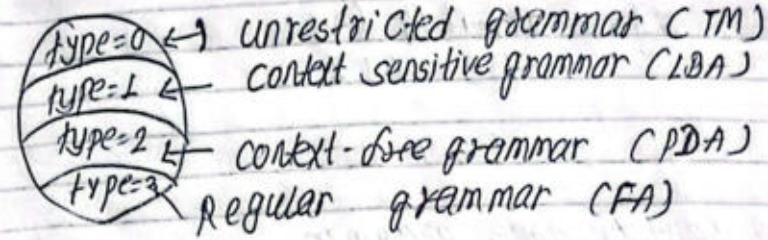


fig: Chomsky's Hierarchy

① type-0 grammar

→ include all grammar.
→ language generated by this called Recursive Enumerable Language (REL).
→ recognized by TM.

rule: α → β
α = non-terminal (in uppercase)
β = terminal or non-terminal (in lowercase)

eg: AB → c
Aa → ab

② type-1 grammar

- context - sensitive grammar
- generated language : context-sensitive language
- recognized by Linear Bound Automata (LBA).

Rules:
with restriction: $(|\alpha| \leq |\beta|)$ (length)

$\alpha \rightarrow \beta$

eg: $A \rightarrow bB$ (✓)
$AB \rightarrow c$ (✗)

(iii) type-2 Grammar

→ context free grammar
→ recognized by Push Down Automata (PDA).

→ rules:
no context on left or right.
i.e. no dependency, can be replaced independently.
→ $|\alpha| = 1$
$A \rightarrow aaBD$
$B \rightarrow aBcd$
[no terminal can be replaced by terminal]

(iv) type-3 Grammar // regular language
→ generated language is regular language.
→ recognized by Finite Automata (FA).

rule:
$\alpha \rightarrow \beta$
eg: $A \rightarrow ab$ (→ or)
$B \rightarrow ab/\varepsilon$

Unit 2 Finite Automata (10hrs) 22.7 marks

And Regular Language

Automata: self acting (automation)
Regular language: simplest mathematical model.

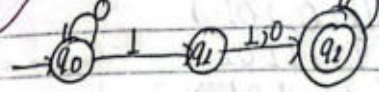
Typ
FA: The self acting machine which has fixed states.

Types of FA:

i) Deterministic Finite Automata (DFA)
→ for each input and state, there is exactly one transition state.

Formal defination: A has 5 tuples, defined as
$M = \{Q, \Sigma, \delta, q_0, F\}$
where
Q = set of finite states
Σ = set of symbols
δ = transition function $(Q \times \Sigma \rightarrow Q)$
q_0 = initial state
F = set of final states

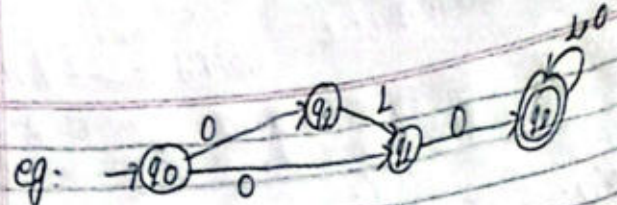
eg:



ii) Non-Deterministic FA (NDFA/NFA)

→ not totally depend on input. we can't determine the exact transition state.

eg:



Instantaneous Description (ID)

→ describes how automata works.

eg: given string 10110, write a entire sequence of state.

Consider finite state automata in which,

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

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

$q_0 = \{q_0\}$

$F = \{q_0\}$

and δ is given by:

state	input	
	0	1
q_0	q_2	q_1
q_1	q_3	q_0
q_2	q_0	q_3
q_3	q_1	q_2

// state diagram

sequence of state:

$q_1 \to q_0 \to q_2 \to q_3 \to q_1 \to q_0$

$(q_0, 10110) \vdash (q_1, 0110)$

$\vdash (q_3, 110)$

$\vdash (q_2, 10)$

$\vdash (q_3, 0)$

$\vdash (q_1, 1)$

$\vdash (q_0, \epsilon)$

Since, q_0 is final state. So string is accepted.

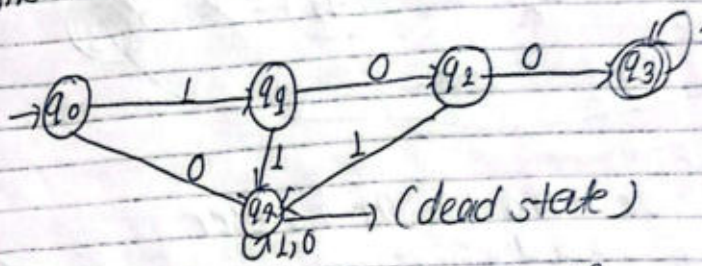
Design DFA to accept string of 0's and 1's starting with '100'. over $\Sigma = \{0, 1\}$.

Soln: substring = 100

length of substring = 3

no. of state = 4

first, we design DFA for '100' substring:



$\therefore M = \{Q, \Sigma, \delta, q_0, F\}$

where, $Q = \{q_0, q_1, q_2, q_3, q_4\}$

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

$q_0 = \{q_0\}$

$F = \{q_3\}$

δ is given by:

state	input	
	0	1
→q_0	q_4	q_1
q_1	q_2	q_4
q_2	q_3	q_4
q_3*	q_3	q_3
q_4	q_4	q_4

, now processing for '1000101'

$(q_0, 1000101) \vdash (q_1, 000101)$

$\vdash (q_2, 00101)$

$\vdash (q_3, 0101)$

$\vdash (q_3, 101)$

$\vdash (q_3, 01)$

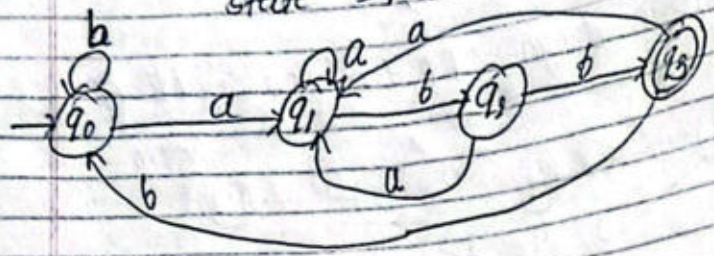
$\vdash (q_3, 1)$

$\vdash (q_3, \epsilon)$

Hence accepted

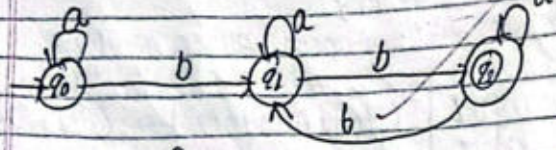
Q. Design DFA a's and b's over $\Sigma = \{a, b\}$, ending with 'abb'.

Soln: substring = 'abb'
length = 3
state = 4

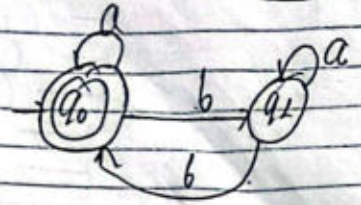


Q. DFA for even no. of b's over $\Sigma = \{a, b\}$

Soln: substring = 'bb'
length = 2
state = 3



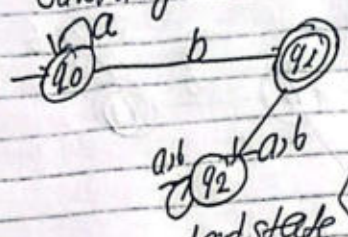
or,



Q. Design a DFA that accept language $L = \{a^n b : n \geq 0\}$

Soln: $L = \{b, ab, aab, aaab\}$

Substring = b



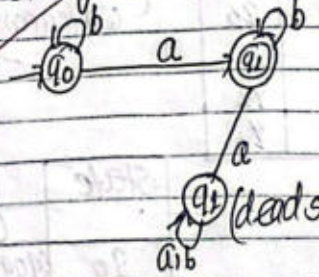
2024 Unit test

Significance of FA:
- foundation of compiler design
- regular language recognization
- hardware and circuit design
- Natural language processing
- Network protocols & security

Q. Design a DFA that accept language $L = \{b^m a b^n : m, n \geq 0\}$

Soln: $L = \{a, bab, ab, ba, bbab\}$

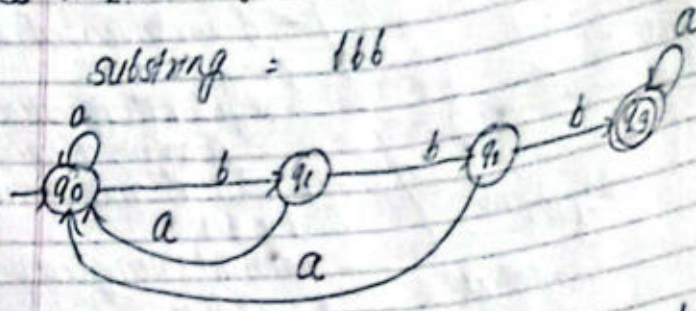
substring: a



Q. Design a DFA that accept string containing 3 consecutive 1's over $\Sigma = \{a, b\}$

Soln: L = {abbb, bbb}

substring = bbb



Equivalence of DFA and NFA (imp)

Q. Construct a DFA equivalent to NFA, $M = \{q_0, q_1, q_2, q_3\}, \{0, 1\}, \delta, \{q_0\}, \{q_3\}$

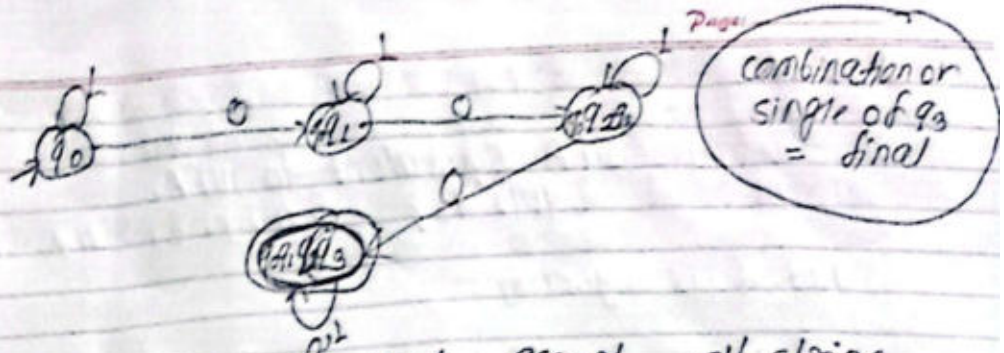
δ:

state	input 0	input 1
$\to q_0$	$q_0 q_1$	q_0
q_1	q_2	q_1
q_2	q_3	q_2
$*q_3$	q_3	q_3

(initial same with new states only)

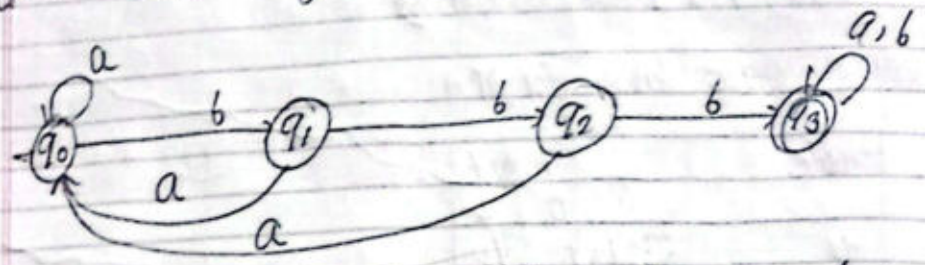
Soln: state table for DFA:

state	input 0	input 1
q_0	$\{q_0 q_1\}$	q_0
$\{q_0 q_1\}$	$\{q_0 q_1 q_2\}$	$\{q_0 q_2\}$
$\{q_0 q_1 q_2\}$	$\{q_0 q_1 q_2 q_3\}$	$\{q_0 q_2 q_3\}$
$\{q_0 q_1 q_2 q_3\}$	$\{q_0 q_1 q_2 q_3\}$	$\{q_0 q_2 q_3\}$

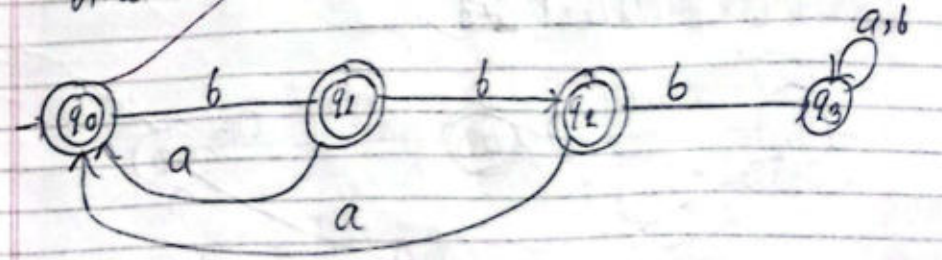


Q. Construct DFA that accept all string that doesn't have 3 consecutive b's over $\Sigma = \{a, b\}$

Soln: first design for 3 consecutive b's:



now, for not 3 consecutive b's, we change final to not final & vice versa.



Assignment

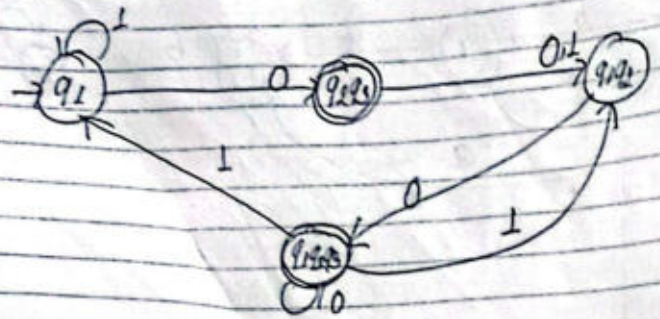
Q. Design DFA equivalent to NFA, $M = (\{q_1 q_2 q_3\}, \{0,1\}, \delta, \{q_1\}, \{q_3\})$

and δ is given by:

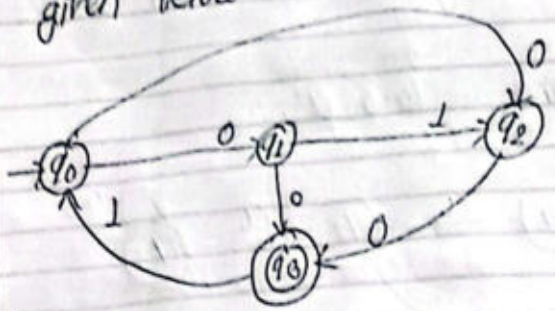
$\delta(q_1, 0) = \{q_2 q_3\}$
$\delta(q_1, 1) = \{q_1\}$
$\delta(q_2, 0) = \{q_1 q_2\}$
$\delta(q_2, 1) = \{\emptyset\}$
$\delta(q_3, 0) = \{q_2\}$
$\delta(q_3, 1) = \{q_1 q_2\}$

Sol^n: state table for DFA:

state	input	
	0	1
q_1	$\{q_2 q_3\}$	q_1
$\{q_2 q_3\}$	$\{q_1 q_2\}$	$\{q_1 q_2\}$
$\{q_1 q_2\}$	$\{q_2 q_3\}$	q_1
$\{q_1 q_2 q_3\}$	$\{q_1 q_2 q_3\}$	$\{q_1 q_2\}$



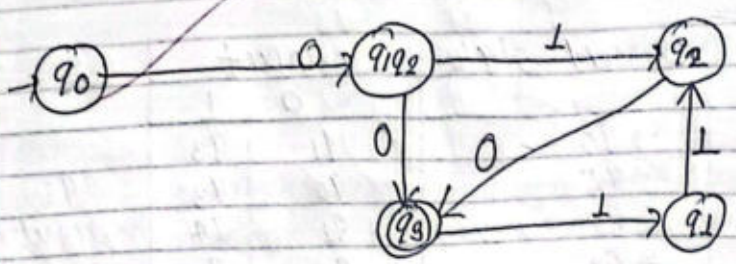
Q2. Construct DFA equivalent to NFA given below.



Sol^n: state table for DFA:

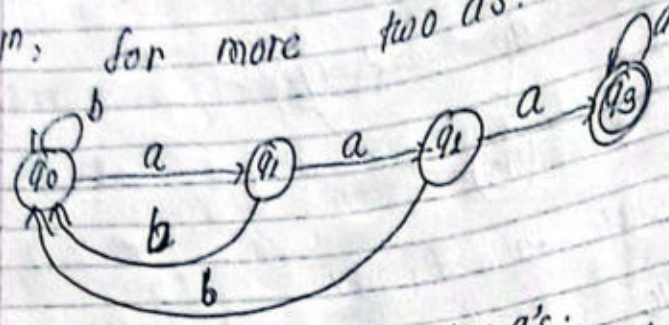
state	input	
	0	1
q_0	$\{q_1 q_2\}$	$\emptyset$
$\{q_1 q_2\}$	q_3	q_2
q_2	q_3	$\emptyset$
q_3	$\emptyset$	q_1
q_1	$\emptyset$	q_2

State diagram:

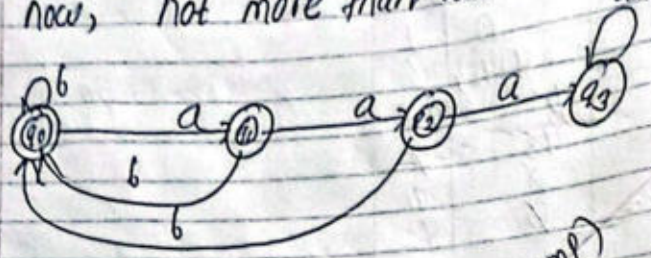


Q3. Design a DFA that accepts all string not having more than two a's over.

Soln: for more two a's:



now, not more than two a's:



Minimization of DFA (imp)

Assignment

current state	input 0	input 1	
→ q0	q1	q3	
q1	q0	q3	
q2	q1	q4	
* q3	q5	q5	
q4	q3	q3	
* q5	q5	q5	

// q2 is not reachable from starting state (no reachable)

step: eliminate state that can't be reached from starting state.

Here, q2 is not reachable. so remove q2.

current state	input 0	input 1
→ q0	q1	q3
q1	q0	q3
* q3	q5	q5
q4	q3	q3
* q5	q5	q5

step2: Divide the table into 2 sets as:

a) set1: Containing row which start from non-final state.

current state	input 0	input 1
→ q0	q1	q3
q1	q0	q3
q4	q3	q3

b) set2: containing row which start from row which is from final state:

current state	input 0	input 1
* q3	q5	q3
* q5	q5	q3

step3: considering set2:

set2 has no similar rows.
It is already minimized.

considering set2:

q3 has same input as q5.
So remove q5 and replace q5 with q3.

current state	input 0	input 1
* q3	q3	q3

step4: combining these set1 & set2 as:

current state	input 0	input 1
→ q0	q1	q3
q1	q0	q3
* q3	q3	q3
q4	q3	q3

as, q3 and q4 has same input, so remove q4.

note: regular language: the language accepted by FA

current state	input 0	input 1
→ q0	q1	q3
q1	q0	q3
* q3	q3	q3

Regular Expression:

→ used to represent regular language accepted by FA

formal defination over $\leq$:

1) any terminal symbols: ϵ or $\wedge$ are regular expressions.

2) The union of 2 regular expressions R1 and R2 is written as R1+R2 is also a regular expression.

3) Concatenation of 2 regular expressions = R1.R2, is also a regular expression.

4) The iteration (Kleene closure) of regular expressions = R1* is also a regular expression

example: let, $\Sigma = \{a, b\}$, find regular expression for language L whose length is exactly 2.

Soln: $L = \{aa, ab, ba, bb\}$

now,

$RE = aa + ab + ba + bb$

$= a(a+b) + b(a+b)$

$= (a+b)(a+b) \approx (a+b)^2$

2. for at least length 2:

$L = \{aa, ab, ba, bb, abb, baa, aba, \dots\}$

R.E: $= (a+b)(a+b)(a+b)^*$

note: $(a+b)^* = \{\epsilon, a, b, ab \dots$ आदि$\}$

$(a+b)^+ = \{a, b, ab \dots$ but not ϵ

3. language accepting all string containing any no. of a's & b's.

Soln: $(a+b)^*$

4. RE for string over $\Sigma = \{0, 1\}$ starting with 1 & ending with 0.

$1(0+1)^*0$

$P = Q + RP$

Identities Rule:

$R = QP^*$

1. $\emptyset + R = R$
2. $\emptyset R + R\emptyset = \emptyset$
3. $\epsilon R = R\epsilon = R$
4. $\epsilon^* = \epsilon$ and $\emptyset^* = \epsilon$
5. $R + R = R$
6. $R^*R^* = R^*$
7. $RR^* = R^*R$
8. $(R^*)^* = R^*$
9. $\epsilon + RR^* = \epsilon + R^*R = R^*$
10. $(PQ)^*P = P(PQ)^*$
11. $(P+Q)^* = (P^*Q^*)^* = (P^* + Q^*)^*$
12. $(P+Q)R = PR + QR$

$R(P+Q) = RP + RQ$

$R = QR + P$

$P = QP^*$

$P = Q + RP$

$R = Q + RP$

$\epsilon + R = R$

imp Arden's Theorem (DFA → RE) // final state

If P, Q, R are regular expressions and $R = Q + RP$, then $R = QP^*$

Assignment

arden's in

Soln: incoming edges:

$q_1 = q_1a + q_2b + \epsilon$ — (I)

$q_2 = q_2b + q_1a + q_3a$ — (II)

$q_3 = q_2a$ — (III)

Date:

Page:

solving (II) & (III)

$$q_2 = q_1 a + q_2 b + q_2 a a$$

$$q_2 = q_1 a + q_2 (b + a^2) \quad -(IV)$$

Solving (I), (III) & (V):

$$q_2 = (q_3 1 + \varepsilon)0 + (q_3 10 + 0)1$$

$$= q_3 10 + 0 + (q_3 10 + 0)1$$

$$= (q_3 10 + 0)(\varepsilon + 1) \quad \text{— (VI)}$$

Solving (II), (IV) & (VI):

$$q_3 = (q_3 10 + 0)0 + ((q_3 10 + 0)(\varepsilon + 1))0$$

$$= q_3(100 + 00) + (q_3 10 + 0)(0 + 10)$$

$$q_3 = q_3(100 + 00 + 1001000)$$

$$q_3 = q_3(100 + 00 + 100 + 1010) + 0(0 + 10)$$

using arden's theorem:

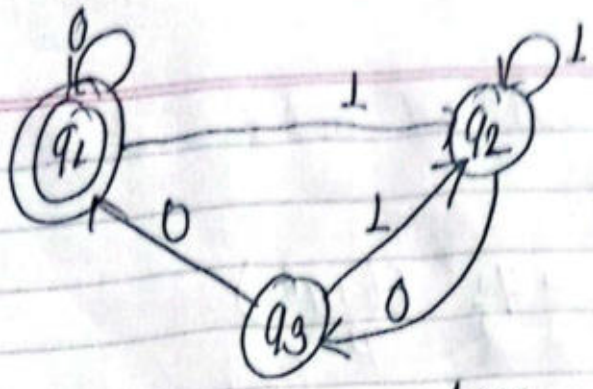
$$q_3 = 0(0 + 10)(100 + 00 + 100 + 1010)^*$$

#

Q: why DFA is called language recognizer?

Soln: Because → it decides whether a string belongs to a language.
→ it operates deterministically.
→ it is efficient.
→ it is foundation for pattern matching.

Q1.



Soln: incoming edges:

$q_1 = q_1 0 + q_3 0 + \varepsilon$ — (I)
$q_2 = q_3 1 + q_2 1 + q_1 1$ — (II)
$q_3 = q_2 0$ — (III)

solving (II) & (III)

$q_2 = q_2 01 + q_2 1 + q_1 1$ — (IV)
$q_2 = q_2 (01 + 1) + q_1 1$
R R P Q

using arden's theorem:

$q_2 = q_1 1 (01+1)^*$ — (V)

solving (I) & (III)

$q_1 = q_1 0 + q_2 00 + \varepsilon$ — (VI)

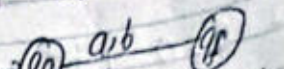
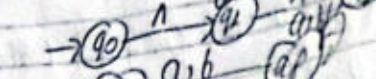
solving (V) & (VI):

$q_1 = q_1 0 + (q_1 1 (01+1)^*) 00 + \varepsilon$
$q_1 = q_1 [0 + 1 (01+1)^* 00)^* + \varepsilon$
R R P Q

using arden's theorem:

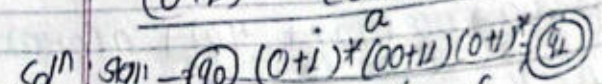
$q_1 = \varepsilon (0 + 1(01+1)^* 00)^*$ #

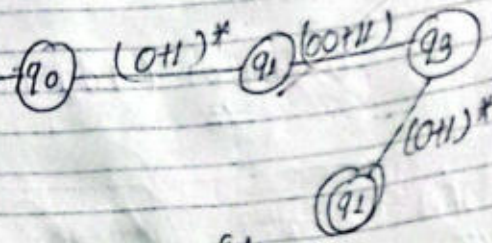
Construction of FA from R.E

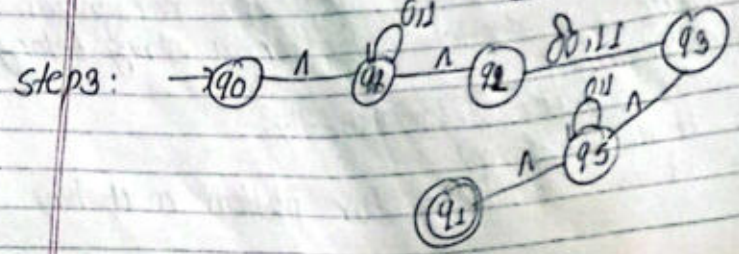
1. $a+b$ 
2. ab 
3. a^* 
4. a^+ 
5. $(a+b)^*$ 
   $(a+b)^+$ 

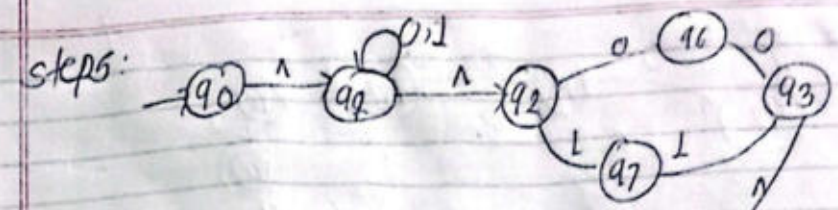
Q1. Construct FA equivalent to RE

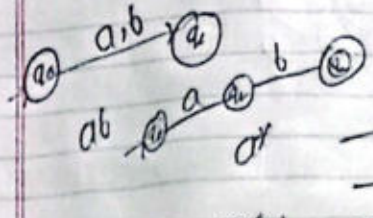
$$\frac{(0+1)^*}{a}\ \frac{(00+11)}{b}\cdot\frac{(0+1)^*}{c}$$

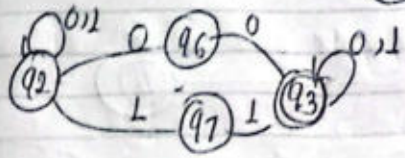
Sol^n: step1 

step2: 

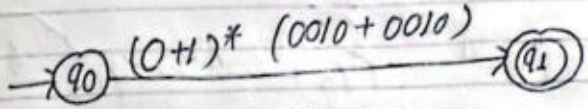
step3: 

steps: 

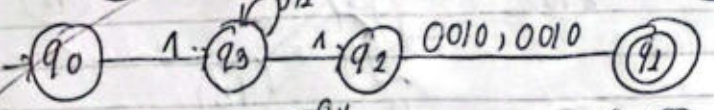


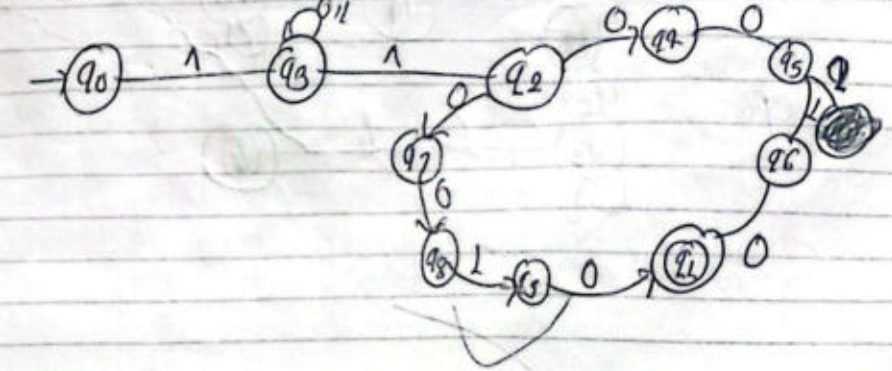
step6: 

Q2. $(0+1)^*\ (0010+0010)$

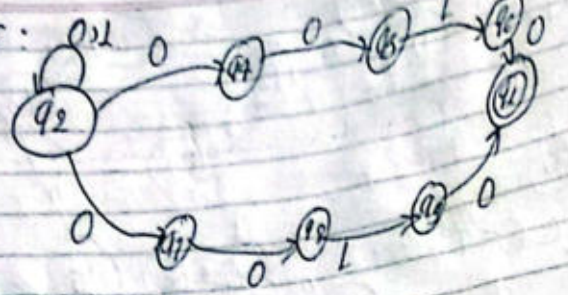
Step1: 

step2: 

steps: 

steps: 

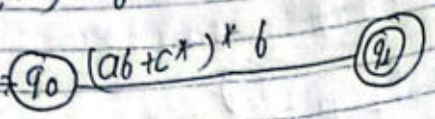
Step6:



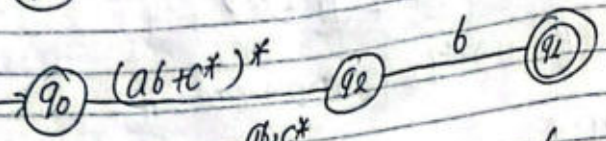
Q3. $(ab+c^*)^* b$

Soln:

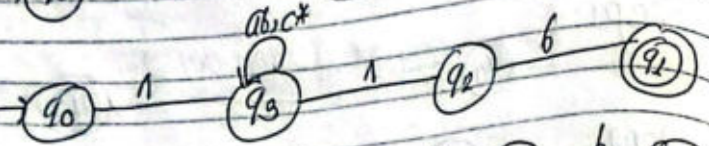
Step1:



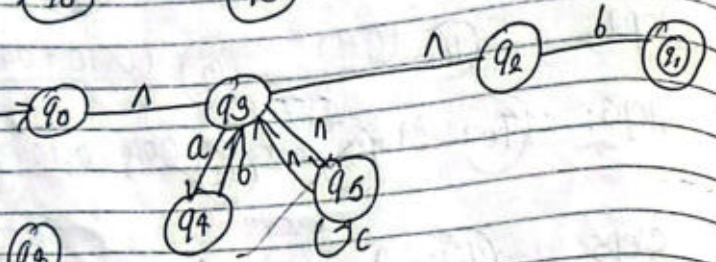
Step2:



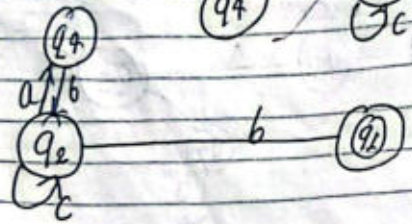
Step3:



Step5:



Step6:



Elimination of null move ($\wedge$ or ε move)

// ε-NFA to without ε (imp)

Steps:

Suppose we replace a null move from a vector V_1 and V_2 then we process as follows:

1. find all the edges starting from V_2.
2. Duplicate all edges of V_2 from V_1 without changing the edge level.
3. if V_1 is initial state makes V_2 as initial state.
4. if V_2 is final state make V_1 as final state.

Assignment — Reduce the given NFA with NULL to without null-move.

$M = \{Q, \Sigma\{a,b,c,d\}, \delta, q_0, \{q_3\}\}$

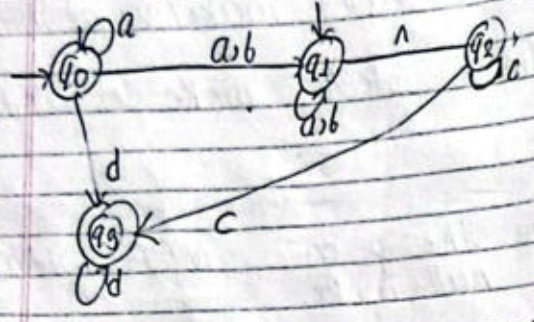
state table:

	input				
state	a	b	c	d	$\wedge$
$\to q_0$	q_0	q_1	$\emptyset$	q_3	q_1
q_1	q_1	q_2	$\emptyset$	$\emptyset$	q_2
q_2	$\emptyset$	$\emptyset$	$[q_2, q_3]$	$\emptyset$	$\emptyset$
$*q_3$	$\emptyset$	$\emptyset$	$\emptyset$	q_3	$\emptyset$

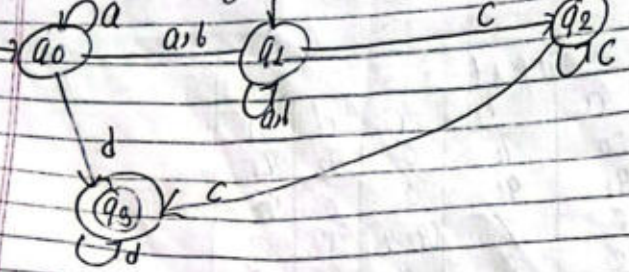
$Q = \{q_0, q_1, q_2, q_3\}$

state diagram

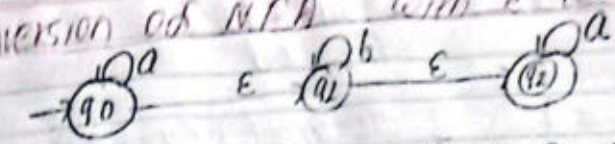
Step: 1st, removing ∧-move between q0 and q1



Step2: removing ∧ between q1 and q2



Conversion of NFA with ε to DFA: (Imp)

eg: 

Soln. ε-closure $(q_0) = \{q_0, q_1, q_2\}$,
ε-" $(q_1) = \{q_1, q_2\}$
ε-" $(q_2) = \{q_2\}$

Note: $\boxed{\varepsilon\text{-closure}(\emptyset) = \emptyset}$

now, DFA transition function δ'

i) $\delta'(\{q_0,q_1,q_2\}, a) = \varepsilon\text{-closure}(\delta(q_0,a) \cup \delta(q_1,a) \cup \delta(q_2,a))$

$= \varepsilon\text{-closure}(q_0 \cup \emptyset \cup q_2)$

$= \{q_0,q_1,q_2\} \cup \{q_2\}$

$= \{q_0,q_1,q_2\}$ //old

ii) $\delta'(\{q_0,q_1,q_2\}, b) = \varepsilon\text{-closure}(\delta(q_0,b) \cup \delta(q_1,b) \cup \delta(q_2,b))$

$= \varepsilon\text{-closure}(\emptyset \cup q_1 \cup \emptyset)$

$= \{q_1,q_2\}$ //new

iii) $\delta'(\{q_1,q_2\}, a) = \varepsilon\text{-closure}(\delta(q_1,a) \cup \delta(q_2,a))$

$= \varepsilon\text{-closure}(\emptyset \cup q_2)$

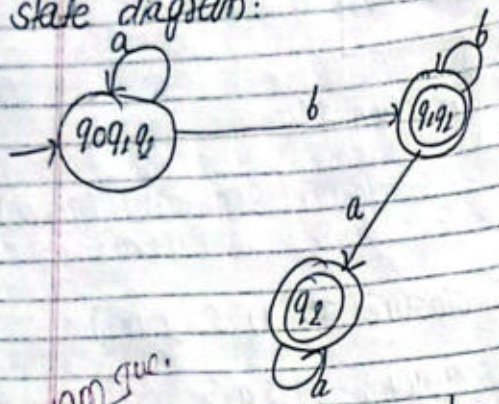
$= \{q_2\}$ //new

iv) $\delta'(q_2, a) = E\text{closure}\{q_2\}$
$= \{q_2\}$ old

v) $\delta(q_2, b) = \varepsilon\text{-closure}(\delta(q_2, b))$
$= \emptyset$

no, new step, so stop.

state diagram:



exam que.

Q.	state	input a	input b	input ε
→	Q_0	Q_0	$\{Q_0, Q_2\}$	Q_1
	Q_1	Q_4	$\{Q_2, Q_3\}$	$\emptyset$
	Q_2	Q_3	$\emptyset$	$\emptyset$
*	Q_3	$\emptyset$	$\emptyset$	Q_4
*	Q_4	Q_3	$\emptyset$	$\emptyset$

Soln: closure of states:
$\varepsilon\text{-closure}(Q_0) = \{Q_0, Q_1\}$
" $(Q_1) = \{Q_1\}$
" $(Q_2) = \{Q_2\}$

$\varepsilon\text{-closure}(Q_3) = \{Q_3, Q_4\}$
$\varepsilon\text{-closure}(Q_4) = \{Q_4\}$

DFA transition function:

i) $\delta'(\{Q_0Q_1\}, a) = \varepsilon\text{-closure}(\delta(Q_0, a) \cup \delta(Q_1, a))$
$= \varepsilon\text{-closure}(Q_0 \cup Q_4)$
$= \{Q_0Q_1\} \cup \{Q_4\}$
$= \{Q_0Q_1Q_4\}$ // new

ii) $\delta'(\{Q_0Q_1\}, b) = \varepsilon\text{-closure}(\delta(Q_0, b) \cup \delta(Q_1, b))$
$= \varepsilon\text{-closure}(\{Q_0Q_2\} \cup \{Q_2Q_4\})$
$= \{Q_0Q_2\} \cup \{Q_2\} \cup \{Q_4\}$
$= \{Q_0Q_1Q_2Q_4\}$ // new

Similarly,
iii) $\delta'(\{Q_0Q_1Q_4\}, a) = \varepsilon\text{-closure}(Q_0 \cup Q_4 \cup Q_3)$
$= \{Q_0Q_1\} \cup \{Q_4\} \cup \{Q_3Q_4\}$
$= \{Q_0Q_1Q_3Q_4\}$ // new

iv) $\delta'(\{Q_0Q_1Q_4\}, b) = \varepsilon\text{-closure}(\{Q_0Q_2\} \cup \{Q_2Q_4\} \cup \emptyset)$
$= \{Q_0Q_2\} \cup \{Q_2\} \cup \{Q_4\}$
$= \{Q_0Q_1Q_2Q_4\}$ // old

v) $\delta'(\{Q_0Q_1Q_2Q_4\}, a) = \varepsilon\text{-closure}(Q_0 \cup Q_4 \cup Q_3 \cup Q_3)$
$= \{Q_0Q_1\} \cup \{Q_4\} \cup \{Q_3Q_4\}$
$= \{Q_0Q_1Q_3Q_4\}$ // old

vi) $\delta'(\{q_0,q_1,q_2,q_4\},b) = \varepsilon\text{-closure}(\delta(\{q_0,q_1,q_2\},b) \cup \delta(\{q_4\},b))$

$= \varepsilon\text{-closure}(\{q_0,q_1\} \cup \{q_2\} \cup \emptyset)$

$= \varepsilon\text{-closure}(\{q_0,q_1,q_2\})$ //old

vii) $\delta'(\{q_0,q_1,q_3\},a) = \varepsilon\text{-closure}(q_0 \cup q_4 \cup q_3)$

$= \{q_0,q_1\} \cup \{q_4\} \cup \{q_3,q_4\}$

$= \{q_0,q_1,q_3,q_4\}$ //old

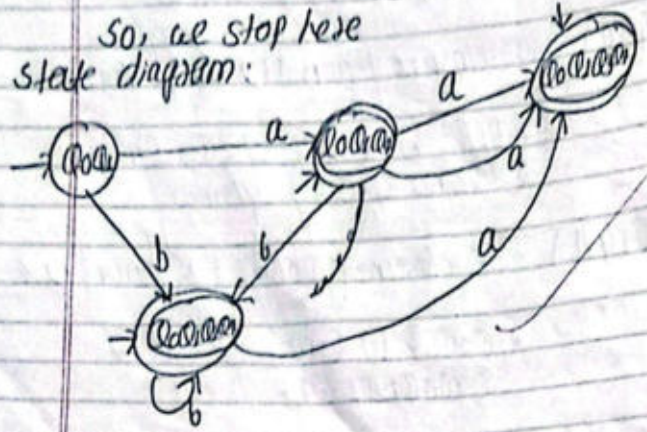
viii) $\delta'(\{q_0,q_1,q_4\},b) = \varepsilon\text{-closure}(\{q_0,q_2\} \cup \{q_2,q_4\})$

$= \{q_0,q_2\} \cup \{q_2\} \cup \{q_4\}$

$= \{q_0,q_1,q_2,q_4\}$ //old

So, we stop here

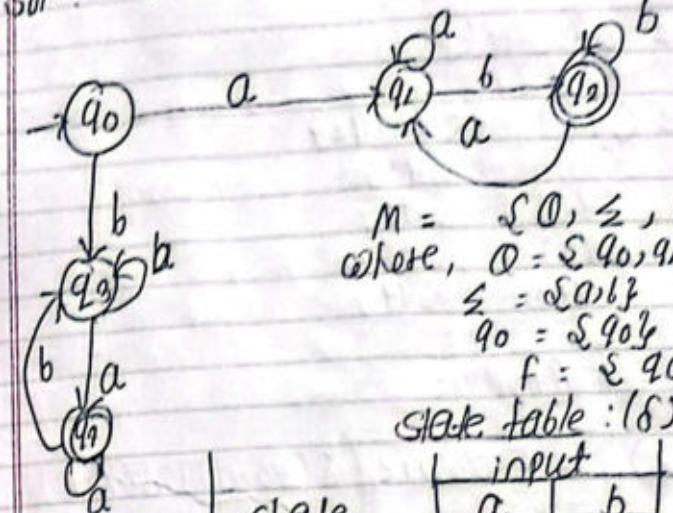
State diagram:



20 M Spring

1b) Construct a DFA that recognizes language L that accepts the set of strings that starts and end with different symbols over $\Sigma = \{a,b\}$ and test your design with a valid string.

Soln: $L = \{ab, ba, abb, baa \ldots\}$



$M = \{Q, \Sigma, \delta, q_0, F\}$

where, $Q = \{q_0, q_1, q_2, q_3, q_4\}$

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

$q_0 = \{q_0\}$

$F = \{q_2\}, \{q_4\}$

State table: (δ):

State	input a	input b
→ q_0	q_1	q_3
q_1	q_1	q_2
* q_2	q_1	q_2
q_3	q_4	q_3
* q_4	q_4	q_3

now, processing for, 'ababab'

$(q_0, ababab) \vdash (q_1, babab)$

$\vdash (q_2, abab)$

$\vdash (q_1, bab)$

$\vdash (q_2, ab)$

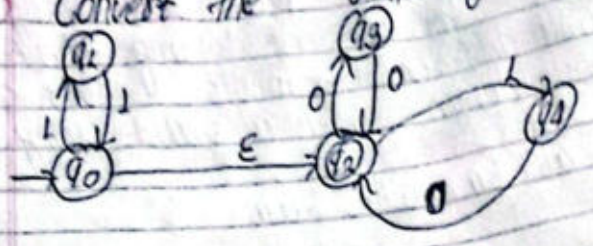
$\vdash (q_1, b)$

$\vdash (q_2, \varepsilon)$

Hence accepted

2079 Spring

20) Convert the following NFA to equivalent DFA



Soln: closure of states:

ε-closure (q0) = {q0, q2}
" (q1) = {q1}
" (q2) = {q2}
" (q3) = {q3}
" (q4) = {q4}

→ DFA transition function:

δ'({q0q2}, 0) = ε-closure(δ(q0,0) ∪ δ(q2,0))
= ε-closure(q3)
= {q3} //new

δ'({q0q2}, 1) = ε-closure(q1 ∪ q4)
= ε-closure({q1} ∪ {q4})
= {q1q4} //new

δ'(q3, 0) = ε-closure(q2)
= {q2} //new

δ'(q3, 1) = ε-closure{q4}
= {q4} //new

δ'({q0,q4}, 0) = ε-closure{q2}
= {q2} //new old

δ'({q1,q4}, 1) = ε-closure{q0 ∪ ∅)
= {q0q2} //old

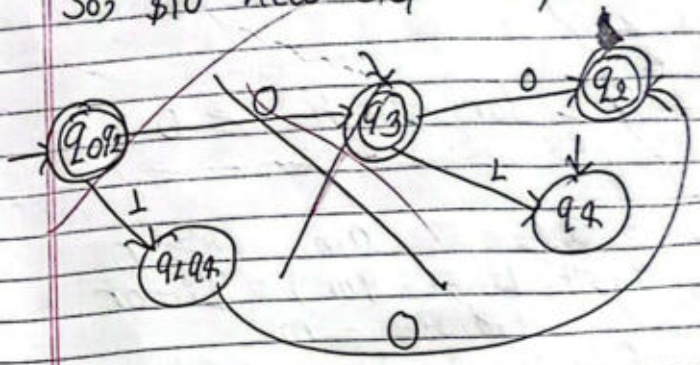
δ'(q2, 0) = ε-closure{q3)
= {q4} //old

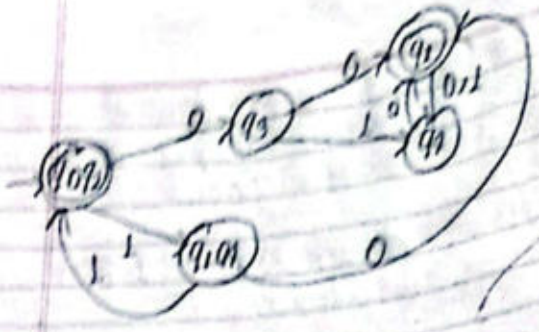
δ'(q2, 1) = ε-closure{q4}
= {q4} //old

δ'{q4, 0) = ε-closure{q2}
= {q2} //old

δ'(q4, 1) = ε-closure{∅)
= ∅

So, no new step, stop here.





Pumping Lemma for regular set:

→ used to prove language is not regular.

Theorem: Let, $m = (Q, \Sigma, \delta, q_0, F)$ be a finite automata with 'n' state that accepts regular language 'L'. Let ω be the string, such that $\omega \in L$ and length of string $|\omega| \geq m$.
if $m \geq n$, there exists substring then, x, y, z such that:

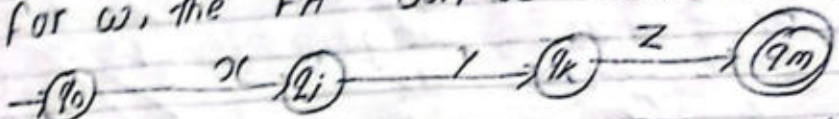
i) $\omega = xyz$
ii) $y \neq \wedge$
iii) $xy^i z \in L$ for each $i \geq 0$

proof: Let, $\omega = a_1 a_2 \ldots a_m$, $m \geq n$
$\delta(q_0, q_1, q_2, \ldots q_m) = q_i$ for $i = 1, 2, 3, \ldots m$
Here, $Q = \{q_0, q_1, q_m\}$ is sequence of states in path with path value $\omega = a_1 a_2 \ldots a_m$

now, input string can be decomposed into substring as,
$x = a_1 a_2 \ldots a_j$
$y = a_{j+1} \ldots a_k$
$z = a_{k+1} \ldots a_m$

since, we know that for any regular expression there exist a path from initial state to final state with path value in the given regular expression.

for ω, the FA can be constructed as:



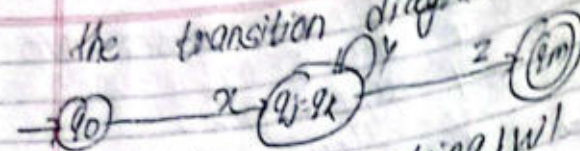
length of string, $|\omega| = 3$ (i.e. x, y, z.) $= m$
no. of state(n) $= 4$ (i.e. $q_0 q_i q_k q_m$)

On reading the string xyz, a new state are added on existing state but by the destination by the pumping Lemma we have $|\omega| \geq m$ and $m \geq n$.

Thus by using Pigeon hole principle there must be at least two state in Q. As, there are only 'n' existing state define but on applying the input string the state becomes $n+1$.

Thus, among various pair of repeated states we take first as q_i and q_k, for merging and hence the path with ω in

the transition diagram of FA is:



now, length of string, $|w| = 3 = m$

no. of state $(n) = 3$

so, $m \geq n$ (true)

since, $w = xyz$, where:

$x = a_1 a_2 \ldots a_j$

$y = a_{j+1} \ldots a_k$

$z = a_{k+1} \ldots a_m$

$0 \leq j < k \leq n$

$|xy| \leq n$

so,

we can write,

$0 \leq j < k \leq n$ and this implies that,

$|xy| \leq n$, // $0 \leq j < k \leq n$

Here,

$xy^i z \in L$ for each $i \geq 0$

i) when, $i = 0$

$w = xz$ (accepted by FA)

ii) when, $i = 1$

$w = xyz$ (accepted)

iii) when $i = 2$

$w = xy^2 z$ (accepted)

and so on.

Hence, we conclude that this condition is valid for all regular expression.

Thus, the string of y can be pump in several no. of time to generate a large string.

Hence, $xy^i z \in L$ for $i = 0$ //

⇒ Application of Pumping Lemma: $|xy| \leq n$, $|y| \neq 0$

step1: assume that L is regular let 'n' be no. of state in FA.

step2: choose a string 'ω' such that $|\omega| \geq n$. use pumping lemma to write $\omega = xyz$, $|xy| \leq n$ and $|y| > 0$

step3: find a suitable 'i' for which $xy^i z \notin L$. This contradict our assumption. So L is not regular.

example: Show that $L = \{ww \mid w \in (a,b)^*\}$ is not regular.

Soln: step1: Assume L is regular language

step2: let, $ww = a^n b a^n b$, where $w = a^n b$

Then, by pumping lemma, we can write,

$w'w = xyz$, $|xy| \leq n$, $|y| > 0$

suppose, $n = 7$

$w = a^7 b a^7 b = aaaaaaa\,b\,aaaaaaa\,b$

case1: y has only parts of a

aaaaaaaa b aaaaaaab
x y z

case2: y has only parts of b

aaaaaaa b aaaaaaab
x y z

case3: y has both a & b

aaaaaaaa b aaaaaaab
x y z

step3: we want to find i such that $xy^iz \notin L$.

take $i = 2$

incase1: $xy^2z = xyyz$
$= aaaaaaaaaaaaaaab$
$= a^9ba^7b$

this is not is pattern. also,
$|xy| = 7$ i.e. $|xy| \leq n$ (true)
$xy^2z \notin L$

incase2: $xy^2z = xyyz$
$= aaaaaaabbaaaaaaab$
$= a^7b^2a^7b$

this is not in pattern.
thus, $xy^2z \notin L$,
$|xy| = 8$, i.e. $|xy| \leq n$ (false).

incase3: $xy^2z = xyyz$
$= aaaaaaaa\,gtab\,aaaaaaab$
$= a^8b^2a^7b$

$n: a^5ba^5b$ (x = aaaaa, y = a, z = baaaaab)
$a^5b^2a^5b$ not in pattern
$|xy| = 8$, i.e. $|xyl| \leq n$ (false)

thus in all cases we get contradiction,
so, given language L is not regular

Assignment: Show that, $L = \{0^n1^n / n \geq 1\}$ is not regular

Sol^n: Here, $L = \{0^n1^n / n \geq 1\}$

step1: Assume that L is regular language.

step2: let, $\omega = 0^n1^n$, by pumping lemma, we can write, $\omega = xyz$, $|xy| \leq n$, $|y| > 0$
suppose, $n = 5$
$\omega = 0^51^5 =$ 00000 11111

case1: y has only parts of 0:

00000 11111
x y z

case2: y has only parts of 1:

00000 11111
x y z

case3: y has both parts of 0 and 1:

00000 11111
x y z

steps: we have to find i such that $xy^iz \notin L$

take, $i = 2$:

incase: $xy^2z = 0000000\ 1111$
$= 0^7 1^5$

this is not in pattern.
So, $xy^2z \notin L$.
also, $|xy| = 7$
i.e. $|xy| \leq n$ (true) //false.

incase: $xy^2z = 00000\ 11111\ 11$
$= 0^5 1^7$
this is not in pattern, So $xy^iz \notin L$
also, $|xy| = 8 \leq n$ (false)

incase3: $xy^2z = 0000\ 0101\ 1111$
$= 0^6 1^6$
this is not in pattern, So $xy^iz \notin L$.
also, $|xy| = 6 \leq n$ (false)

thus, in all cases we get contradiction, So given language L is not regular.

Q. Show that $L = \{a^p \mid p \text{ is prime}\}$ is not regular.

Soln: steps: Let, L be a regular language.

step2: let, $p = 5$

$\because \omega = a^5 = \underset{x}{aa}\,\underset{y}{aa}\,\underset{z}{a}$, $|xy| \leq p$, $|y| > 0$

step3: Choose $i = 3$
$\omega = xy^3z = aa\,aaaaa\,a$
$= a^9$

since, g is not prime. so, $xy^iz \notin L$.

also, $|xy| = 5 \leq 5$ (true)

since, it contradicts our assumption. So L is not regular language.

Closure properties of regular sets / Language

1) if L_1 and L_2 are regular, then $L_1 \cup L_2$ is also regular.

proof: Let, $M_1 = (Q_1, \Sigma_1, \delta_1, q_1, f_1)$ be an NFA that accepts regular language L_1 (i.e. $L_1 = L(M_1)$)
and, $M_2 = (Q_2, \Sigma_2, \delta_2, q_2, f_2)$ be another NFA that accepts regular language L_2 i.e. ($L_2 = L(M_2)$).

Here, Q_1 and Q_2 are disjoint sets.
now, we construct NFA:
$M = (Q, \Sigma, \delta, q, F)$
such that if and only if it can accepts:

$L = L_1(M_1) \cup L_2(M_2)$

where, q is new state not in Q_1 and Q_2.

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

$\Sigma = \Sigma_1 \cup \Sigma_2 \cup \{\wedge\}$

$\delta = \delta_1 \cup \delta_2 \cup [\delta(q, \wedge) \to q_1, \delta(q, \wedge) \to q_2]$

$F = F_1 \cup F_2$

M begins any computation by non-deterministically, choosing to either initial state of M_1 or the initial state of M_2 and interprets M initiates either M_1 or M_2.

If $\omega \in \Sigma^*$, then

$(q, \omega) \xrightarrow[M]{*} (p, \wedge)$

for some $p \in F$, iff

$(q_1, \omega) \xrightarrow[M_1]{*} (p, \in)$ [$p \in F_1$ or $p \in F_2$]

or,

$(q_2, \omega) \xrightarrow[M_2]{*} (p, \in)$

Hence, $L_1 \cup L_2$ is also regular.

ii) If L_1 and L_2 are regular then $L_1 \cdot L_2$ is also regular.

Soln: Let, $M_1 = (Q_1, \Sigma_1, \delta_1, q_1, F_1)$ be an NFA that accepts regular language L_1 and

$M_2 = (Q_2, \Sigma_2, \delta_2, q_2, F_2)$ be another NFA that accepts regular language L_2.

Here, Q_1 and Q_2 are disjoint sets.

Now we construct NFA:

$M = (Q, \Sigma, \delta, q, F)$, such that it accepts L_1 and L_2.

i.e. $L = L_1(M_1) \cdot L_2(M_2)$ ($\wedge$-$\in$)

$L_1 \cdot L_2$

where, $Q = Q_1 \cup Q_2$

$\Sigma = \Sigma_1 \cup \Sigma_2 \cup \{\wedge\}$

$\delta = \delta_1 \cup \delta_2 \cup [\delta(F_1, \wedge) \to q_2]$

$q = q_1$

$F = F_2$

formally,

$(q, \omega) \xrightarrow[M]{*} (p, \in)$ for some $p \in F$

iff, there exist, $\omega_1 \cdot \omega_2 \in \Sigma^*$ and $p_1 \in F_1$, $p_2 \in F_2$ such that,

$\omega = \omega_1 \cdot \omega_2$ and $(q_1, \omega_1) \xrightarrow[M_1]{*} (p_1, \in)$

and,

$(q_2, \omega_2) \xrightarrow[M_2]{*} (p_2, \in)$

iii) If L_1 and L_2 are two regular language, then $L_1 \cap L_2$ is also regular.

proof: Let, $M_1 = (Q_1, \Sigma_1, \delta_1, q_1, f_1)$ be a DFA that accepts language L_1 and $M_2 = (Q_2, \Sigma_2, \delta_2, q_2, f_2)$ be another DFA that accepts language L_2.

We assume that alphabet of both automata are same and Σ is union of alphabets of L_1 and L_2 if they are different.

Construct, a DFA: $M = (Q, \Sigma, \delta, q, F)$ such that it can accepts $L_1 \cap L_2$.

where, $Q = Q_1 \times Q_2$
$\Sigma = \Sigma_1 \cup \Sigma_2$
$\delta = (\delta_1, \delta_2)$
$q = (q_1, q_2)$
$F = f_1 \times f_2$

if, M_1 goes from state p to s on reading ω and M_2 goes from q to t on reading ω, then M will be from state (p, q) to (s, t) on reading ω.

formally, if $(p, \omega) \xrightarrow[M_1]{*} (s, \varepsilon)$ and $(q, \omega) \xrightarrow[M_2]{*} (t, \varepsilon)$, then

$(\{p, q\}, \omega) \xrightarrow[M]{*} (\{s, t\}, \varepsilon)$

$\delta(\{p, q\}, \omega) = (\delta_1(p, \omega), \delta_2(q, \omega))$

String ω is accepted by M iff ω is accepted by M_1 and M_2.

4. if L is regular language over Σ then $\bar{L} = \Sigma^* \setminus L$ is also regular.

proof: Let, L is recognized by DFA.
$A = (Q, \Sigma, \delta, q_0, F)$

Then, $\bar{L} = L(B)$, where B is a DFA,
$B = (Q, \Sigma, \delta, q_0, Q \setminus F)$

i.e. B is exactly like A but final state of A has become non final state of B and viceversa.

Then, ω is in $L(B)$ iff $\delta(q_0, \omega)$ is in $Q \setminus F$, which occurs iff ω is not in $L(A)$.

proved

v) The closure of regular language i.e. L^* is also a regular.

Soln: Let, $M_1 = (Q_1, \Sigma_1, \delta_1, q_1, F_1)$ be an NFA that accepts a regular language L. Now we construct, $M = (Q, \Sigma, \delta, q_0, F)$ such that it can accepts L^* i.e. $L = L(M)^*$.

where,

$Q = Q_1 \cup \{q\}$

$\Sigma = \Sigma_1 \cup \{\wedge\}$

$\delta = \delta_1 \cup [(q, \wedge) \to q_1, (q_1, \wedge) \to f_1, (f_1, \wedge) \to q_1]$

$q_0 = q$

$F = f_1 \cup \{q\}$

Here, M consists a state of M_1 & all transition of M_1 also, and final state of M_1 is also a final state of M. The new initial state $\wedge$ is accepted.

It follows the inspection of M that if $\omega \in L(M)$ then either $\omega = \wedge$ or $\omega = \omega_1, \omega_2, \ldots, \omega_i$ for $i = 1, 2, 3, \ldots$ there is $f_i \in F$, such that

$$(q_1, \omega) \xrightarrow[M]{*} (f_1, \varepsilon)$$

proved

vi) if L and M are regular then difference L/M is also a regular.

proof: Here,

$L/M = L \cap \bar{M}$

we already know that, regular language are closed under complement and intersection. so, L/M is also regular.

Decision Properties / Algorithm for regular Language

1. The membership Problem
2. The emptiness "
3. The infiniteness "

→ 1. The membership Problem:

: Is string 'ω' is regular language?

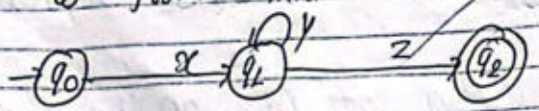
i) Assume L is regular language represented by a DFA 'M'.

ii) Simulate the action of M on the sequence of input symbols forming ω.

iii) If processing the ω, M reaches the final state, then ω is member of L, otherwise not.

2. The emptiness problem
→ Is L is empty?
i) Assume a DFA for language 'L'.
ii) Construct a transition graph. Compute the set of states reachable from the start state.
iii) If for any string, the final state is reachable then the string is present in L, else no.

3. The infiniteness problem
→ Is L is infinite?
i) Start with a DFA for the language L.
ii) If the DFA has n state and the language contains any string of length n or more, then the language is infinite, otherwise not.
iii) If n-state accepts the string ω of length n or more, then there must be a state that appear twice on the path labeled 'ω' from initial state to final state.



$\omega = xyz$
$xy^iz \in L$, for $i \geq 0$

Since, y is not null, there is an infinite no. of string in L.

Assignment –
Q. properties of Regular Expression or language

a) Commutative
Statement: $R_1 + R_2 = R_2 + R_1$
e.g: $(a+b) = (b+a)$. Useful for union.
$R_1 R_2 \neq R_2 R_1$

b) Associative law
→ Applies to Union and Concatenation.
Statement: $(R_1+R_2)+R_3 = R_1+(R_2+R_3)$
$(R_1R_2)R_3 = R_1(R_2R_3)$
eg: $(a+b)+c = a+(b+c)$
$(a.b).c = a.(b.c)$

c) Distributive law
→ Applies to concatenation over union.
Statement. $R_1(R_2+R_3) = R_1R_2 + R_2R_3$
eg: $a(b+c) = ab+ac$

d) Identity element
for Union:
$R+\emptyset = R$ ($\emptyset$ is identity element).
for concatenation:
$R.\varepsilon = R$ (ε is identity element)

e) Annihilator (zero element)
Statement: $R.\emptyset = \emptyset.R = \emptyset$
$\emptyset$ is Annihilator element.
$a.\emptyset = \emptyset$

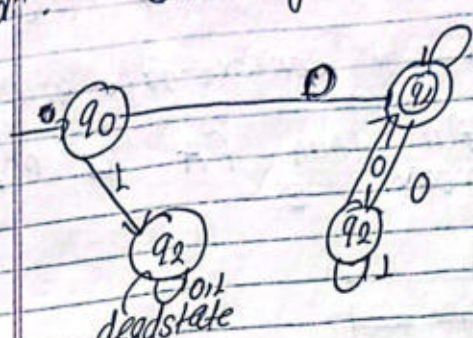
(vi) Idempotent law
- Applies to Union.
statement: $R+R=R$
combining a language with itself doesnot change it.
example: $a+a=a$
$R \cdot R \neq R$

(vii) Law of closure
Regular language are closed under operations like union, intersection, concatenation, Kleene star, complement, reversal, etc.

if, R_1 and R_2 are regular, then:
- $R_1 + R_2$ is regular.
- $R_1 \cdot R_2$ is regular
- R_1^* is regular
- $R_1 \cap R_2$ is regular.
- $\overline{R_1}$ is regular.

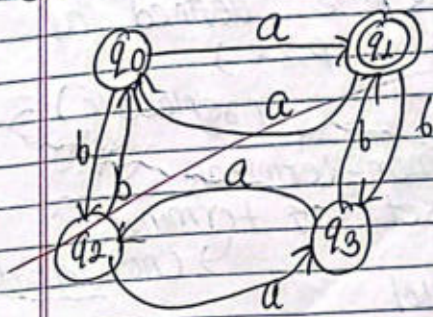
Q1. Construct a DFA start with '0' and has odd no. of 0's over $\Sigma = \{0,1\}$
$L = \{0, 01, 0100 \ldots\}$

Soln: substring = 0



Q2. Construct a DFA that accepts odd no. of a's and even no. of b's over $\Sigma = \{a,b\}$.

Soln, $L = \{a, abb, aaabb \ldots\}$



b, a, b

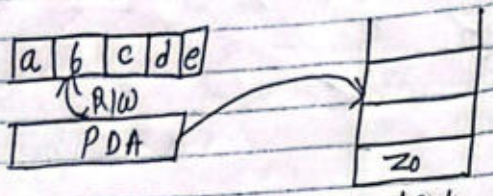
Unit 3 Context Free Language and Pushdown Automata (13hrs, 28 marks)

1. Context-free Language (2029, spring)

→ Language accepted by push down Automata (PDA)

→ Language generated by context-free grammar.

→ PDA is more powerful than FA as PDA has extra memory.



Context-free Grammar: // CFG

- A CFG is a set of quadruple defined as:
  $G = \{V, \Sigma, P, S\}$ (P // R)
  (replacable) (A→B(C))

where,
- V = set of variables / non-terminals
- Σ = I/P symbols or set of terminals (not replacable)
- P = Production rule
- S = starting symbol

note: $S \in V$

- A CFG has production rule of the form:

$A \to \alpha$

where, $\alpha = \{\Sigma \cup V\}^*$, $A \in V$

OR

non-terminal → non-terminals
non-terminal → terminal → (represented by small letter)
non-terminal → ε
(represented by capital letter)

Assignment: Q. Why CFG is called Context-free?

Soln: A Context-free Grammar is called context-free because its production rules are applied without considering the surrounding symbols (context) of the non-terminal being replaced. There is no dependency nonterminal element can replaced by ~~non~~-terminal element.

in a CFG, a production rule has the form.

$A \to \alpha$

where, A is a single nonterminal symbol
α is string of terminal or nonterminal symbol

$A \xrightarrow{\alpha}$ nonterminal A can be replaced by terminals element.

Q. Construct a CFG for the language having any no. of a's over $\Sigma = \{a\}$?

Soln. $L = \{\varepsilon, a, aa, aaa, \ldots\}$

As we know,

$R.E = a^*$

now, we define production rule:

$S \to \varepsilon$
$S \to aS$

$\Big\rangle \quad S \to \varepsilon / aS$

So, the grammar is

$G = \{V, \Sigma, P, S\}$

where,

$V = \{S\}$
$\Sigma = \{a\}$
$S = \{S\}$
$P = \{S \to \varepsilon, S \to aS\}$

now, we can derive 'aaaa' string from Grammar as,

$S \to aS$
$\to aaS$ [$S \to aS$]
$\to aaaS$ [$S \to aS$]
$\to aaaaS$ [$S \to aS$]
$\to aaaa\varepsilon$ [$S \to \varepsilon$]
$\to aaaa$

Hence, accepted.

$[V, \Sigma, P, S]$

Q. Construct CFG for R.E $(0+1)^*$

Soln: $L = \{\varepsilon, 0, 1, 01, 10, 101, 010, \ldots\}$

R.E $= (0+1)^*$

production rule:

$S \to \varepsilon$
$S \to 0S$
$S \to 1S$

Grammar: $G = \{V, \Sigma, P, S\}$

where,

$V = \{S\}$
$\Sigma = \{0, 1\}$
$S = \{S\}$
$P = \{S \to \varepsilon, S \to 0S, S \to 1S\}$

now, we can derive '0011' string from Grammar as.

$S \to 0S$
$\to 00S$ ($S \to 0S$)
$\to 001S$ ($S \to 1S$)
$\to 0011S$ ($S \to 1S$)
$\to 0011\varepsilon$ ($S \to \varepsilon$)
$\to 0011$

Hence, accepted.

Q3. Construct CFG for string having atleast two a's over $\Sigma = \{a, b\}$

$$\underset{\sim}{A}$$

Sol^n: Q4 R.E. = $0(0+1)^*1$

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

$L = \{01, 10, 001, 110, \ldots\}$

production rule,

$S \to 0A1$

$S \to 1A0$

$A \to \epsilon / 0A / 1A$

So, the grammar is:

$G = \{V, \Sigma, P, S\}$

$V = \{S, A\}$, $\Sigma = \{0, 1\}$, $S = \{S\}$,

$P = \{S \to 0A1, S \to 1A0, A \to \epsilon / 0A / 1A\}$

now, we derive '001001' from grammar as:

$S \to 0A1$

$\to 00A11$ [A→0A]

$\to 001A1$ [A→1A]

$\to 0010A1$ [A→0A]

$\to 00100\epsilon 1$ [A→ε]

$\to 001001$

Q4. Construct a CFG for string having different first and last symbol over $\Sigma = \{0, 1\}$

Ans: $L = \{aa, aab, bbaab, \ldots\}$

Sol^n:

R.E: $= \underset{AQ}{(a+b)^*} a(a+b)^* a(a+b)^*$

AaAaA

A | ε | aA | bA

$S \to AaAaA$

$A \to aA | bA | \epsilon$

So, the grammar is:

$G = \{V, \Sigma, P, S\}$

where

$V = \{S, A\}$

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

$S = \{S\}$

$P = \{S \to AaAaA, A \to aA/bA/\epsilon\}$

now we derive 'ababab' from grammar as:

$S \to AaAaA$

$\to abaAaA$ [A→ε, A→aA]

$\to ababAaA$ [A→bA]

$\to ababaA$ [A→ε]

$\to ababab A$ [A→bA]

$\to ababab$ [A→ε]

Multi-tape Turing Machine

Q: Construct CFG for string for $L = \{a^n b^n \mid n \geq 1\}$

Soln: $L = \{ab, aabb, aaabbb, \ldots\}$

R.E = $a^n b^n \mid n \geq 1$

R.E = $(a+b)^+ \cdot (a+b)^+$

$S \to aS/\epsilon$

$S \to bS/\epsilon$

$A = (a+b)^+$ RE: $a^+ \cdot b^+$

$B = (b+a)^+$

$A \to aA/a$

$B \to bB/b$

$S \to AB$

Grammar: $G = \{V, \Sigma, P, S\}$

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

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

$P = \{A \to aA/a,\ B \to bB/b,\ S \to AB\}$

$S = \{S\}$

now, we derive 'aaabbb' from grammaras:

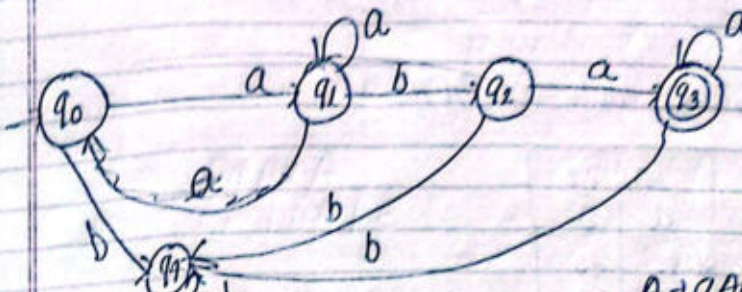
$S \to AB$
$\to aAB$ [A→aA]
$\to aaAB$ [A→aA]
$\to aaaB$ [a→a]
$\to aaabB$ [B→bB]
$\to aaabbB$ [b→bB]
$\Rightarrow aaabbb$ [b→b]

2074 Chit test — SetA

1.a) Construct FA that accepts $L = \{a^m b a^n : m, n > 0\}$

Soln: $L = \{a^m b a^n : m, n > 0\}$

$= \{aba, aba^2, a^2ba \ldots\}$



A → aA/a
B → bB/b
S → AB

transition function: δ:

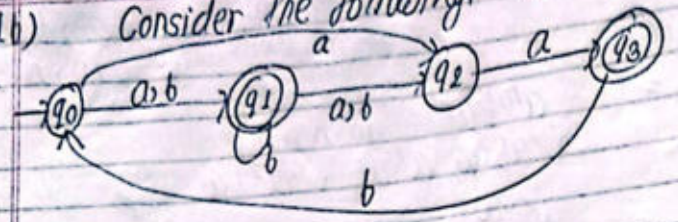
state	input a	input b
→ q0	q1	q4
q1	q1	q2
q2	q3	q4
* q3	q3	q4
q4	q4	q4

→ now processing for aabaaba

$(q_0, aabaaba) \vdash (q_1, abaaba)$
$\vdash (q_1, baaba)$
$\vdash (q_2, aaba)$
$\vdash (q_3, aba)$
$\vdash (q_3, ba)$
$\vdash (q_3, a)$
$\vdash q_3$

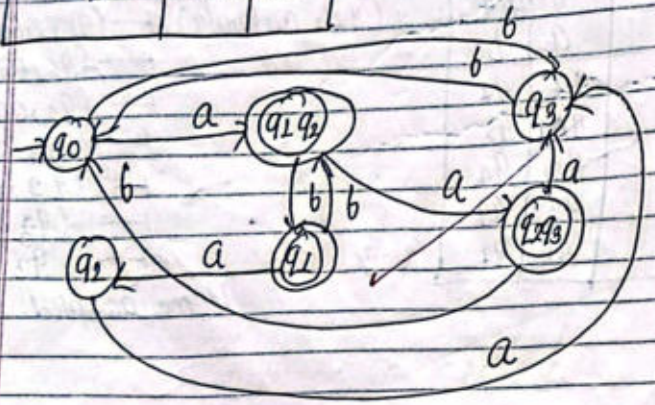
Hence, accepted.

1b) Consider the following NFA to DFA.



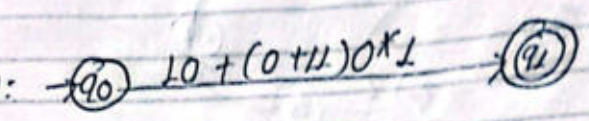
Sol^n: state table of DFA:

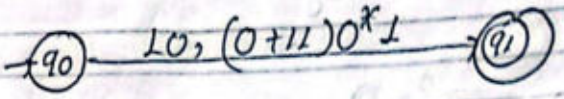
state	input a	input b
→ q_0	$\{q_1 q_2\}$	q_3
* $\{q_1 q_2\}$	$\{q_2 q_3\}$	$\{q_1$
$q_2 q_3$	$\emptyset$	q_0
* $\{q_1 q_3\}$	q_3	q_0
* q_1	q_2	$\{q_1 q_2\}$
q_2	q_3	$\emptyset$



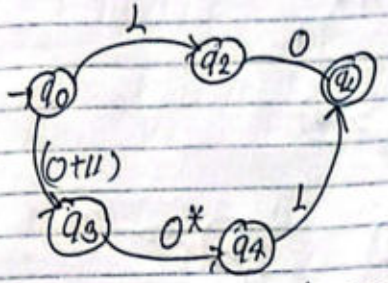
SCLB

1a) Construct a FA equivalent to $10 + (0+11)0^*1$.

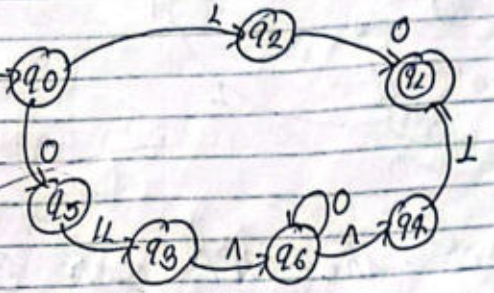
Sol^n: step1: 

step2: 

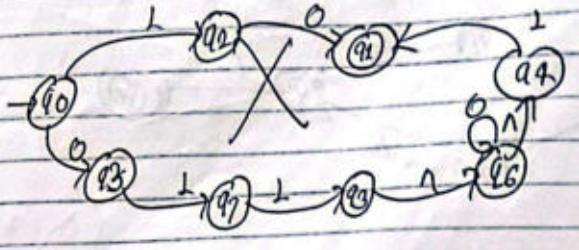
step3:



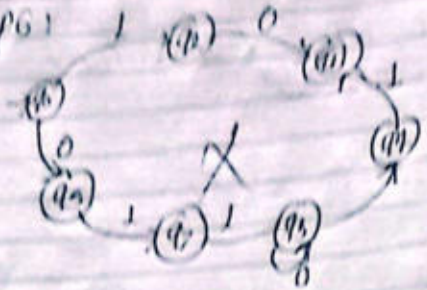
step4:



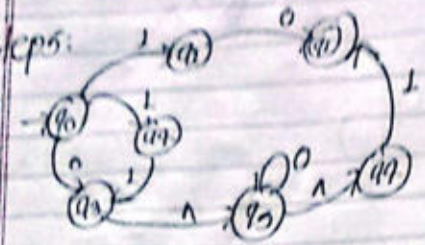
step5:



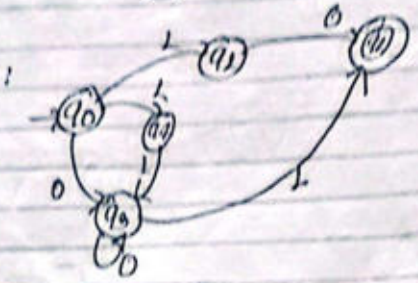
Steps:



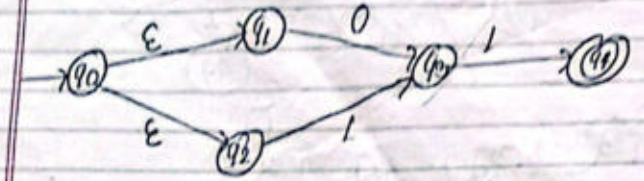
Steps:



Steps:



16) Convert the NFA with Epilson to DFA.



Soln: Closure of states:

ε-closure (q_0) = {q_0, q_1, q_2}
" (q_1) = {q_1, q_2}
" (q_2) = {q_2}
" (q_3) = {q_3}
" (q_4) = {q_4}

Now, DFA transition function: δ':

i) δ'({$q_0q_1q_2$}, 0) = ε-closure(δ(q0,0) ∪ δ(q1,0) ∪ δ(q2,0))
= ε-closure (∅ ∪ q_3 ∪ ∅)
= ε-closure (q_3) = {q_3} //new

similarly, ii) δ'({$q_0q_1q_2$}, 1)
= ε-closure(∅ ∪ ∅ ∪ q_3)
= ε-closure (q_3) = {q_3} //old.

iii) δ'(q_3, 0)
ε-closure(∅) = ∅

iv) δ'(q_3, 1)
= ε-closure(q_4)
= {q_4} //new

v) δ'(q_4, 0)
= ∅

vi) δ'(q_4, 1)
= ε-closure(q_4)
= q_4 //old.

c) Many to many:
- many domains have many ranges

d) Onto function (Surjective)

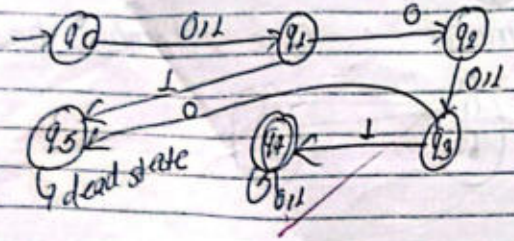
range = codomain

e) One to one Onto (bijective)

Set C

1c) Design DFA, $L = \{w \in (0,1)^*$ / second symbol is '0' and fourth symbol is 1. substring: 0001

Soln:

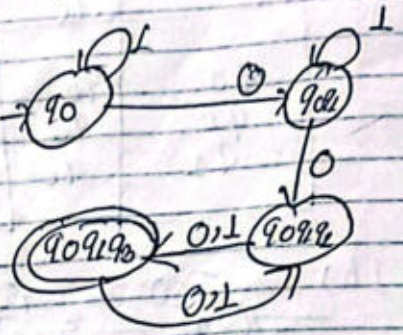


Transition table:

state	input	
	0	1
q_0	q_1	q_1
q_1	q_2	q_5
q_2	q_3	q_3
q_3	q_4	q_5
q_4	q_4	q_4
q_5	q_5	q_5

1.(1) Convert the NFA $M = (\{q_0, q_1, q_2, q_3\}, \{0,1\}, \delta, q_0, \{q_3\})$ to it's equivalent DFA. δ is given by:

states	0	1
q_0	q_0, q_1	q_0
q_1	q_2	q_1
q_2	q_3	q_3
q_3	-	q_2



Soln: state table for DFA:

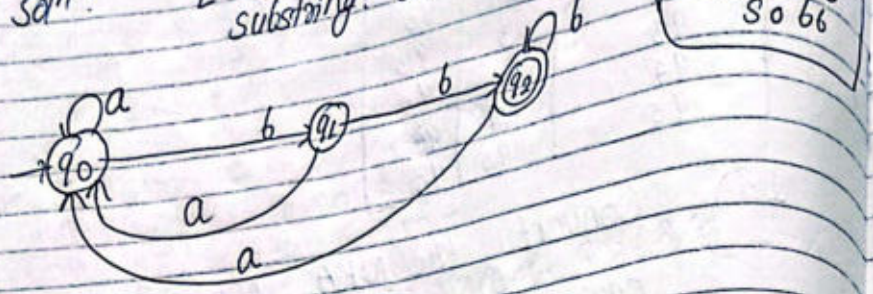
state	0	1
q_0	$\{q_0 q_1\}$	q_0
$\{q_0 q_1\}$	$\{q_0 q_1 q_2\}$	$\{q_0 q_1\}$
$\{q_0 q_1 q_2\}$	$\{q_0 q_1 q_3\}$	$\{q_0 q_1 q_3\}$
$\{q_0 q_1 q_3\}$	$\{q_0 q_1 q_2\}$	$\{q_0 q_1 q_2\}$

2021 Spring Recurrence

1a) DFA for language: $L = \{\omega \in (a,b)^* : \omega \text{ ends with } bb\}$

Soln: $L = \{bb, abb, abbb \ldots\}$

substring: bb

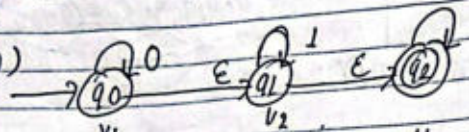


q0: a ∉ again q0 so bb

δ:

state	input	
	a	b
q_0	q_0	q_1
q_1	q_0	q_2
q_2	q_0	q_2

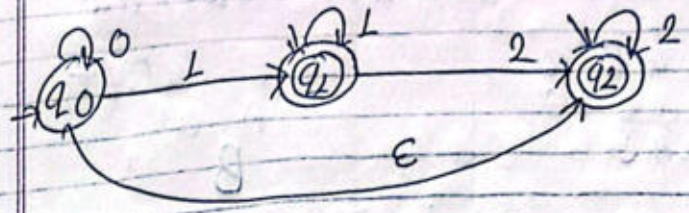
1b)



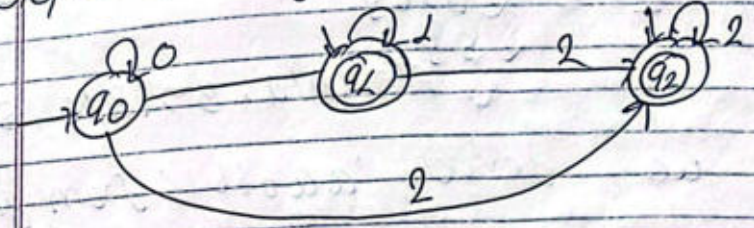
Convert NFA with ε transition into NFA without ε-transition.

Soln: step1: let removing ε between q_0 and q_1.

step2: removing ε between q_1 and q_2.



step3: removing ε between q_0 and q_2.



Q. Construct CFG for $L = \{a^n b^m / n \geq 1, m \geq 0\}$

Soln: $L = \{a, ab, abb, aab, aa \ldots\}$

$A \to aA/a$
$B \to bB/\epsilon$
$S \to AB$

$R.E = a^+.b^*$
$A = a^+$
$B = b^*$

Grammar: $G = \{V, \Sigma, P, S\}$
$V = \{A, B\}$
$\Sigma = \{a, b\}$
$S = \{S\}$
$P = \{A \to aA/a, B \to bB/\epsilon, S \to AB\}$

Now, we derive 'aabbb' from grammar

$S \to AB$
$\to aAB$ [A→aA]
$\to aaB$ [A→a]
$\to aabB$ [B→bB]
$\to aabbB$ [B→bB]
$\to aabbbB$ [B→bB]
$\to aabbb\epsilon$ [B→ε]
$\Rightarrow aabbb$

$(a+b)^+ \Rightarrow A \to aA/a, B \to bB/b$

Derivation of CFG:

- process of deriving strings from grammar using production rule.
- begin with start symbol and replace variable / nonterminal symbols.

Eg: Derive a^4 from grammar.

$S \to aS$
$S \to \epsilon$

Soln:
$S \to aS$
$\to aaS$ [S→aS]
$\to aaaS$ [S→aS]
$\to aaaaS$ [S→aS]
$\to aaaa\epsilon$ [S→ε]
$\to aaaa$

Hence, given string a^4 is accepted.

From grammar:

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

$\therefore L = \{a^n / n \geq 0\}$

Eg: Let, $G = \{(S, \epsilon), (a, b), P, S\}$

where, P:
$S \to aCa$
$C \to aCa/b$

find $L(G)$ i.e. CFL.
(all string accepted by G)

Q. Construct CFG for $L = \{a^n b^m \mid n \geq 1, m \geq 1\}$

Sol^n: $L = \{a, ab, abb, aab, aa\}$

$A \rightarrow aA/a$
$B \rightarrow bB/\epsilon$
$S \rightarrow AB$

$P.E = a^+.b^+$
$A = a^+$
$B = b^+$

CFG Grammar: $G = \{V, \Sigma, P, S\}$

$V = \{A, B\}$
$\Sigma = \{a, b\}$
$S = \{S\}$
$P = \{A \rightarrow aA/a, B \rightarrow bB/\epsilon\}$

now, we derive 'aabbb' from

$S \rightarrow AB$
$\rightarrow aAB$ [$A \rightarrow aA$]
$\rightarrow aaB$ [$A \rightarrow a$]
$\rightarrow aabB$ [$B \rightarrow bB$]
$\rightarrow aabbB$ [$B \rightarrow bB$]
$\rightarrow aabbbB$ [$B \rightarrow bB$]
$\rightarrow aabbb\epsilon$ [$B \rightarrow \epsilon$]
$\Rightarrow aabbb$

$(a+b)^+ \Rightarrow A \rightarrow aA/a$
$B \rightarrow bB/b$

Q. Construct CFG for $L = \{a^n b^m / n \geq 1, m \geq 0\}$

Soln: $L = \{a, ab, abb, aab, aa \ldots\}$

$A \to aA/a$
$B \to bB/\epsilon$
$S \to AB$

$R.E = a^+.b^*$
$A = a^+$
$B = b^*$

Grammar: $G = \{V, \Sigma, P, S\}$
$V = \{A, B\}$
$\Sigma = \{a, b\}$
$S = \{S\}$
$P = \{A \to aA/a, B \to bB/\epsilon, S \to AB\}$

now, we derive 'aabbb' from grammar

$S \to AB$
$\to aAB$ [a→aA]
$\to aaB$ [a→a]
$\to aabB$ [B→bB]
$\to aabbB$ [B→bB]
$\to aabbbB$ [B→bB]
$\to aabbb\epsilon$ [B→ε]
$\to aabbb$

$(a+b)^+ \to A \to aA/a$
$B \to bB/b$

Derivation of CFG:

- process of deriving strings from grammar using production rule.
- begin with start symbol and replace variable / nonterminal symbols.

Eg: Derive aA from grammar.

$S \to aS$
$S \to \epsilon$

Soln:
$S \to aS$
$\to aaS$ [S→aS]
$\to aaaS$ [S→aS]
$\to aaaaS$ [S→aS]
$\to aaaa\epsilon$ [S→ε]
$\to aaaa$

Hence, given string a^4 is accepted.

From grammar:

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

$\therefore L = \{a^n / n \geq 0\}$

Eg: Let, $G = \{(S, \epsilon), (a, b), P, S\}$

where, P:
$S \to aCa$
$C \to aCa/b$

find L(G) i.e. CFL.
(all string accepted by G)

Sol^n: (i) $S \to aCa$
$\to aba$ [$C \to aba$]

$\therefore aba \in L(G)$

(ii) $S \to aCa$
$\to aaCaa$ [$C \to aCa$]
$\to aabaa$ [$C \to b$]
$\therefore aabaa \in L(G)$.

similarly:
(iii) $aaabaaa \in L(G)$
(iv) $aaaabaaaa \in L(G)$
and so on.

This, language generated by grammar is:
$L = \{a^n b a^n \ [n \geq 1\}$

Example: Find $L(G)$
$S \to aS \mid a$
$S \to bS \mid b$

Sol^n: (i) $S \to a$
(ii) $S \to aS$
$\to aa$ [$S \to aS$]

(iii) $S \to aS$
$\to aaS$ [$S \to aS$]
$\to aaa$ [$S \to a$] and so on.

(iv) $S \to b$
$S \to bS$
$S \to bb$ [$b \to bS$]
and so on

(v) $S \to aS$
$\to abS$ [$S \to bS$]
$\to abaS$ [$S \to aS$]
$= abab$ [$S \to b$]

$L = \{a, b, aa, bb, ab, ba, aaba, bbaa \ldots\}$

$L = \{(a+b)^+\}$ #

Example3: Find $L(G)$
$S \to XaaX$
$X \to aX \mid bX \mid \varepsilon$

Sol^n: (i) $S \to XaaX$
$\to aa$ [$X \to \varepsilon$]

(ii) $S \to XaaX$
$\to aXaa\varepsilon$ ($X \to aX, X \to \varepsilon$)
$\to aaa$ ($X \to \varepsilon$)

(iii) $S \to XaaX$
$\to bXaa\varepsilon$ ($X \to bX, X \to \varepsilon$)
$= baa$ ($X \to \varepsilon$)

iv) S → XaaX [X → bX]
→ baabX [X → ε]
→ baab

similarly,

v) S → aaab
→ baaa
and so on.

L = {aa, aaa, aaaa, baab, baabb, baa ...}

∴ L = (a+b)* aa (a+b)*

Example: Derive 'abbaba' from the grammar.

S → XaaX
X → aX | bX | ε

where, X = (a+b)*

Soln:
S → XaaX
→ aXaabX [X → aX, X → bX]
→ abXaabaX [X → bX, X → aX]
⇒ abbXaaba [b → bX X → ε]
→ abbεaaba [X → ε]
→ abbaaba

Q. If G is a grammar

G = (V, Σ, P, S)
V = {S, A, B}
Σ = {a, b}
S = {S}

and production rule: P is:

S → AaB
A → baA | a
B → abB | b

Check whether the following string is accepted by the grammar or not.

i) baaab
ii) aab
iii) baaabb

Soln: i) baaab
⇒ S → AaB
→ baAaB [A → baA]
→ baaaB [A → a]
→ baaab [B → b]
Hence, baab is accepted.

ii) aab
S → AaB
→ aaB [A → a]
→ aab [B → b]

Multi-tape Turing

iii) baaabb
→ S → AaB
→ baAaB [A→bAa]
→ baaaB [A→a]
→ baaab [B→b]

Hence, baaabb is not accepted.

Derivation of CFG

Types:

a) left-most derivation:
apply product rule at left most variable every time.

b) Right-most derivation:
apply product rule at right most variable every time.

c) Mixed derivation:
combination of both left-most derivation & right-most derivation.

Derivation Tree (Parse Tree)
→ derivation of CFG can be represented as tree.

Defination: A derivation tree for CFG, G = {V, Σ, P, S} is a tree satisfying following condition:

(i) every vertex has a label which is a variable or terminal or ε.

(ii) The root has label S.

(iii) The label of an internal vertex is a variable.

(iv) If the vertices $n_1, n_2, \ldots n_k$ written with labels $X_1, X_2, \ldots X_k$ are the children of vertex 'n' with label A, then $A \to X_1 X_2 \ldots X_k$ is a production rule.

For example:

$S \to bSb \mid a \mid b$

derivation of

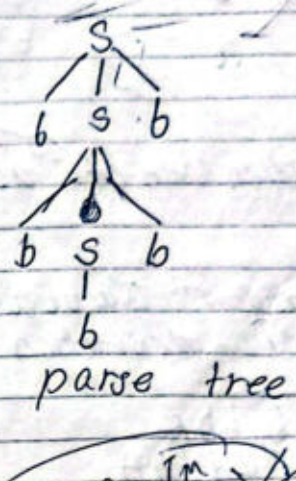


fig: parse tree

PDA, TM

Q. Draw a derivation tree for the string "abaaba" from the grammar G. Where P: { S → aSa, S → bSb, S → a|b|ε }

Sol": using left most derivation.

S → aSa
→ abSba [S → bSb]
→ abaSaba [S → aSa]
→ abaεaba [S → ε]
→ abaaba

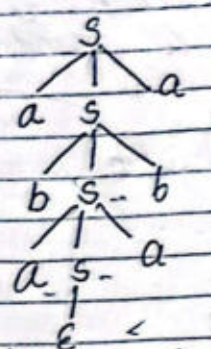


fig: derivation tree of "abaaba".

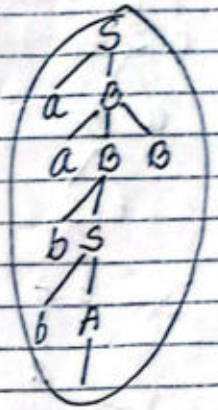
Q. Derive the string "aabbabba" for left most and right most derivation using a given grammar.

S → aB | bA
A → a | aS | bAA
B → b | bS | aBB

aabbabba

Sol": using left most derivation

S → aB
→ aaBB [B → aBB]
→ aabSB [B → bS]
→ aabbAB [S → bA]
→ aabbaB [A → a]
→ aabbabS [B → bS]
→ aabbabbA [S → bA]
→ aabbabba [A → a]



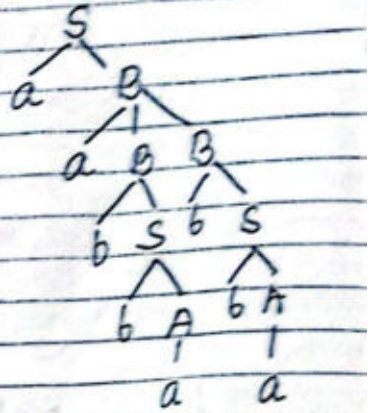


fig: derivation tree of aabbabba.

using right most derivation:

S → aB
→ aaBB [B → aBB]
→ aaBbS [B → bS]
→ aaBbaB [S → aB]
→ aaBbabS [B → bS]
→ aaBbabbA [S → bA]

using

→ aaBbabba [A → a]
→ aabbabba [B → b]

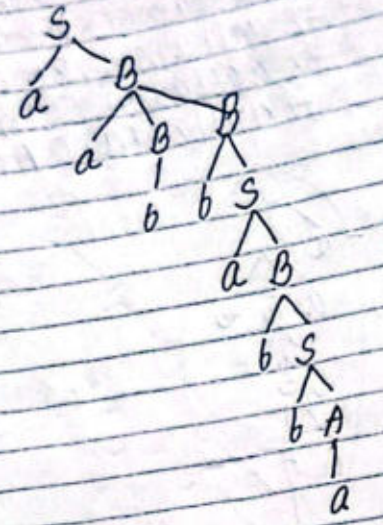


fig: right most derivation tree of "aabbabba".

Q. Let G be grammar
S → 0B | 1A
A → 0 | 0S | 1AA
B → 1 | 1S | 0BB
find string 00110101 by
(i) left most derivation
(ii) right " "
(iii) mixed derivation

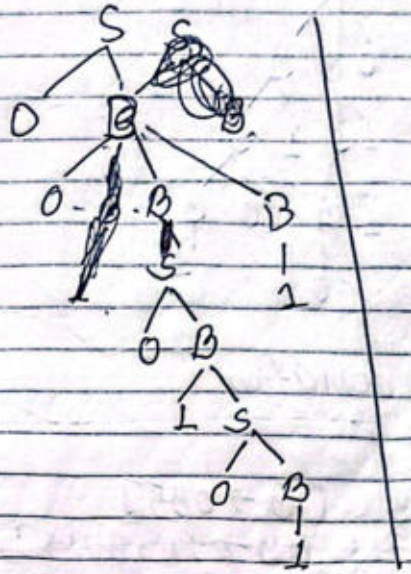
(iv) make parse tree of each derivation.

Soln: "00110101"

(i) using left most derivation:

S → 0B
→ 00BB [B → 0BB]
→ 001B [B → 1]
→ 0011S [B → S]
→ 00110B [S → 0B]
→ 001101S [B → 1S]
→ 0011010B [S → 0B]
→ 00110101 [B → 1]

OR



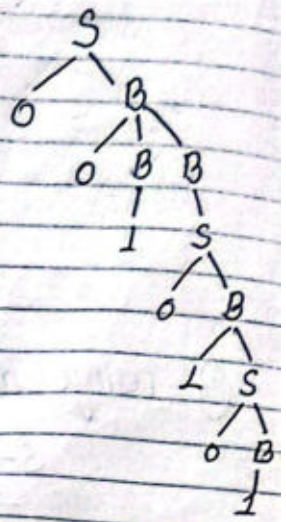
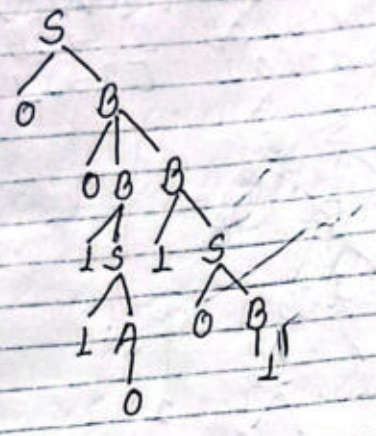


fig: left most derivation tree of "00110101".

$S \to 0B \mid 1A$
$A \to 0 \mid 0S \mid 1AA$ $B \to 1 \mid 1S \mid 0BB$

(i) using right most derivation

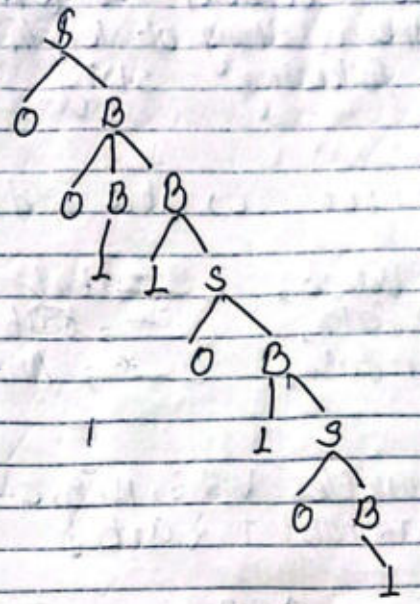
$S \to 0B$
$\to 00BB$ [B → 0BB]
$\to 00B1S$ [B → 1S]
$\to 00B10B$ [S → 0B]
$\to 00B101$ [B → 1]
$\to 001S101$ [B → 1S]
$\to 0011A101$ [S → 1A]
$\to 00110101$ [A → 0]



(ii) using mixed derivation

$S \to 0B$
$\to 00BB$ [B → 0BB]
$\to$ ~~00 1A S~~ ~~[B → 1A, B → 1S]~~
$\to$ ~~00 1AA 0B~~ ~~[A → 1AA, S → 0B]~~
$\to$ ~~00 1010 1AA 01~~ ~~[B → 1, A → 0, A → 1AA]~~
$\to$

$S \to 0B$
$\to 00BB$ [B → 0BB]
$\to 001S$ [B → 1, B → 1S]
$\to 0011 0B$ [S → 0B]
$\to 0011 01S$ [B → 1S]
$\to 0011010B$ [S → 0B]
$\to 00110101$ [B → 1]



(imp)

Q. Consider a context free for following production.

$S \rightarrow ASA \mid B$
$B \rightarrow aCb \mid bCa$
$C \rightarrow ACA \mid A$
$A \rightarrow a \mid b$

Q(i) What are variables & terminals in G?
Q(ii) Give the string of length '7' in L(G).
Q(iii) Are the following string in language i.e. L(G)?

a) aaa b) bbb c) aba d) abb

Q(iv) True or false, $C \rightarrow bab$
Q(v) True or false, $C \not\rightarrow bab$
Q(vi) True or false, $C \not\rightarrow AAA$

Sol^n: i) Variables: $V = \{S, A, B, C\}$
terminals: $T = \{a, b\}$

ii) $S \rightarrow ASA$
$\rightarrow aBa$ $(A \rightarrow a, S \rightarrow B)$
$\rightarrow aaCba$ $[B \rightarrow aCb]$
$\rightarrow aaACAba$ $[C \rightarrow ACA]$
$\rightarrow aaaAAaba$ $[C \rightarrow A, a \rightarrow a]$
$\rightarrow aaababa$ $[A \rightarrow b]$

iii) a) aaa
$S \rightarrow ASA$
$\rightarrow aSa$ $[A \rightarrow a]$
$\rightarrow aASAa$ $[A \rightarrow ASA]$
$\rightarrow aasaa$ $[A \rightarrow a]$

So, $aaa \notin L(G)$

b) bbb
$S \rightarrow ASA$
$\rightarrow bSb$ $[A \rightarrow b]$
$\rightarrow bBb$ $[S \rightarrow B]$
$\rightarrow bbCab$ $[B \rightarrow bCa]$

So, $bbb \notin L(G)$

$S \rightarrow B$
$\rightarrow aCb$

c) aba
$S \rightarrow B$
$\rightarrow aCb$
$S \rightarrow ASA$
$\rightarrow aSa$ $[A \rightarrow a]$
$\rightarrow abCa$ $[S \rightarrow bCa]$
$\rightarrow abCaa$ $[B \rightarrow bCa]$

$\therefore aba \notin L(G)$

from both: $S \rightarrow ASA$ and $S \rightarrow B$ 1 starting symbol, 'aba' doesnot derive. So, $aba \notin L(G)$.

d) abb

$S \to ASA$ [A→a, A→b]
$\to asb$ [A→B]
$\to aBb$ [B→bCa]
$\to abCab$

∴ abb ∉ L(G)

$S \to B$
$\to aCb$
$\to aAb$
$\to abb$

∴ abb ∈ L(a)

(iii) C → bab

S → ASA [A→b, A→b]
→ bSb
→ bBb [B→bCa]
→ bbCab [C→A]
→ bbb

(iv) C → bab

Soln: C → ACA [A→b]
→ bCb [C→A]
→ bAb [A→a]
→ bab

as C → bab takes multiple step.
(False)

v) C ⇒ bab

as C takes multiple step.

S → True.

problem - unsolvable

Student at 4 of them are born in so

n = 12
= K+1
= 4
K = 3

a^{K+1} = 3^{3+1} = 3^4

vi) C ⇒ AAA

Soln: C → A·CA
→ AAA [C→A]

as C takes single step. So

C ⇒ AAA is true.

R = {(a,a), (a,c), (b,c), (c,d), (d,b)}

R	a	b	c	d
Rac a	1	0	1	0
Rbc b	0	0	1	0
Rcc c	0	0	0	1
Rdd d	0	1	0	0

D = abcd = 1000

D' = 0111 = {b,c,d}

imp-

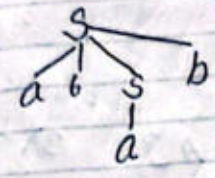
Ambiguity in CFG

$w \in L(G)$ is ambiguous if there exist two or more than two derivation tree.

Eg: Show that the grammar

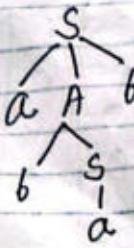
$S \to a \mid abSb \mid aAb$
$A \to bS \mid aAAb$ is ambiguous.

Soln: i) $S \to abSb$
$\to ababb$ [S→a]



and ii)

$S \to aAb$
$\to abSb$ [A→bS]
$\to ababb$ [S→a]

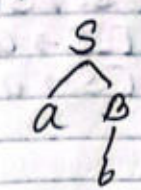


since, string 'abab' has two different parsetree.
so, given grammar is ambiguous.

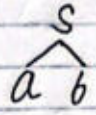
Q. show the grammar: $w \in L(G)$

$S \to aB \mid ab$
$B \to ABb \mid b$
$A \to aAB \mid a$ is ambiguous.

Soln: i) $S \to aB$
$\to ab$ [B→b]



ii) $S \to ab$



since, string 'ab' has two different parse tree. so, given grammar is ambiguous.

Q. Show that the grammar

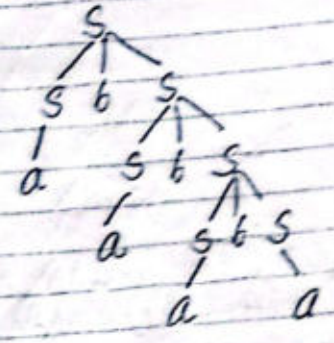
$S \to SbS \mid a$

is ambiguous for the string 'abababa'.

Soln:

1st

$S \to SbS$ [S → a]
$\to abS$ [S → SbS]
$\to abSbS$ [S → a]
$\to ababS$ [S → SbS]
$\to ababSba$ [S → a]
$\to abababa$ [S → a]



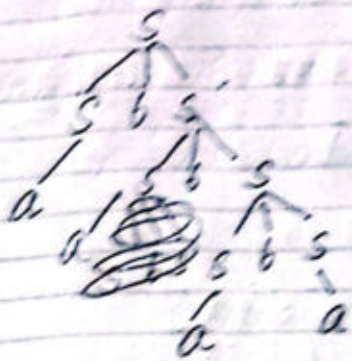
leftmost

2nd

parse tree

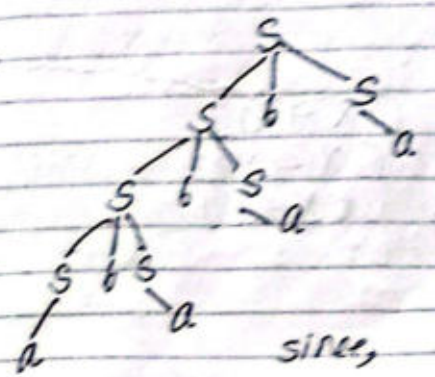
$S \to SbS$
$\to SbSbS$ [S → SbS]
$\to SbabSbS$ [S → a, S → SbS]
$\to SbababS$ [S → a, S → a]
$\to abababa$ [S → a]

Note: We leftmost and rightmost



rightmost

$S \to SbS$
$\to Sba$ [S → a]
$\to SbSba$ [S → SbS]
$\to Sbaba$ [S → a]
$\to SbSbaba$ [S → SbS]
$\to Sbababa$ [S → a]
$\to abababa$ [S → a]



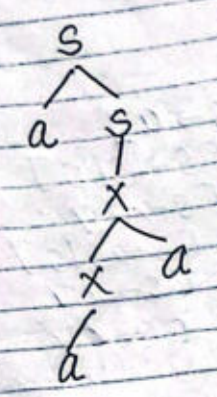
since, string 'abababa' has two different parse tree. So it is ambiguous.

Assignment

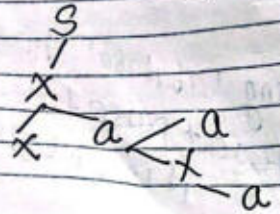
→ Show that the grammar is ambiguous.

a) $S \to aS / aSb / X$
$X \to Xa / a$

Sol^n: i) $S \to aS$
$\to aX$ [S→X]
$\to aXa$ [X→Xa]
$\to aaa$ [X→a]



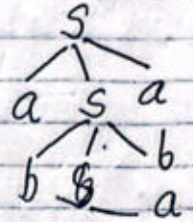
ii) $S \to X$
$\to Xa$ [X→Xa]
$\to Xaa$ [X→Xa]
$\to aaa$ [X→a]



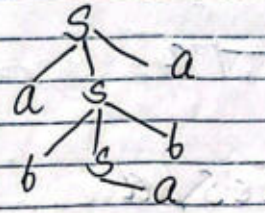
since, string 'aaa' has two different parse tree, so the grammar is ambiguous

b) $S \to aSa / bSb / a / b / \epsilon$

Sol^n: i) $S \to aSa$
$\to abSba$ [S→bSb]
$\to ababa$ [S→a]

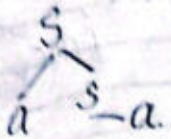


ii) $S \to aSa$
$\to abSba$ [S→bSb]
$\to ababa$ [S→a]



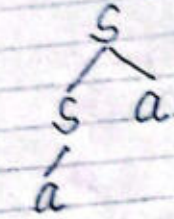
since, string 'abab' has similar parse tree, so the grammar is not ambiguous.

c) $S \to aS \mid Sa \mid a$



i) $S \to aS$
$\to aa$ [S→a]

ii) $S \to Sa$
$\to aa$ [S→a]



since, string 'aa' has two different parse tree. So, the grammar is ambiguous.

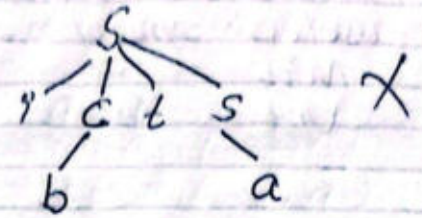
d) ~~$S \to iCtS \mid iCtSeS \mid a$~~
~~$C \to$~~

$S \to iCtS \mid iCtSeS \mid a$
$C \to b$

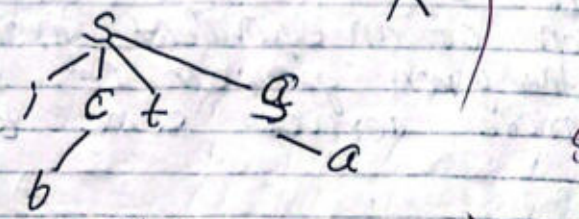
Soln: i) $S \to iCtS$
$\to ibtS$ [C→b]
$\to ibta$ [S→a]

ii) $S \to iCtS$
$\to ictS$
$\to ictS$
$\to ictS$
$\to ibta$ [C→b] X

i) parse tree:



ii) Parse tree:

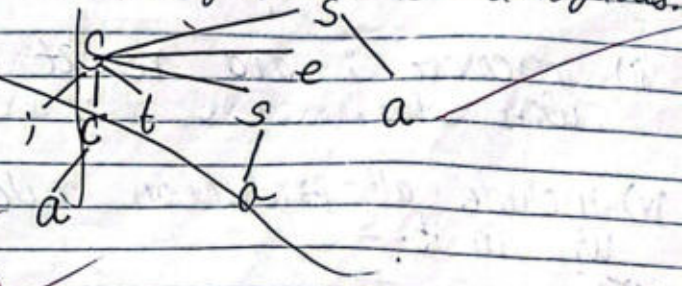


Soln: $S \to iCtSeS$
$\to iatSeS$ [C→a]
$\to iataeS$ [S→a]
$\to iataea$ [S→a]

So, the grammar is ambiguous.

See parsing part copy for solution of this.

parse tree:



Checked
02/12

Simplification of Grammar:

→ Removing all unnecessary symbols.

(i) Removal of useless symbol
(ii) " " NULL "
(iii) " " Unit "

→ (i) Removal of useless symbol: (Reduction of CFG)

a) Phase1: (Wi+1 or Wi and derive ∑T*)
Derivation of an equivalent grammar G' from the given grammar G, such that each variable derives some terminal string.

Steps:
i) include all symbols Wi, that derives some terminal and initialize i = 1.
ii) include symbol Wi+1, that derives Wi.
iii) increment i and repeat step (ii) until Wi+1 = Wi.
iv) include all production rule that have Wi in it.
v)

b) Phase2: (Wi+1, Wi or derive ∑T*)
→ Derivation of an equivalent grammar G' from the CFG, G' such that each symbol appears in a sentential form, (i.e. derived from start symbol).

Steps:
i) include the start symbol in Yi and initialize i = 1.
ii) include all symbol Yi+1 that can be derived from Yi and include all production rules that have been applied.
iii) increment i and repeat step (ii) until Yi+1 = Yi

Example: 1. Find a reduced grammar equivalent to grammar G having production rule P as:

S → AC | B, A → a, C → c | BC, E → aA/e

Soln: Phase1:
T = {a, c, e}
W1 = {A, C, E}

$G = \{V, \Sigma, P, S\}$

$W_0 = \{A, C, E, S\}$

$W_3 = \{A, C, E, S\}$

$G' = (\{A, C, E, S\}, \{a, c, e\}, P, S)$

$P: S \to AC, \quad A \to a, \quad C \to c, \quad E \to aA \mid e$

phase2:

$Y_1 = \{S\}$

$Y_2 = \{S, A, C\}$ ↺ Y2

$Y_3 = \{S, A, C, a, c\}$

$Y_4 = \{S, A, E, a, c\}$

$G'' = (\{A, C, S\}, \{a, c\}, P, S)$

$P: S \to AC, A \to a, C \to c$

So, this is reduced grammar required

example:

$S \to ABa \mid BC, \quad A \to aC \mid BCC,$

$C \to a$

$B \to bcc$

$D \to E$

$E \to d$

$F \to e$

Soln: phase1:

$T = \{a, b, c, d, e\}$

$W_1 = \{S, A, B, C, E, F\}$

$W_2 = \{S, A, B, C, E, F\}, D\}$

$W_3 = \{S, A, B, C, D, E, F\}$

~~$W_3 = \{S, A, C, E\}$~~

$G' = (\{S, A, B, C, E, F\}, \{a, b, c, d, e\}, P, S)$

$P: \quad S \to ABa \mid BC \qquad D \to E$

$A \to aC \mid BCC$

$B \to bcc$

$E \to d$

$F \to e$

phase2:

$Y_1 = \{S\}$

$Y_2 = \{S, A, B, C\}$

$Y_3 = \{S, A, B, E, a, c, b\}$

$Y_4 = \{S, A, B, C, a, b, c\}$

$G'' = (\{S, A, B, C\}, \{a, b, c\}, P, S\}$

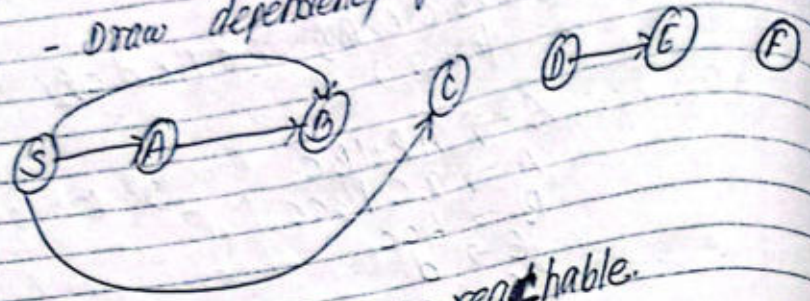
$P: S \to ABa \mid BC, A \to aC \mid BCC,$

$C \to a, B \to bcc$

Alternate method:

step1: eliminate non-generating symbols.
Hence All variables are generating symbols.

step2: elimination of non-reachable symbols.
- Draw dependency graph



Here, D, E, F are non-reachable.

After removing useless symbol, we get grammar G as:

$G = \{\{A, B, C, S\}, \{a, b, c\}, P, S\}$

P: $S \to ABa \mid BC$, $A \to aC \mid BCC$,
$C \to a$, $B \to bcc$

Assignment:

Date:
Page:

examples: Q1. $S \to AB \mid CA$, $S \to BC \mid AB$, $A \to a$
$C \to aB \mid b$

Soln: phase i:
$T = \{a, b\}$
$W_1 = \{A, C\}$ ($\to$ AC तर्फ derive गर्दा वाला)
$W_2 = \{S, A, C\} = \{S, A, C\}$
$W_3 = \{S, A, C\}$

$G' = (\{A, B, C\}, \{a, b\}, P, S\}$

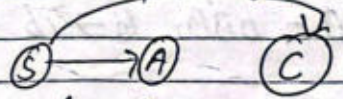
P: $A \to a$, $C \to aB \mid b$, $S \to BC \mid CA$,
~~$S \to BC \mid AB$~~

phase ii: $Y_1 = \{S\}$ $\to$ S बाट derive गर्दा
$Y_2 = \{S, A, B, C\}$
$Y_3 = \{S, A, a, B, b\}$
$Y_4 = \{S, A, a, B, b\}$

$G'' = \{\{S, A, B, C\}, \{a, b, c\}, P, S)$
P: $S \to AB \mid CA \mid BC$ CA
$A \to a$
$C \to aB \mid b$

alternate: i) eliminate B as it is non-generating symb

ii) Dependency graph



$\therefore G = (\{(S, A, C), (a, b), P, S\}$
P: $S \to CA$
$A \to a$
$C \to b$

Q2. $S \to aAa$, $A \to bBB$, $B \to ab$, $C \to ab$

Soln: phase i):

$T = \{a, b\}$

$W_1 = \{S, A, B, C\}$

$W_2 = \{S, A, B, C\}$

$\therefore G' = \{(S, A, B, C), (a, b), P, S\}$

$P: S \to aAa,\ A \to bBB,\ B \to ab,\ C \to ab$

phase ii):

$Y_1 = \{S\}$

$Y_2 = \{a, A\}$

$Y_3 = \{a, b, B\}$

$Y_4 = \{a, b, B\}$

$\therefore G'' = \{(S, A, B), (a, b), P, S\}$

$\therefore P: S \to aAa,\ A \to bBB,\ B \to ab$

alternate:

i) as all are generating symbols.

ii) dependency graph:

(S) — (A) — (B) (C)

Here, C is not reachable.

So,

$G = \{(S, A, B), (a, b), P, S\}$

$\therefore P: S \to aAa,\ A \to bBB,\ B \to ab$

Q3. $S \to aS \mid A \mid BC$

$A \to a$

$B \to aa$

$C \to aCb$

[use both method]

Soln: alternate:

Here, C is useless as it doesnot derive any string.

$S \to aS \mid A$

$A \to a$

$B \to aa$

Also, B is not reachable from start symbol.

$S \to aS \mid A$

$A \to a$

phase i): $T = \{a, b\}$

$W_1 = \{S, B, A, C\}$

$W_2 = \{S, A, B, C\}$

$\therefore G = \{(S, A, B, C), (a, b), P, S\}$

$P: S \to aS \mid A$

$A \to a,\ B \to aa,\ C \to aCb$

phase ii)

$Y_1 = \{S\}$

$Y_2 = \{S, a, A, B, C\}$

$Y_3 = \{S, a, A, B, C, b\}$

$Y_4 = \{S, a, A, B, C, b\}$

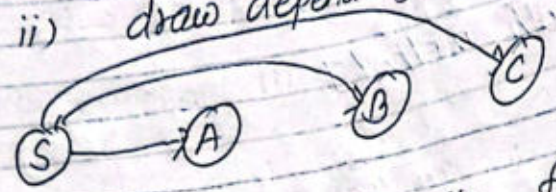
$\therefore G = \{(S, A, B, C), (a, b, c), P, S\}$

$P: S \to aS \mid A,\ A \to a,\ B \to aa,\ C \to aCb$

alternate:

i) these are all generating symbols.

ii) draw dependency graph:



as, A, B, C are reachable from S.

$\therefore G = \{(S, A, B, C), (a, b), P, S\}$

P: $S \to aS \mid A \mid BC$
$A \to a$
$B \to aa$
$C \to aCb$

Removal of Unit Production:

→ Any production rule of the form $A \to B$, where $A, B \in$ Non-terminal is called unit production.

procedure:

step1: to remove $A \to B$ at production $A \to x$ to the grammar whenever $B \to x$ occurs.
($x \in$ Terminal & x can be NULL)

step2: Remove $A \to B$ from grammar.

step3: Repeat above steps until all unit production are removed.

Eg: P: $S \to XY$, $X \to a$, $Y \to Z \mid b$, $Z \to M$
$M \to N$, $N \to a$

Soln: Here, Unit production are:
$Y \to Z$, $Z \to M$, $M \to N$

(i) Since, $N \to a$, add $M \to a$
P: $S \to XY$, $X \to a$, $Y \to Z \mid b$,
$Z \to M$, $M \to a$, $N \to a$

(ii) since, $M \to a$ add $Z \to a$
P: $S \to XY$, $X \to a$, $Y \to Z \mid b$, $Z \to a$,
$M \to a$, $N \to a$

(iii) Since, $Z \to a$, add $Y \to a$

P: $S \to XY$, $X \to a$, $Y \to a|b$, $Z \to a$.
$M \to a$, $N \to a$

Also remove unreachable symbols.

Z, M and N are not reachable by start. So remove them.

P: $S \to XY$
$X \to a$
$Y \to a|b$

examples: P: $S \to A|bb$
$A \to B|b$
$B \to S|a$

Soln: Here, unit production are:
$S \to A$
$A \to B$
$B \to S$

(i) Since, $B \to a$ add $A \to a$

P: $S \to A|bb$
$A \to a|b$
$B \to S|a$

(ii) Since, $S \to A$, add $S \to a|b$

(ii) Since $S \to bb$ add $B \to bb$

P: $S \to A|bb$
$A \to a|b$
$B \to bb|a$

(iii) since, $A \to a|b$ add $S \to a|b|bb$

P: $S \to a|b|bb$
$A \to a|b$
$B \to bb|a$

since, A, B are not reachable from start, so remove A & B.

$\therefore$ P: $S \to a|b|bb$

examples: P: $S \to Aa|B$,
$B \to A|bb$
$A \to a|bc|B$

Soln: Here, unit production are:
$S \to B$, $B \to A$, $A \to B$

(i) since, $A \to a|bc|$, add $B \to a$

$\therefore$ P: $S \to Aa|B$
$B \to a|bc|bb$
$A \to a|bc|B$

(iii) since, B → a|bc|bb add A → a|bc|bb

∵ P: S → Aa|B
B → a|bc|bb
A → a|bc|a|bc|bb

since,

(iv) Since, ~~A → a|bc|a|bc|bb~~, B → a|bc|bb add ~~S → a|bc|a|bc|bb~~ S → a|bc|bb

∵ ~~P: S → a|bc|a|bc|bb~~

P: S → Aa|a|bc|bb
B → a|bc|bb
~~Since~~ A → a|bc|a|bc|bb

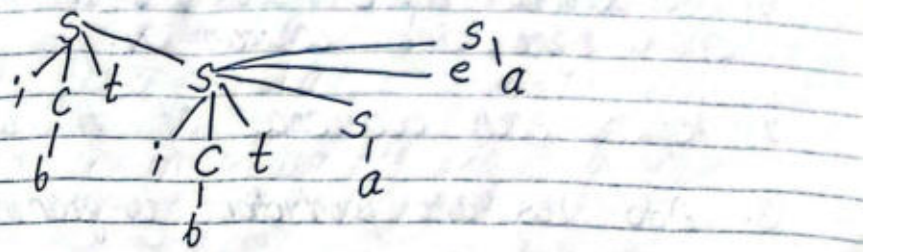
(v) Since, B is not reachable from S. So remove B.

∵ P: S → Aa|a|bc|bb
A → a|bc|a|bc|bb

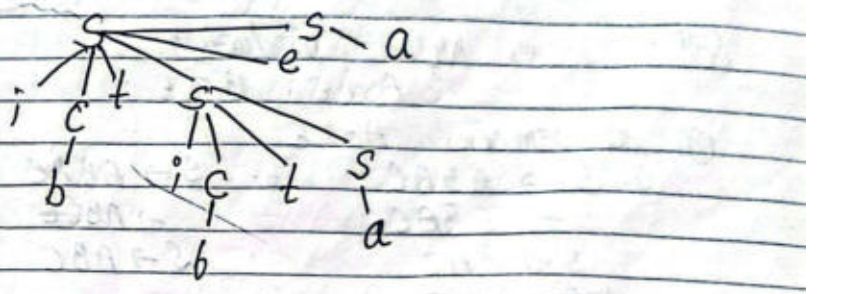
Paselxee assignment

d) S → iCtS | iCtSeS | a
C → b

(i) S → iCtS
→ ibtictSeS [C → b, S → iCtSeS]
→ ibtibtaea [C → b, S → a]



(ii) S → iCtSeS
→ ibtictSeS [C → b, S → iCtS]
→ ibtibtaea [C → b, S → a]



since, the grammar has two parse tree, so it is ambiguous.

Null. / . –

(III) Elimination of NULL moves

In a CFG, a non-terminal 'A' is nullable variable or null production if $A \to \varepsilon$ or there is derivation that start with 'A' and leads to ε.

Procedure:

1. to remove null production $A \to \varepsilon$, look for all production whose right side contain A.
2. replace each occurence of 'A' with ε.
3. Add resultant production to grammar.

Example:

$S \to ABAC$, $A \to aA|\varepsilon$, $B \to bB|\varepsilon$, $C \to c$

Soln: Here, NULL production:

$A \to \varepsilon$, $B \to \varepsilon$

(I) to remove, $A \to \varepsilon$

a. $S \to ABAC$
$\to \varepsilon BAC$
$S \to BAC$

b. $S \to ABAC$
$\to ABEC$
$S \to ABC$

c. $S \to ABAC$
$\to \varepsilon B\varepsilon C$
$S \to BC$

d. $A \to aA$
$\to a\varepsilon$
$\to a$

New grammar (production rule):

$S \to ABAC|BAC|ABC|BC$
$A \to aA|a$
$B \to bB|\varepsilon$
$C \to c$

(II) To remove, $B \to \varepsilon$

a) $S \to ABAC$
$\to A\varepsilon AC$
$\to AAC$

b) $S \to BAC$
$\to \varepsilon AC$
$\to AC$

c) $S \to ABC$
$\to A\varepsilon C$
$\to AC$

d) $S \to BC$
$\to \varepsilon C$
$\to C$

e) $B \to bB$
$\to b\varepsilon$
$\to b$

∴ new production rule:

$S \to ABAC|BAC|ABC|BC|AAC|AC|\varepsilon$
$A \to aA|a$
$B \to bB|b$
$C \to c$

Example: $S \to ABaC, A \to BC, B \to b/\varepsilon, C \to D/\varepsilon,$
$D \to d$

Soln: Here, NULL production:
$B \to \varepsilon, \quad C \to \varepsilon$

(i) to remove $B \to \varepsilon$

a) $S \to ABaC$
$\to AEaC$
$\to AaC$

b) $A \to BC$
$\to EC$
$\to C$

∴ new production rule:

$S \to ABaC \mid AaC \mid C$
$B \to b$
$C \to D \mid \varepsilon, \quad D \to d$
$A \to BC \mid C$

(ii) to remove $C \to \varepsilon$

a) $S \to ABaC$
$\to ABaE$
$\to ABa$

b) $A \to BC$
$\to BE$
$\to B$

∴ c) $S \to AaC$
$\to AaE$
$\to Aa$

d) $A \to C$
$A \to \varepsilon$

∴ new production rule:

$S \to ABaC \mid AaC \mid ABa \mid Aa$
$A \to BC \mid B \mid C \mid \varepsilon, \quad B \to b$
$C \to D, \quad D \to d$

(iii) to remove $A \to \varepsilon$

a) $S \to ABaC$
$\to BaC$

b) $S \to AaC$
$\to aC$

c) $S \to ABa$
$\to Ba$

d) $S \to Aa$
$\to a$

∴ new production rule:

$S \to ABaC \mid AaC \mid ABa \mid Aa \mid BaC \mid aC \mid Ba \mid a$
$A \to BC \mid B \mid C$
$B \to b$
$C \to D$
$D \to d$

CIMP preboard 2024)

Simplify the Grammar / reduce

$S \to aA \mid aBB$
$A \to aAA \mid \varepsilon$
$B \to bB \mid bbC$
$C \to B$

Soln: (i) first remove NULL production, then unit production and at last useless symbol.

(i) Removing ε- production gives resulting grammar,

$S \to aA \mid aBB \mid a$
$A \to aAA \mid aA \mid a$

$B \to bB \mid bbC$
$C \to B$

(ii) Removing Unit production, we get

$S \to aA \mid aBB \mid a$
$A \to aAA \mid aA \mid a$
$B \to bB \mid bbC$
$C \to bB \mid bbC$

(iii) Remaing useless symbols, we get not generating

Here, C is useless, since it is not reachable. Also B is useless. It doesn't derive an string.

final, CFG,
$S \to aA \mid a$
$A \to aAA \mid aA \mid a$

Normal Forms: (2024, spring) contain restriction in production rule

Types:

imp (i) Chomsky Normal Forms (CNF): (1)

- elements in R.H.S is either two variables (non-terminals) or a terminal.

Defn: A ~~CN~~ CFG is in CNF if the productions are in following form:

contain restriction on production rule

$A \to a$
$X \to AB$

Procedure for converting to CNF:

step1: if the start symbol occurs on right side create a new start symbol S' and a new production $S' \to S$.

step2: remove null production

step3: remove unit production

step4: remove string of terminal on right hand side if it exceeds as follows:

suppose, $S \to a_1a_2a_3$, where $a_1a_2a_3$ are terminals
Then, we introduce, new variable as
$C_{a_1} \to a_1$, $C_{a_2} \to a_2$, $C_{a_3} \to a_3$

step5: To restrict the no. of variable on the R.H.S., introduce new variable and seperate them. Suppose we have the production with n non-terminals,

$Y \to X_1X_2X_3X_4X_5$, $n = 5$

Add, n-2 new production using n-2 new non-terminals

$Y \to X_1R_3$ $\quad R_1 \to X_4X_5$
$R_3 \to X_2R_2$
$R_2 \to X_3R_1$

steps: if the right side of any production is in the form $A \to aB$, then the production is replaced by $A \to XB$ and $X \to a$

example: Convert CFG to CNF.

$S \to ASA \mid aB$
$A \to B \mid S$
$B \to b \mid \varepsilon$

Soln: step1: since, S is in R.H.S, we add $S' \to S$

P: $S' \to S$, $S \to ASA \mid aB$, $A \to B \mid S$, $B \to b \mid \varepsilon$

step2: remove null production: $B \to \varepsilon$

$S' \to S$, $S \to ASA \mid aB \mid a$, $A \to B \mid S \mid \varepsilon$
$B \to b$

remove, $A \to \varepsilon$

P: $S' \to S$, $S \to ASA \mid aB \mid a \mid SA \mid AS \mid S$
$A \to B \mid S$
$B \to b$

step3: remove unit production:

$S \to S$, $S \to S$, $A \to B$, $A \to S$

after removing $S \to S$: $S \to S$

P: $S' \to S$, $S \to ASA \mid aB \mid a \mid SA \mid AS$
$A \to B \mid S$, $B \to b$

after removing $S' \to S$: P: $S' \to ASA \mid aB \mid a \mid AS \mid SA$
$S \to ASA \mid aB \mid a \mid AS \mid SA$
$A \to B \mid S$, $B \to b$

after removing $A \to B$:

$S' \to ASA \mid aB \mid a \mid AS \mid SA$
$S \to ASA \mid aB \mid a \mid AS \mid SA$
$A \to b \mid S$, $B \to b$

after removing $A \to S$:

$S' \to ASA \mid aB \mid a \mid AS \mid SA$
$S \to ASA \mid aB \mid a \mid AS \mid SA$
$A \to b \mid ASA \mid aB \mid a \mid AS \mid SA$
$B \to b$

step4: Replace $S' \to ASA$, $S \to ASA$, $A \to ASA$ as:

$S' \to ASA$
$\to AX$, where $X \to SA$

Also,

$S \to AX$ $[X \to SA]$
$A \to AX$ $[X \to SA]$

we replace, $S' \to aB$, $S \to aB$, $A \to aB$ as:

$S' \to aB$
$\to YB$, where, $[Y \to a]$

also,

$S \to YB$ $(Y \to a)$
$A \to YB$ $(Y \to a)$

Final required CNF:

$S' \to AX \mid YB \mid a \mid AS \mid SA$
$S \to AX \mid YB \mid a \mid AS \mid SA$
$A \to b \mid AX \mid YB \mid a \mid AS \mid SA$
$B \to b$
$X \to SA$
$Y \to a$

Example: Convert CFG to CNF. (2024, Fall)

$S \to bA \mid aB$, $A \to bAA \mid aS \mid a$, $B \to aBB \mid bS \mid b$

Soln: step1: since S is in R.H.S, we add $S' \to S$

P: $S' \to S$, $S \to bA \mid aB$, $A \to bAA \mid aS \mid a$,
$B \to aBB \mid bS \mid b$

step2: there is no null and unique production.

step3: replace, $A \to bAA$, $B \to aBB$, $S \to bA$, $S \to aB$,
$A \to aS$, $B \to bS$ as:

(i) $A \to bAA$
$A \to$ ~~[illegible]~~, where $X \to AA$
$A \to XA$ $[X \to bA]$

(ii) $B \to aBB$
$B \to YB$ $[Y \to aB]$

(iii) $X \to bA$
$X \to CA$ $[C \to b]$

(iv) $Y \to aB$
$\to DB$ $[D \to a]$ X

(v) $S \to bA$
$\to EA$ $[E \to b]$

(vi) $S \to aB$
$\to FB$ $[F \to a]$

(vii) $A \to aS$
$\to GS$ $[G \to a]$

(viii) $B \to bS$
$\to HS$ $[H \to b]$

∴ Final required CNF:

~~$S' \to S$~~

step2: remove unique production:

removing: $S' \to S$

P: $S' \to bA \mid aB$, $S \to bA \mid aB$, $A \to bAA \mid aS \mid a$,
$B \to aBB \mid bS \mid b$

steps: replace, $A \to bAA$, $B \to aBB$:

(i) $A \to bAA$
$\to XA$ $[X \to bA]$

(ii) $B \to aBB$
$\to YB$ $[Y \to aB]$

replace, $X \to bA$, $Y \to aB$, $S' \to bA$, $S' \to aB$, ~~$S \to bA$~~, $S \to aB$,

(iii) $A \to aS$, $B \to bS$

(iii) $X \to bA$
$\to ZA$ $[Z \to b]$

(iv) $Y \to aB$
$\to CB$ $[C \to a]$

(v) $S' \to bA$
$\to DA$ $[D \to b]$

(vi) $S' \to aB$
$\to CB$ $[C \to a]$

(vii) $S \to bA$
$\to ZA$ $[Z \to b]$

(viii) $S \to aB$
$\to CB$ $[C \to a]$

∴ Final required CNF:

P: $S' \to ZA \mid CB$, $S \to ZA \mid CB$,
$A \to XA \mid DS \mid a$
$B \to YB \mid ES \mid b$

ix) $A \to aS$
$\to DS$ $[D \to a]$

x) $B \to bS$
$\to ES$ $[E \to b]$

Date:
Page:

example: Convert into CNF:

$S \to AB \mid aB$
$A \to aab \mid \varepsilon$
$B \to bbA$

i) remove NULL production:

$S \to AB \mid aB \mid B$
$A \to aab$
$B \to bbA \mid bb$

ii) remove unit production:
unique production: $S \to B$

P: $S \to AB \mid aB \mid bbA \mid bb$
$A \to aab$
$B \to bbA \mid bb$

iii) replace, ~~$S \to aB$~~, $S \to bbA$, $B \to bbA$, $A \to aab$

a) $S \to bbA$
$\to XA$ $[X \to bb]$

* $A \to aab$
$\to WB$ $[W \to aa, U \to b]$

b) $B \to bbA$
$\to XA$ $[X \to bb]$

$W \to aa$
$\to VV$ $[V \to a]$

replace, $X \to bb$, $S \to aB$, $S \to bb$, $B \to bb$

c) $X \to bb$
$X \to YY$ $[Y \to b]$

~~$X \to YS$ $[Y \to b, \text{rest}]$~~

d) $S \to aB \to zB \quad [z \to a]$

e) $S \to bb \to yy \quad [y \to b]$

f) $B \to bb \to yy \quad [y \to b]$

∴ required CNF:

$S \to AB \mid zB \mid XA \mid yy$
$A \to WU \mid$
$B \to XA \mid yy$
$W \to VV$
$U \to b$
$V \to a$
$y \to b$
$z \to a$

(11)

Chomsebach Normal form (CNF)
A CFG is in CNF if production are following form:

$A \to b$
$A \to bC_1C_2 \ldots C_n$

where, $A, C_1, C_2, \ldots C_n$ are non-terminals and b is terminal.

procedure:

step1: check if the given CFG has any unique production or NULL production and remove if these any.

step2: check whether the CFG is in CNF, and convert it to CNF if it's not.

step3: change the name of non-terminal into some Ai in ascending order of i.

Eg: $S \to CA \mid BB$ // look this
$B \to b \mid SB$
$C \to b$
$A \to a$

Here we replace S with A_1, C with A_2.
A with A_3, B with A_4
we get,
$A_1 \to A_2A_3 \mid A_4A_4$, $A_4 \to b \mid A_1A_4$
$A_2 \to b$, $A_3 \to a$

Step4: Alter the rules so that non-terminals are in ascending order, such that if the production is of the form

$A_i \to A_j X$; then $i < j$ and should never be $i \geq j$

now, $A_4 \to b \mid A_1 A_4$ $[i > j]$

$A_4 \to b \mid A_2 A_3 A_4 \mid A_4 A_4 A_4$

[$A_1 \to A_2 A_3$ substitute]

(left recursion हटाना है)

$A_4 \to b \mid bA_3A_4 \mid A_4A_4A_4$ [$A_2 \to b$]

Step5: remove left recursion [Introduce new variable]

$A_4 \to b \mid bA_3A_4 \mid A_4A_4A_4$

$Z \to A_4A_4Z \mid A_4A_4$

$A_4 \to b \mid bA_3A_4 \mid bZ \mid bA_3A_4Z$

now, the grammar is

$A_1 \to A_2A_3 \mid A_4A_4$

$A_4 \to b \mid bA_3A_4 \mid bZ \mid bA_3A_4Z$

$Z \to \mid A_4A_4Z \mid A_4A_4$

$A_2 \to b$

$A_3 \to a$

Again, we convert into GNF:

$A_1 \to bA_3 \mid bA_4 \mid bA_3A_4A_4 \mid bZA_4 \mid bA_3A_4ZA_4$

$A_4 \to b \mid bA_3A_4 \mid bZ \mid bA_3A_4Z$

$Z \to bA_4Z \mid bA_3A_4A_4Z \mid bA_4Z \mid bA_3A_4Z A_4A_4Z \mid bA_3 \mid bA_3A_4A_4 \mid bZA_4 \mid bA_3A_4A_4Z$

$A_2 \to b$

$A_3 \to a$

example: Convert into GNF:

$S \to AA \mid a$

$A \to SS \mid b$

soln: step1: replace S with A_1, A with A_2 we get,

$A_1 \to A_2A_2 \mid a$

$A_2 \to A_1A_1 \mid b$

step2: In, $A_2 \to A_1A_1 \mid b$ $(i > j)$

replace it,

$A_2 \to A_2A_2A_1 \mid aA_1 \mid b$

since $i \geq j$, replace it:

$A_2 \to A_1A_1A_2A_1 \mid bA_2A_1 \mid aA_1 \mid b$

The grammar is:

$A_1 \rightarrow A_2A_2 \mid a$

$A_2 \rightarrow A_1A_1A_2A_2 \mid bA_2A_1 \mid aA_1 \mid b$

again, we convert into GNF:

Step 3: remove left recursion:

$A_2 \rightarrow A_2A_2A_1 \mid aA_1 \mid b$

$Z \rightarrow A_2A_2Z \mid A_2A_2$

$Z \rightarrow A_2A_2Z \mid A_2A_2$

$A_2 \rightarrow aA_1 \mid b \mid aA_1Z \mid bZ$ ✓

$A_2 \rightarrow aA_1 \mid b \mid aA_1Z \mid bZ$

now, the grammar is:

$A_1 \rightarrow A_2A_2 \mid a$

$A_2 \rightarrow aA_1 \mid b \mid aA_1Z \mid bZ$

$Z \rightarrow A_2A_2Z \mid A_2A_2$

Again, we convert into GNF:

$A_1 \rightarrow aA_1A_2 \mid bA_2 \mid aA_1ZA_2 \mid bZA_2 \mid a$

$A_2 \rightarrow aA_1A_2 \mid b \mid aA_1ZA_2 \mid bZA_2$

$Z \rightarrow aA_1A_2Z \mid bA_2Z \mid aA_1ZA_2Z \mid bA_2A_2Z \mid aA_1A_2 \mid bA_2 \mid aA_1ZA_2 \mid bZA_2$

Pushdown Automata (PDA)

A PDA is a machine design to implement context-free language.

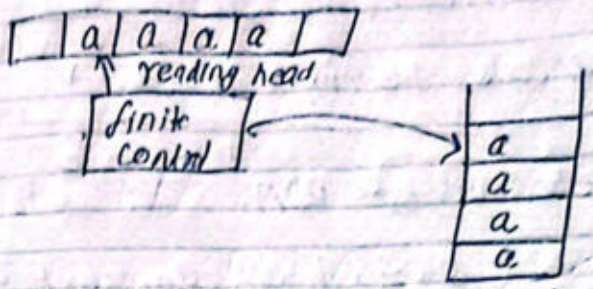


fig: PDA

- CFL is accepted by PDA.
- similar to FA, but it has some extra memory, called pushdown stack.
- read / write can be done on stack.

Eg: $L = \{a^n b^n \mid n \geq 1\}$ is not regular, but accepted by PDA.

- Formally, A PDA consists of seven tuples defined as,

$M = \{Q, \Sigma, \Gamma, \delta, q_0, Z_0, F\}$

where,

Q = set of finite state

Σ = set of symbols

Γ = set of symbols in stack

δ = transition function, $\delta(q, a, X)$

- q is state in Q
- a is input symbol
- X is memory of Γ

Γ = tape symbol (stack)

q_0 = initial state
Z_0 = top of stack
F = set of final state

1.) Instantaneous Description (ID)

- shows behaviour of PDA. [NCIT]

Suppose, $A = \{Q, \Sigma, \Gamma, \delta, q_0, z_0, F\}$ is a PDA and it's ID is (q, x, α).

~~Eg: $(q_0 q_1 q_2 \ldots q_n, a_1 a_2 \ldots a_n, Z_0 Z_1 \ldots Z_n)$~~

Eg: $(q_0 q_1 q_2 \ldots q_n, a_1 a_2 \ldots a_n, Z_0 Z_1 \ldots Z_n)$ is ID.

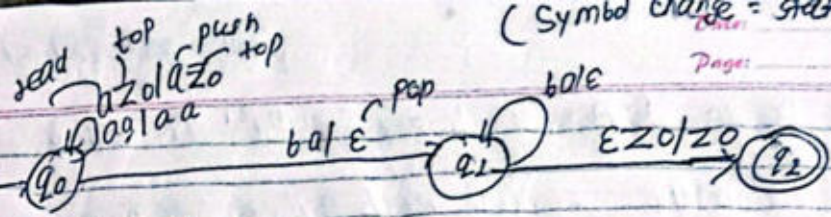
This describes when PDA is in current state q_0 and input symbol is to be processed i.e. $a_1 a_2 \ldots a_n$ and pushdown stack has $Z_0 Z_1 \ldots Z_n$, where Z_0 is top element, Z_1 is second top element & Z_n is the lowest element.

Example: Design a PDA for $L = \{a^n b^n \mid n \geq 1\}$ $\{0^n 1^n \mid n \geq 0\}$

Soln: Concept: push every a in the stack then reading each b remove each a from the stack.

Initially, top of the stack is Z_0 i.e. empty.

(Symbol change = state change)



Thus, PDA is
$A = \{Q, \Sigma, \Gamma, \delta, q_0, z_0, F\}$
where,
$Q = \{q_0, q_1, q_2\}$
$\Sigma = \{a, b\}$
$\Gamma = \{a, Z_0\}$
$q_0 = \{q_0\}$
$F = \{q_2\}$

δ is:

$\delta[q_0, a, Z_0) = (q_0, aZ_0)$ // push
$\delta(q_0, a, a) = (q_0, aa)$
$\delta(q_0, b, a) = (q_1, \varepsilon)$ // pop
$\delta(q_1, b, a) = (q_1, \varepsilon)$
$\delta(q_1, \varepsilon, Z_0) = (q_2, Z_0)$

we simulate PDA for string 'aaabbb'as,

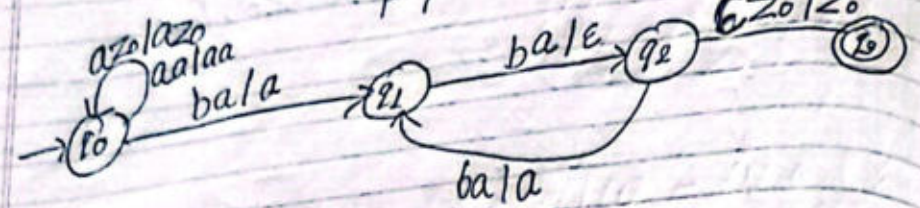
ID:

$\delta(q_0, aaabbb, Z_0) \vdash (q_0, aabbb, aZ_0)$
$\vdash (q_0, abbb, aaZ_0)$
$\vdash (q_0, bbb, aaaZ_0)$
$\vdash (q_1, bb, aaZ_0)$
$\vdash (q_1, b, aZ_0)$
$\vdash (q_1, \varepsilon, Z_0)$
$\vdash (q_2, Z_0)$

Hence accepted.

Q. Design PDA for $L = \{a^n b^n \mid n \geq 1\}$

Soln: concept: - push each 'a' in stack
- skip first 'b' and for 2nd b to pop 'a'.



Thus PDA is:

$A = \{Q, \Sigma, \Gamma, \delta, q_0, z_0, F\}$

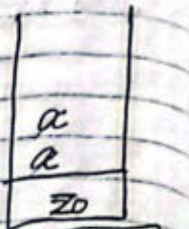
$Q = \{q_0, q_1, q_2, q_3\}$

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

$\Gamma = \{a, z_0\}$

$q_0 = \{q_0\}$

$F = \{q_3\}$



aabbbb [stack]

δ:

$\delta(q_0, a, z_0) = (q_0, az_0)$ // push

$\delta(q_0, a, a) = (q_0, aa)$ // "

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

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

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

$\delta(q_2, \varepsilon, z_0) = (q_3, z_0)$

we simulate for string: 'aabbbb'

$\delta(q_0, aabbbb, z_0) \vdash (q_0, abbbb, az_0)$

$\vdash (q_0, bbbb, aaz_0)$

$\vdash (q_1, bbb, aaz_0)$

$\vdash (q_2, bb, az_0)$

$\vdash (q_1, b, az_0)$

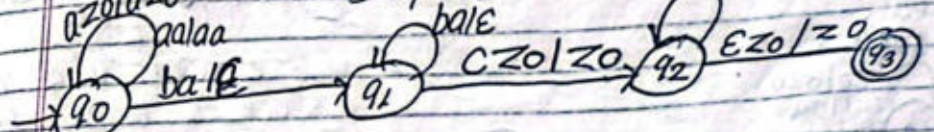
$\vdash (q_2, \varepsilon, z_0)$

$\vdash (q_3, z_0)$

accepted

Q. Design PDA for $L = \{a^n b^n c^m \mid n, m \geq 1\}$

Soln: concept: push each a on stack
- pop each b reading each b &
pop each a
- skip c.



PDA is: $A = \{Q, \Sigma, \Gamma, \delta, q_0, z_0, F\}$

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

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

$\Gamma = \{a, z_0\}$

$q_0 = \{q_0\}$

$F = \{q_3\}$

we simulate PDA for, aabbc :

$\delta(q_0, aabbc, z_0) \vdash (q_0, abbc, az_0)$
$\vdash (q_0, bbc, aaz_0)$
$\vdash (q_1, bc, az_0)$
$\vdash (q_1, c, z_0)$
$\vdash (q_2, c, z_0)$
$\vdash (q_3, z_0)$

Assignment

Q3. Design PDA for $L = \{ w \mid w \in (a+b)^* \}$ and i) $n_a(\omega) = n_b(\omega)$ (i.e. equal no. of a's & b's)

ii) $n_a(\omega) > n_b(\omega)$

iii) $L = \{ a^n b b^n \mid n > 0 \}$

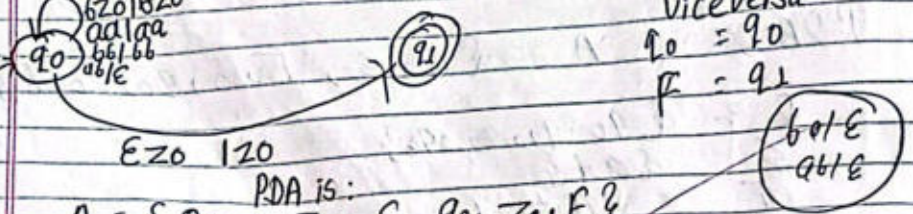
iv) $L = \{ a^n b^m c^n \mid n, m \geq 1 \}$

Soln: i) $n_a(\omega) = n_b(\omega)$

concept: whenever we read either 'a' or 'b' push it to stack, when stack is empty pop a or b

- if we read 'a' & top of stack is 'b' pop b and viceversa.

$q_0 = q_0$
$F = q_1$



PDA is:

$A = \{ Q, \Sigma, \Gamma, \delta, q_0, z_0, F \}$

where: $Q = \{ q_0, q_1 \}$
$\Sigma = \{ a, b \}$

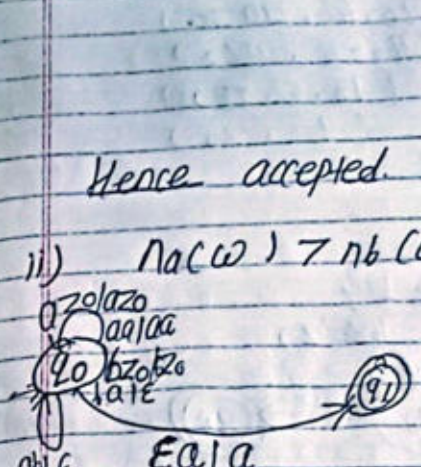
we simulate PDA for abab

$\delta(q_0, abab, z_0) \vdash (q_0, bab, az_0)$
$\vdash (q_0, ab, z_0)$
$\vdash (q_0, b, az_0)$
$\vdash (q_0, \varepsilon, z_0)$
$\vdash (q_1, z_0)$

Hence accepted

concept:

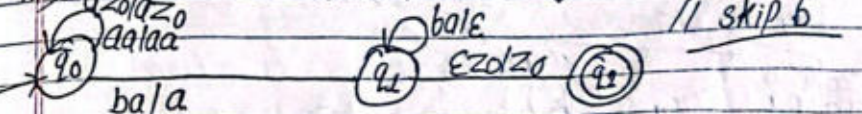
ii) $n_a(\omega) > n_b(\omega)$ → push a, b if stack is empty
→ skip b if top is empty
→ pop a if we read b and viceversa.



we simulate PDA for aab

$\delta(q_0, aab, z_0) \vdash (q_0, ab, az_0)$
$\vdash (q_0, b, aaz_0)$
$\vdash (q_0, \varepsilon, az_0)$
$\vdash (q_1, az_0)$

iii) $L = \{ a^n b b^n \mid n > 0 \}$



Date:
Page:

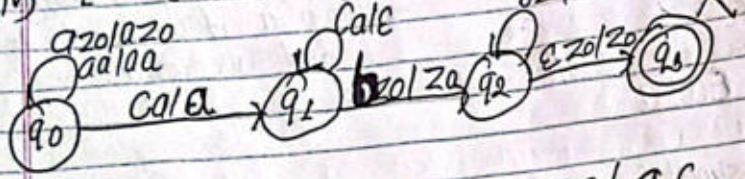
we simulate for: aabbb

$\delta(q_0, aabbb, z_0) \vdash (q_0, abbb, az_0)$
$\vdash (q_0, bbb, aaz_0)$
$\vdash (q_0, bb, aaaz_0)$
$\vdash (q_0, b, aaz_0)$
$\vdash (q_0, \varepsilon, z_0)$
$\vdash (q_1, z_0)$

Hence accepted.

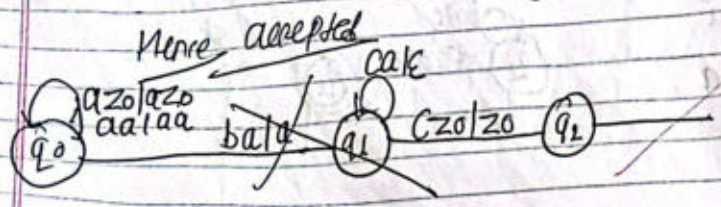
(imp)

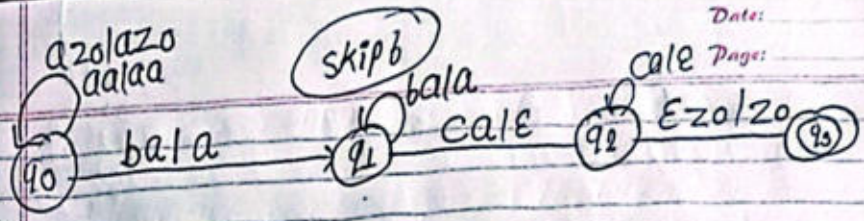
iv) $L = \{ a^n b^m c^n \mid n, m \geq 1 \}$



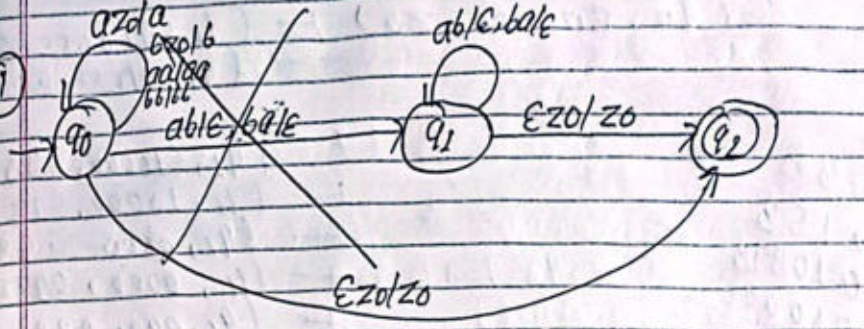
we simulate PDA for aabacc

$\delta(q_0, aabcc, z_0) \vdash (q_0, abcc, az_0)$
$\vdash (q_0, bcc, aaz_0)$
$\vdash (q_0, cc, aaz_0)$
$\vdash (q_0, c, az_0)$
$\vdash (q_0, \varepsilon, z_0)$
$\vdash (q_3, z_0)$





(imp)



Equivalence of CFG and PDA:

① Construct PDA from CFG (2024, Sp 11/9)

$S \to aBB$
$B \to aS \mid bS \mid a$

Sol^n: suppose, PDA $= \{Q, \Sigma, \Gamma, \delta, q_0, z_0, F\}$

now, we define transition function δ as,

$\delta(q_0, \varepsilon, z_0) = (q_1, Sz_0)$
~~$\delta(q_1, \varepsilon, z_0) = (q_1, Bz_0)$~~ pop
$\delta(q_1, \varepsilon, S) = (q_1, aBB)$
$\delta(q_1, a, B) = (q_1, aS), (q_1, bS), (q_1, a)$
$\delta(q_1, a, a) = (q_1, \varepsilon)$
$\delta(q_1, b, b) = (q_1, \varepsilon)$
$\delta(q_1, \varepsilon, z_0) = (q_2, z_0)$

$S \to 0BB$
$B \to 0S | 1S | 0$

we simulate for '010000'

$(q_0, \varepsilon 010000, z_0) \vdash (q_1, \varepsilon 010000, SZ_0)$
$\vdash (q_1, 010000, 0BBZ_0)$
$\vdash (q_1, 10000, 1SBZ_0)$
$\vdash (q_1, 0000, SBZ_0)$
$\vdash (q_1, 0000, 0BBBZ_0)$
$\vdash (q_1, 000, BBBZ_0)$
$\vdash (q_1, 000, 0BBZ_0)$
$\vdash (q_1, 00, BBZ_0)$
$\vdash (q_1, 0, BZ_0)$
$\vdash (q_1, \varepsilon, 0Z_0)$
$\vdash (q_1, \varepsilon, Z_0)$
$\vdash (q_2, Z_0)$

0BB
0 1SB
0 10BBB
010000

Hence accepted

Assignment:

Q1. Construct PDA from CFG.

$S \to aSa / bSb / c$. ~~we simulate for~~

Soln: Suppose, PDA $= \{Q, \Sigma, \Gamma, \delta, q_0, Z_0, F\}$

now, we define transition function δ as:

$\delta(q_0, \varepsilon, Z_0) = (q_1, SZ_0)$
$\delta(q_1, \varepsilon, S) = (q_1, aSa), (q_1, bSb), (q_1, \varepsilon)$
$\delta(q_1, a, a) = (q_1, \varepsilon)$
$\delta(q_1, b, b) = (q_1, \varepsilon)$
$\delta(q_1, a, b) = (q_1, \varepsilon)$
$\delta(q_1, c, c) = (q_1, \varepsilon)$
$\delta(q_1, \varepsilon, Z_0) = (q_2, Z_0)$

$S \to aSa$
$\to abSba$
$\to abcba$

we simulate for,
'abcba'

$(q_0, \varepsilon abcba, z_0) \vdash (q_1, \varepsilon abcba, SZ_0)$
$\vdash (q_1, abcba, bSaZ_0)$
$\vdash (q_1, bcba, caZ_0)$
$\vdash (q_1, cba, aZ_0)$
$\vdash (q_1, ba, Z_0)$
$\vdash (q_1, \varepsilon abcba)$

$(q_0, \varepsilon abccba, z_0) \vdash (q_1, SZ_0)$
$\vdash (q_1, abcba, bSbZ_0)$
$\vdash (q_1, \varepsilon bcba, SbZ_0)$
$\vdash (q_1, bcba, cbZ_0)$
$\vdash (q_1, cba, bZ_0)$
$\vdash (q_1, ba, Z_0)$
$\vdash (q_1, a, bZ_0)$
$\vdash (q_1, \varepsilon, Z_0)$
$\vdash (q_2, Z_0)$

Here accepted

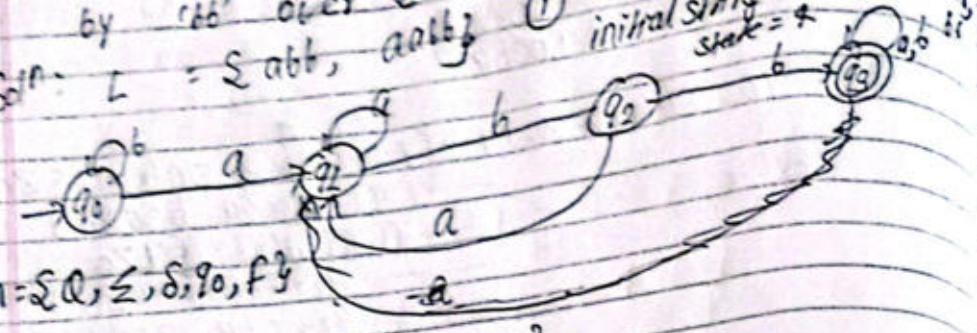
rules to design PDA from CFG:
→ initialize PDA
production rule → Handle non-terminal expansions
→ handle terminal matches
→ final state transitions.

First terminal 2025

2a) Design a DFA which accepts all the strings in which every 'a' should never be followed by 'bb' over $\Sigma = \{a, b\}$

Soln: $L = \{abb, aabb\}$

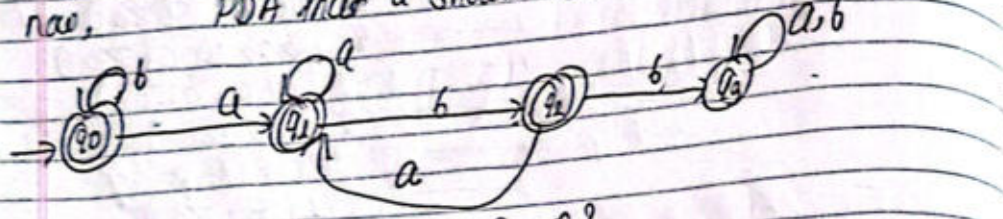
① DFA that accepts every a initial string = abb followed by



$M = \{Q, \Sigma, \delta, q_0, F\}$

$Q = \{q_0, q_1, q_2, q_3\}$
$\Sigma = \{a, b\}$
$q_0 = \{q_0\}$
$F = \{q_3\}$

now, PDA that 'a' should never be followed by bb



$M = \{Q, \Sigma, \delta, q_0, F\}$

where, $Q = \{q_0, q_1, q_2, q_3\}$
$\Sigma = \{a, b\}$
$q_0 = \{q_0\}$
$F = \{q_0, q_1, q_2\}$

2b) Write CFG which generates palindrome of $\{a, b\}$.

Soln: $L = \{aba, abba, bab, baab \ldots\}$

now, production rule is:

$P: S \rightarrow aAa \mid bAb$
$A \rightarrow b \mid a \mid \varepsilon \mid S$

now, grammar is:

$G = \{V, \Sigma, P, S\}$

where $V = \{S, A\}$
$\Sigma = \{a, b\}$
$P = \{S \rightarrow aAa \mid bAb,\ A \rightarrow b \mid a \mid S \mid \varepsilon\}$
$S = \{S\}$

now, we derive 'abba' from grammar:

$S \rightarrow aAa$
$\rightarrow aSa \quad (A \rightarrow S)$
$\rightarrow abAba \quad [S \rightarrow bAb]$
$\rightarrow ab\varepsilon ba \quad [A \rightarrow \varepsilon]$
$\rightarrow abba$

Hence, accepted #

abb

abb

// (आहिलो सम्म exam मा आईदैन)

Date:
Page:

Construct CFG from PDA

Here, PDA = {Q, Σ, Γ, δ, q0, Z0, F}
CFG = {V, T, P, S}

Here,
T = {Σ}

suppose,
Q = {q0 q1}, Γ = a

V can be defined as, [Q Γ Q]

so, V = {S, [q0 a q0], [q0 a q1] [q1 a q1], [q1 a q0]}

now,
suppose, we have transition function,

① → δ(q0, a, Z0) = (q0, ε)
[q0 Z0 q0] → a

② → δ[q0, a, Z0) = (q0, aZ0)
[q0 Z0 q0] → a [q0 a q1] [q1 Z0 q0]
same

जे पनि कुनचाहिं Z0 को छुट्टैमा

③ → δ(q0, a, Z0) = (q0, aXX)
[q0 Z0] → a [q0 a] [X] [X]
same
same
same

④ → δ(q0, b, a) = (q1, a)
[q0 a] → b [q1 a]
same state

example:
Given PDA, A = ({q0, q1}, {a, b}, {a, Z0}, δ, q0, Z0, {φ})
Q, Σ, Γ, transition function: δ, q0, Z0, {φ} and given by final state

δ(q0, a, Z0) = (q0, aZ0)
δ(q0, a, a) = (q0, aa)
δ(q0, b, a) = (q1, ε)
δ(q1, b, a) = (q1, ε)
δ(q1, ε, Z0) = (q1, ε)

Soln:
① Taking δ(q0, a, Z0) = (q0, aZ0)

P1: [q0 Z0 q0] → a [q0 a q0] [q0 Z0 q0]
P2: [q0 Z0 q0] → a [q0 a q1] [q1 Z0 q0]
P3: [q0 Z0 q1] → a [q0 a q0] [q0 Z0 q1]
P4: [q0 Z0 q1] → a [q0 a q1] [q1 Z0 q1]

(i) taking $\delta(q_0, a, a) = (q_0, aa)$

P_1: $[q_0\ a\ q_0] \to a\,[q_0\ a\ q_0]\ [q_0\ a\ q_0]$
P_2: $[q_0\ a\ q_0] \to a\,[q_0\ a\ q_1]\ [q_1\ a\ q_0]$
P_3: $[q_0\ a\ q_1] \to a\,[q_0\ a\ q_0]\ [q_0\ a\ q_1]$
P_4: $[q_0\ a\ q_1] \to a\,[q_0\ a\ q_1]\ [q_1\ a\ q_1]$

(ii) taking: $\delta(q_0, b, a) \to (q_1, \varepsilon)$

P_5: $[q_0\ b\ q_1] \to a$

(iii) taking: $\delta(q_1, b, a) \to (q_1, \varepsilon)$

P_6: $[q_1\ b\ q_1] \to a$

$A \to BC$
$B \to a/\varepsilon$
C is useless remove A.

(iv) taking: $\delta(q_1, \varepsilon, Z_0) = (q_1, \varepsilon)$

P_7: $[q_1\ \varepsilon\ q_1] \to \varepsilon$

Also, $S \to [q_0\ Z_0\ q_0]$ (initial state)
$\quad\quad | [q_0\ Z_0\ q_1]$

Now, we reduce useless symbol:

remove P_6 since $[q_1\ a\ q_0]$ is non generatic symbol. Also, remove P_5 because generatic itself. similarly remove P_7, P_1, P_3, P_2, P_4.
remove $S \to [q_0\ Z_0\ q_0]$

Final productions:

P:

$S \to [q_0 Z_0 q_1]$

$[q_0 Z_0 q_1] \to a[q_0 a q_1][q_1 Z_0 q_1]$

$[q_0 a q_1] \to a[q_0 a q_2][q_2 a q_1]$

$[q_0 a q_1] \to b$

$[q_1 a q_1] \to b$

$[q_1 Z_0 q_1] \to \varepsilon$ #

$\therefore V = \{S, \ldots\}$

$V = V_1 \cup V_2 \cup \{S\}$

$\Sigma = \Sigma_1 \cup \Sigma_2$

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

$w \in (\Sigma_1 \cup \Sigma_2)^*$

Closure properties of CFL (imp)

① closed under Union: (2003, Fall)

Let L_1 & L_2 be two CFL generated by context-free grammar.

$G_1 = \{V_1, \Sigma_1, P_1, S_1\}$ and

$G_2 = \{V_2, \Sigma_2, P_2, S_2\}$ respectively.

Now, we construct new language $L(G)$ using the grammar

$G = \{V, \Sigma, P, S\}$ such that it can accept $L_1(G_1) \cup L_2(G_2)$

where, $V = V_1 \cup V_2 \cup \{S\}$

$\Sigma = \Sigma_1 \cup \Sigma_2$

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

S = starting symbol

now, let us choose a string $w \in \{\Sigma_1 \cup \Sigma_2\}^*$

if $S_1 \xrightarrow{*} w$

or $S_2 \xrightarrow{*} w$

and in our grammar

$S \to S_1 | S_2$, so S will lead to string w.

[i.e. $S \xrightarrow{*} w$]

Hence, G is a context-free grammar that generate $L(G)$ such that $L(G) \to L_1(G_1) \cup L_2(G_2)$.

Date:
Page:

② Closed under Concatenation: (2024, Fall)

Let, L_1 & L_2 be two CFL generated by context-free grammar.

$G_1 = \{V_1, \Sigma_1, P_1, S_1\}$ and
$G_2 = \{V_2, \Sigma_2, P_2, S_2\}$ respectively.

Now, we construct a new ~~grammar~~ language $L(G)$ using the grammar
$G = \{V, \Sigma, P, S\}$ such that

it can accepts $L_1(G_1) . L_2(G_2)$

where,
$V = V_1 \cup V_2 \cup \{S\}$
$\Sigma = \Sigma_1 \cup \Sigma_2$
$P = P_1 \cup P_2 \cup \{S \to S_1 . S_2\}$
S = start state.

Now, let us choose string $\omega_1 \in \Sigma_1^*$ and $\omega_2 \in \Sigma_2^*$.

we know that,
$S_1 \xrightarrow{*} \omega_1$ and
$S_2 \xrightarrow{*} \omega_2$, but in the above grammar G, $S \to S_1 . S_2$.
So, S will lead the concatenation of the string ω_1 and ω_2.
i.e. $S \xrightarrow{*} \omega_1 \omega_2$. and
Since, L_1 and L_2 are CFL. So $L_1 . L_2$ is also a CFL.

Date:
Page:

⑪ Kleene closure

Let, L_1 be context free language generated by context free grammar $G_1 = \{V_1, \Sigma_1, P_1, S_1\}$, Now we construct new language $L(G)$ using the grammar

$L(G)$, $G = \{V, \Sigma, P, S\}$
such that it can accept Kleene star of the language L_1 (i.e. L_1^*)
where,
$V = \{V_1 \cup \{S\}\}$
$\Sigma = \Sigma_1 \cup \{\epsilon\}$
$P = P_1 \cup \{S \to \epsilon, S \to S_1 S\}$
S = start state

Here, production P follows all the properties of CFG as P_1 is a production of given CFG and $S \to \epsilon$, $S \to S_1 S$ also fullfill the requirement. So we said that G is a CFG that can generate context free language.

$L = L_1^*$

(imp)

④ CFL are not closed under (imp)

a) Intersection:

We know that, $L_1 = \{a^n b^n c^i \mid n \geq 1, i \geq 0\}$ and $L_2 = \{a^i b^j c^n \mid n \geq 1, j \geq 0\}$ are context free language.

Now, $L_1 \cap L_2 = \{a^n b^n c^n \mid n \geq 1\}$

Here, $L = L_1 \cap L_2$ is not context-free language.

b) Complement:

Using demargan's law

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

Since, CFL is closed under union but not closed under intersection. Hence, CFL is not closed under complement.

(imp)

Decision Algorithm for CFL

① Algorithm for deciding whether a context free language is finite or not.

Algorithm for finite?

- construct a non-determinant context free grammar G, generated language $L = S \cap \Sigma$. We draw a directed graph whose vertices are variables or non-termina in G.
- If $A \rightarrow BC$ is a production, then there are directed edges from A to B and A to L is finite iff the directed graph ha no cycle.

② Algorithm for deciding whether a CFL is empty.

- Given any cfl 'L', there is a CFG to generate it. We can determine using the construction describe in the contex of elimination of useless symbol, whether the start symbol is useless, if so th $L(G) = \emptyset$, otherwise not.

③ Algorithm for deciding whether any string w can be generated by the same CFG. (membership?)

- To determine whether a particular string w is in language of G, we will inspect all derivation trees of height at most length of string w. If string w is found in any of the derivation tree, then $w \in L$.

Pumping Lemma for CFL: (Imp)

Statement: Let, L be a context free language and n be the length of string or pumping length such that:

i) every $z \in L$ with $|z| \geq n$ can be written as $uvwxy$.

ii) $|vx| \geq 1$

iii) $|vwx| \leq n$

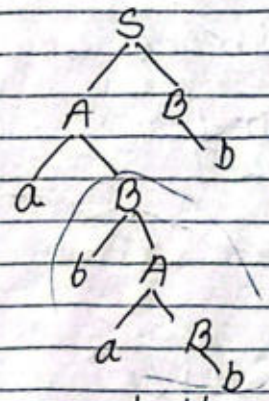
iv) $uv^k wx^k y \in L$ for all $k \geq 0$.

Proof: To prove the theorem we consider a CFG whose production are given by

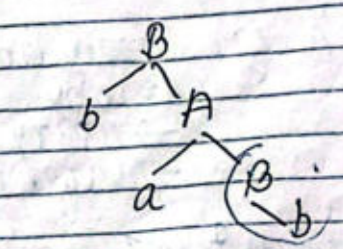
$S \rightarrow AB$
$A \rightarrow aB \mid a$
$B \rightarrow bA \mid b$

Let any string $z = ababb$ such that, $z \in L$.
Now, we decompose z as
$u = a$, $v = ba$, $w = b$, $x = \Lambda$, $y = b$
Now,
we construct derivation tree for z:

$S \rightarrow AB$
$\rightarrow aBB$ [A → aB]
$\rightarrow abAB$ [B → bA]
$\rightarrow ababB$ [A → a]
$\rightarrow ababb$ [B → b]



T: z = ababb



T_1: $z_1 = bab$

T_2: $z_2 = b$

As, z and z_1 are yields of T and a prop. subtree T_1 of T_2 so, we can write $z = uz_1y$ (i.e. $z = uvwxy$)

As z_1 and z_2 are yields of T_1 & prop. subtree T_2 of T_1, so we can write

$z_1 = vz_2x$ i.e. $z_1 = vwx$

Also, $|vwx| \geq |w|$. So $|vx| \geq 1$

Since, we have $z = uvwxy$ with length of $|vwx| \leq n$ and $|vx| \geq 1$

also, from the derivation tree we can write:

$S \xrightarrow{*} uBy$ $\quad |xy| \leq n$

$B \xrightarrow{*} vBx$ $\quad |vx| \geq 1$

$B \xrightarrow{*} w$

Now, from above production, we can write

$S \to uBy$

$\to uvBxy$ [B → vBx]

$\to uvwxy$ [B → w]

$\therefore uv^1wx^1y \in L$ for $k = 1$

again.

$S \to uBy$

$\to uvBxy$ [B → vBx]

$\to uvvBxxy$ [B → vBx]

$\to uv^2wx^2y$ [B → w]

$\therefore uv^2wx^2y \in L$, for $k = 2$ and so on.

uvwxy

Thus, we conclude $uv^kwx^ky \in L$, for $k \geq 0$. $\quad |vx| \geq 1$

Procedure to prove language is not CFL.

step1: Assume L is context-free. Let n be the no. of obtained by using pumping lemma

step2: Choose string $z \in L$. So that $|z| \geq n$. write, $z = uvwxy$ using the pumping lemma.

step3: Find suitable k so that $uv^kwx^ky \notin L$. This is a contradiction and L is not CFL

Example:- Show that $L = \{a^n b^n c^n \mid n \geq 1\}$ is not CFL.

Soln: step1: Assume L is context-free grammar n = no. of obtained. $\quad |vx| \geq 1$

$1 \leq |vx| \leq n$

step2: Let, $z = a^n b^n c^n$.

write $z = uvwxy$, where $|vx| \geq 1$

step3: $uvwxy = a^n b^n c^n$. As $1 \leq |vx| \leq n$, so v and x cannot contain all three symbols.

Suppose, $n = 4$

so, $z = a^4 b^4 c^4 = aaaabbbbcccc$

case1: v and x can contain one type of symbol

$z = aaaabbbbcccc$

for, $k = 2$

$uv^2wx^2y = aaaaaabbbbbccccc$

$= a^6b^5c^5$

Hence, no. of a's, b's & c's are not equal.

So, $uv^2wx^2y \notin L$ for $k=2$.

case2: either v or x has more than one kind of symbol.

$z = \underbrace{aaaa}_{u}\underbrace{abbb}_{v}\underbrace{bccc}_{w}\underbrace{cc}_{x}\underbrace{c}_{y}$

for, $k = 2$

$= uv^2wx^2y$

$= aaaabababbbbccccc$

$= a^4ba^4c^6$

since, not is pattern. So, $uv^2wx^2y \notin L$ for $k=2$.

Hence, in the above cases, we get contradiction. So $L = \{a^n b^n c^n \mid n \geq 1\}$ is not CFL.

Assignment

Q. Show that $L = \{ww \mid w \in (0+1)^*\}$ is not CFL.

Soln: step1: Let assume L is context-free language. n be no. of obtained by pumping lemma.

step2: let, $z = ww = 0^n10^n1$ where $w \in 0^n1$.

write $z = uvwxy$, where $|vx| \geq 1$

step3: $uvwxy = 0^n10^n1$, As $1 \leq |vx| \leq n$. So v and x can't contain all two symbols.

suppose,

$n = 4$

$z = 0^410^41$

$= 000010000 1$

case i: v and x contain only one type of symbol.

$z = \underbrace{000}_{u}\underbrace{0}_{v}\underbrace{1}_{w}\underbrace{000}_{x}\underbrace{1}_{y}$. $\underbrace{000}_{u}\underbrace{0}_{v}\underbrace{1}_{w}\underbrace{000}_{x}\underbrace{1}_{y}$

for $k = 2$

$uv^2wx^2y = 000000100000 1$

$= 0^610^61$

for $k = 2$

$uv^2wx^2y = 00000010000001$

$= 0^810^81$

$uv^2wx^2y = 0000001000001$

$= 0^610^51$

which is not is pattern. So, $uv^2wx^2y \in L$ for $k=2$.

case ii: either v or x has two symbols:

$z = \underbrace{000}_{u}\underbrace{01}_{v}\underbrace{000}_{w}\underbrace{01}_{x}\underbrace{1}_{y}$ $\underbrace{000}_{u}\underbrace{01}_{v}\underbrace{000}_{w}\underbrace{01}_{x}\underbrace{1}_{y}$

at $k = 2$

$uv^2wx^2y = 00001001000001$

$= 0^410^210^61$

which is not in pattern, so $uv^2wx^2y \notin L$ for $k=2$.

Hence, from above cases, we get contradiction. So $L = \{ww \mid w \in (0+1)^*\}$ is not CFL.

Unit 4 Turing Machine (10 Marks)

→ used for general computation
→ designed by Alan Turing.
→ most powerful machine.

$L = a^n b^n \rightarrow$ FA ✗
$L = a^n b^n \rightarrow$ PDA ✓
$L = a^n b^n c^n$ ✗
↳ TM ✓

Σ Q, Σ, Γ, δ, q_0, B, F

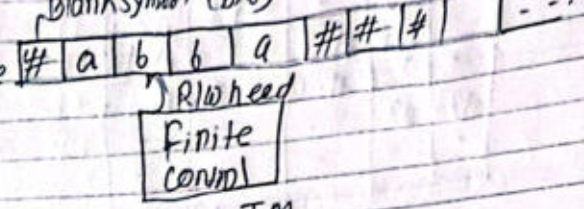


fig: TM

→ The Turing machine can be thought as finite state machine, Connected to read write head. it has one tape which is divided into no. of cells. each cell can store only one symbol. The input to & output from the machine are affected by a read write head which can examine one cell at a time.

→ in one more, the machine proces the present symbol under the read write head and the present state to determine:

i) a new symbol to be written on tape.
ii) a motion of read/write head along the tape either the head more one cell left or one cell right.
iii) the next state of machine.
iv) whether to halt or not.

Mathematically, TM contains 7 tuples.

$M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$
where,
Q = set of state
Σ = input symbol , B = blank symbol
Γ = set of tape symbol
δ = transition function
q_0 = initial state
F = set of final state

Instantaneous Description (ID)

- an ID of Turing Machine M is a string $\alpha\beta\gamma$ here β is the present state of M, the entire input string is spilt as α, γ.

The first symbol of γ is the current symbol under the read write head and γ has all the symbol of the input string and string α is the substring of the input formed by all the symbol to the left of present state

Q. Consider a turing machine describe by the following transition table. Describe the processing of
a) 001 b) 0011 c) 011 (xyii)
and using ID check it is accepted by Turing machine or not.

State	Tape symbol				
	0	1	x	y	b
→ q_1	xRq_2	-	-	-	bRq_5
q_2	$0Rq_2$	yLq_3	-	yRq_2	-
q_3	$0Lq_4$	-	xRq_5	yLq_3	-
q_4	$0Lq_4$	-	xRq_1	-	-
q_5	-	-	-	yRq_5	bRq_6
q_6*	-	-	-	-	-

R = right move
L = left "
(x/w head)

Soln:
a) for 001

$(q_1, 001) \vdash (xq_201)$
$\vdash x0q_21$
$\vdash xq_30y$
$\vdash q_4x0y$
$\vdash xq_10y$
$\vdash xxq_2y$
$\vdash xxyq_2b$

Halt, not accepted.

b) for 0011

$(q_1, 0011) \vdash (xq_20ii)$
$\vdash x0q_2ii)$
$\vdash (xq_30yi.)$
$\vdash (xq_4xoyi)$
$\vdash (xq_1oyi)$
$\vdash (xxq_2yi)$
$\vdash xxyq_2i.$
$\vdash xxq_3yy$
$\vdash xq_3xyy$
$\vdash xxq_5yy$
$\vdash xxyq_1y$
$\vdash$ ~~xxyqi~~

Halt not accepted.

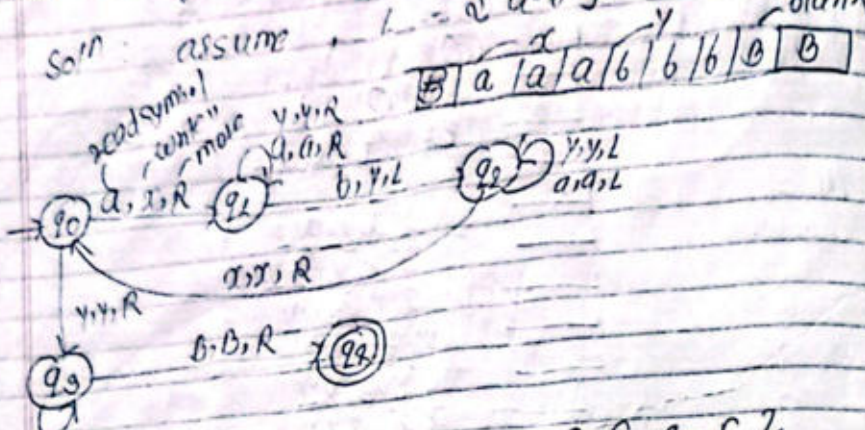
c) $(q_1, 011) \vdash (xq_211)$
$\vdash (q_3xy01)$
$\vdash (xq_5y1)$
$\vdash (xyq_11)$
$\vdash$

Halt, not accepted.

2019, fall

Imp. Design TM for $L = \{a^n b^n \mid n \geq 1\}$

Soln: assume $L = \{a^3 b^3\}$



$M = \{Q, \Sigma, \Gamma, \delta, q_0, F\}$

where, $Q = \{q_0, q_1, q_2, q_3, q_4\}$

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

$\Gamma = \{a, b, B, x, y\}$

$q_0 = q_0$

$F = q_4$

δ:

state	Tape symbol a	b	x	y	B
→q_0	xRq_1	-	-	yRq_3	-
q_1	aRq_1	yLq_2	-	yRq_1	B
q_2	aLq_2	-	xRq_0	yLq_2	-
q_3	-	-	-	yRq_3	BRq_4
*q_4	-	-	-	-	-

we estimate for string 'aabb':

$\delta(q_0, aabb) \vdash (xq_1, abb)$

$\vdash (xaq_1bb)$

$\vdash xq_2ayb$

$\vdash q_2xayb$

$\vdash xq_0ayb$

$\vdash xxq_1yb$

$\vdash xxyq_1b$

$\vdash xxq_2yy$

$\vdash xxyq_2y$

$\vdash xxq_0yy$

$\vdash xxyq_3y$

$\vdash xxyyq_3B$

$\vdash xxyyBq_4B$

$\vdash xxyyBq_4$

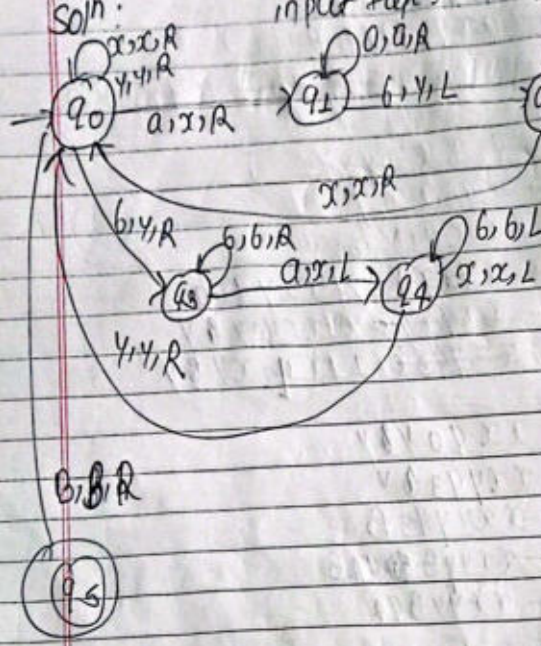
Halt,

so accepted as q_4 is final state.

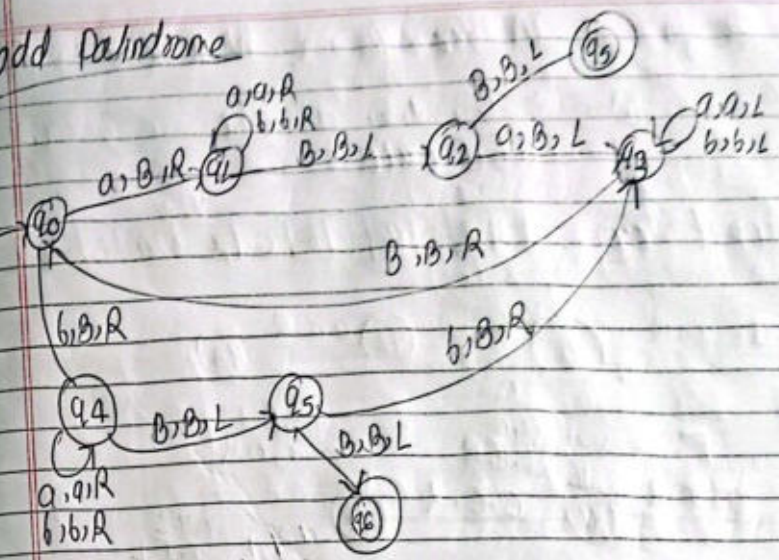
9766999258

Assignment

Q: Design TM for equal no. of a's & b's over $\Sigma = \{a, b\}$

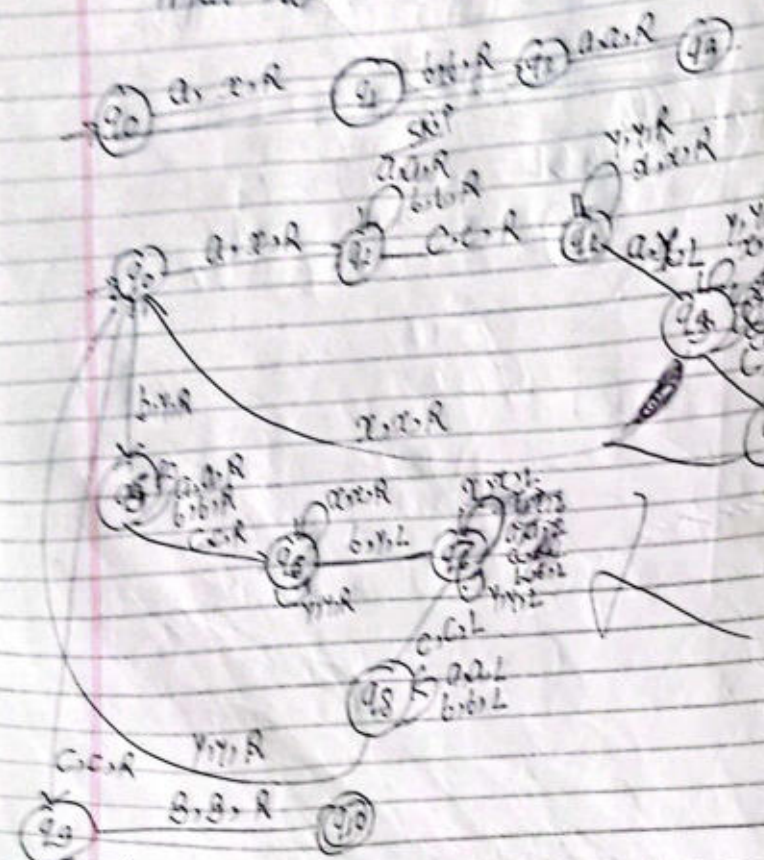


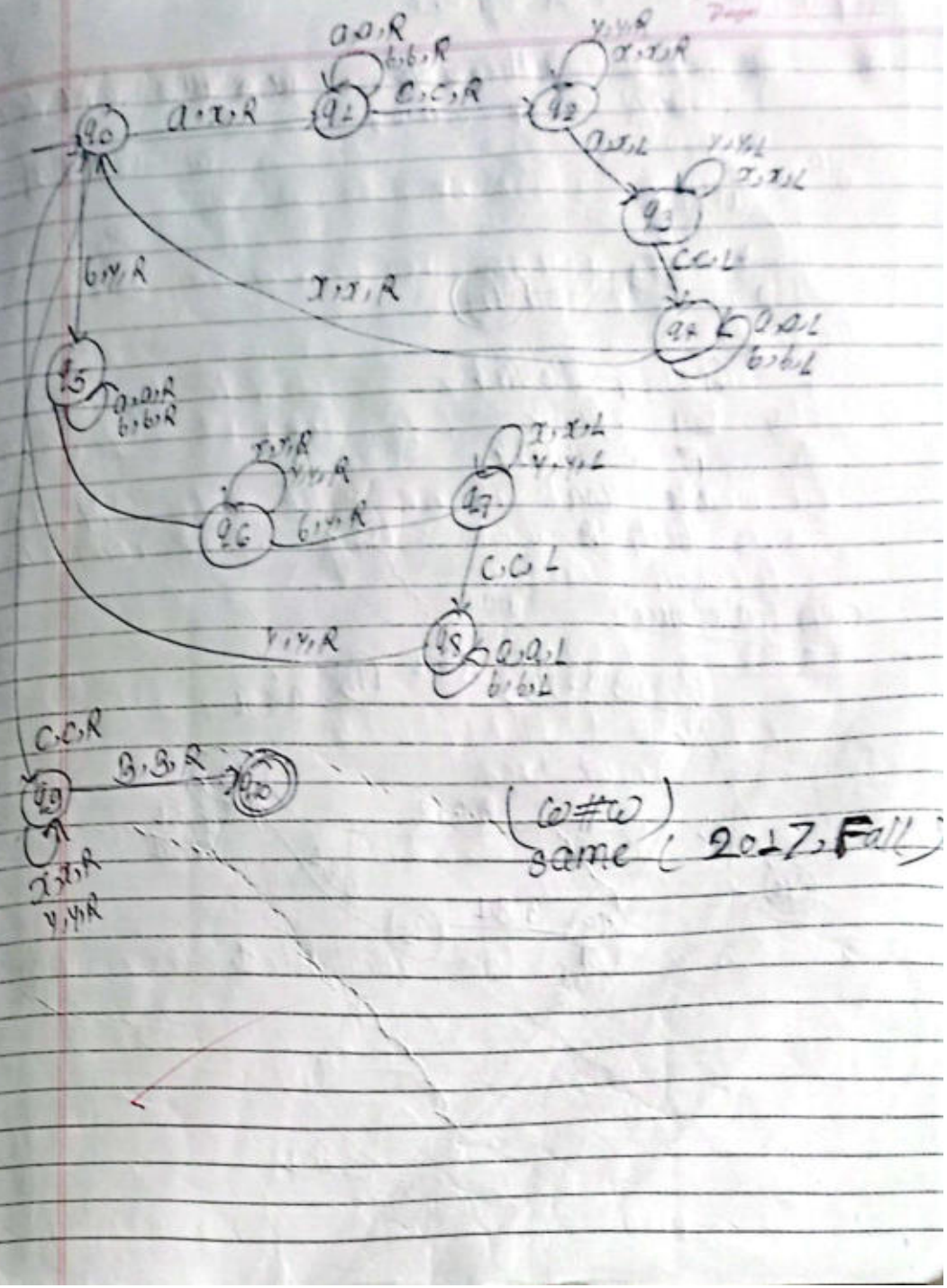
odd Palindrome



Q: Design a TM for $L = \{ ww \mid w \in (a+b)^* \}$

Soln: $w = \epsilon, a, b, ab, aba \ldots$ $w = abaabbCabRbb$





(w#w)

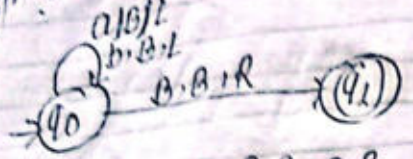
same (2017, Fall)

(Q/K in question) Assignment

Q3. Design a TM that erase all symbols of string over Σ{a,b}

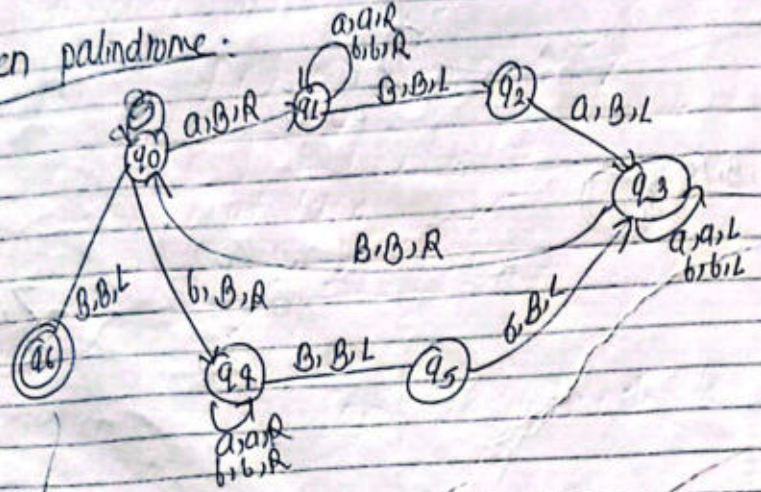
ω = a b b a

Soln:



Q = {q0, q1}
Σ = {a, b}
Γ = {a, b, B}
q0 = q0
F = q1

even palindrome:



(Q/K in question) (2021, Spring)

Q4. Design a TM to check a string, ω ∈ (a,b)* is palindrome or not.

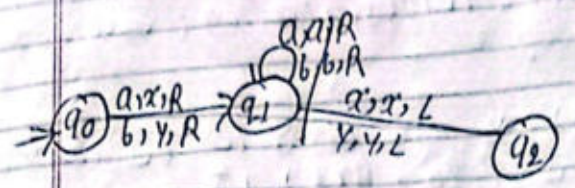
Soln: ω = abba abba

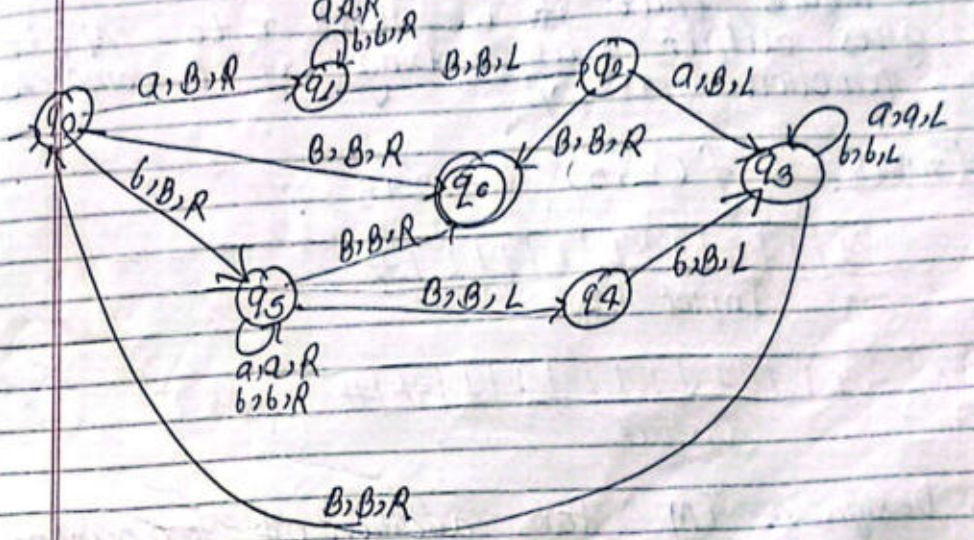
input tape

B	a	b	b	a	B

B	a	b	b	a	B

a	b	a	B
B			





Computing function

* Number Representation:
  Decimal : 5
  Binary : 101
  unary : 11111 or 00000

- we prefer unary representation; easier with TM.

Defination
a function $f(x) = y$ is said to be computed by TM if $\&(q_0, BxB) \vdash^* (q_0, ByB)$

it means that if we input x to TM, it gives output y as string if it compute function $f(x) = y$

Example: $f(2,5) = 2+5 = 7$

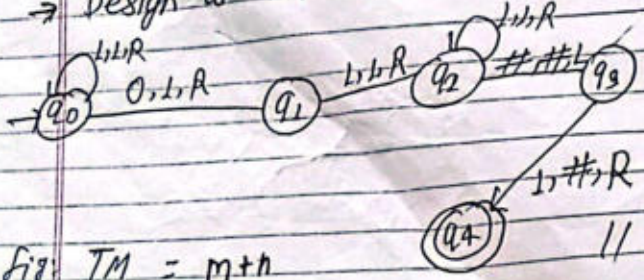
1	1	0	1	1	1	1	1	#

Input tape

1	1	1	1	1	1	1	#	#

Output

→ Design a TM for addition of two numbers.



1 skip, 0 मार्क 1
1 मार्क skip,
मार्क
1 मार्क #

fig: TM = m+n

// fig: n+1

(in exam)

Q. Design a TM for $f(n) = n+1$

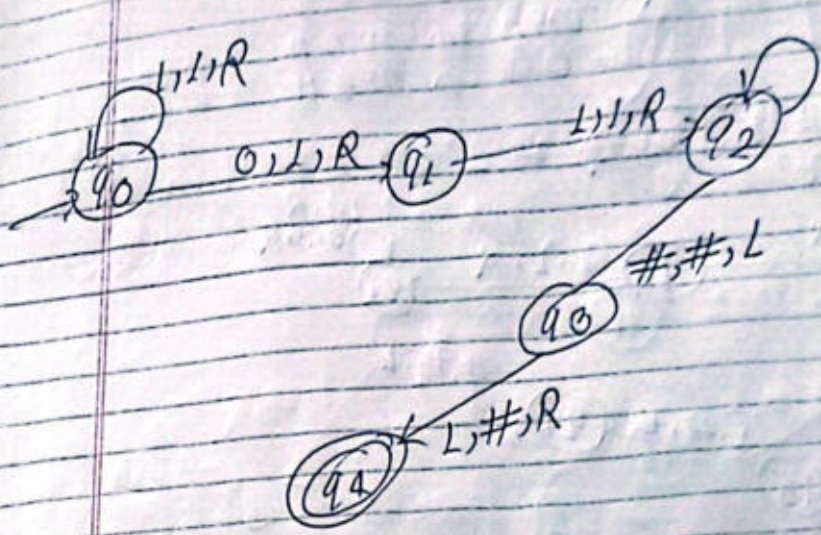


fig: TM for n+1

input: | 1 | 1 | 0 | # | 1 | # |

input

output | 1 | 1 | 1 | 1 | # |

$(q_0, 1101) \vdash 1q_0101 \vdash 11q_001 \vdash 110q_11 \vdash 1101q_1 \vdash 111q_2\#$

$(q_0, 1101\#) \vdash 1q_01101\# \vdash 11q_1101\# \vdash 111q_21\# \vdash 1111q_3\#$

$\vdash 111\#q_4\#$

$\vdash 111\#$ Hence, accepted.

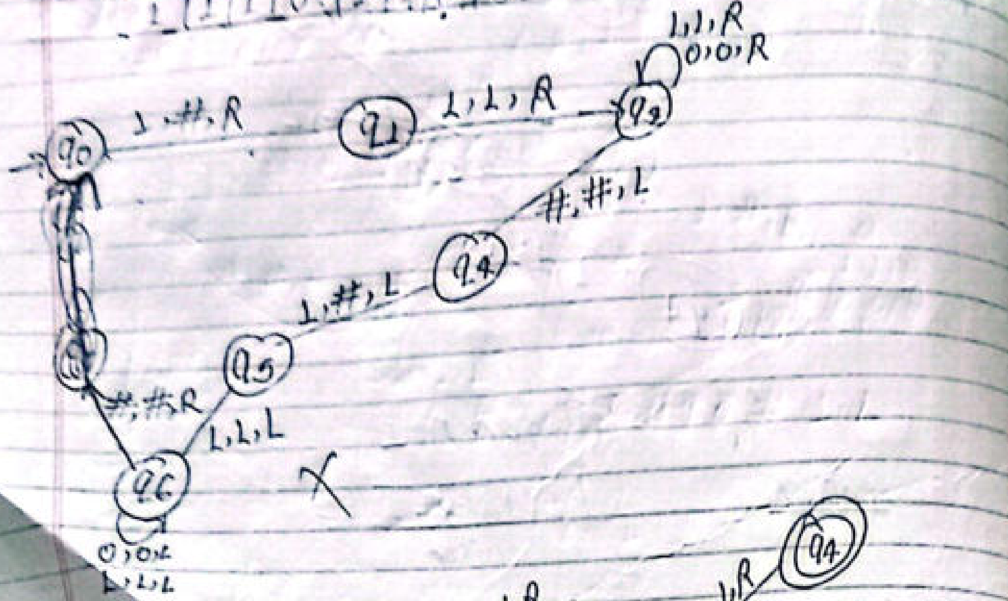
3,12

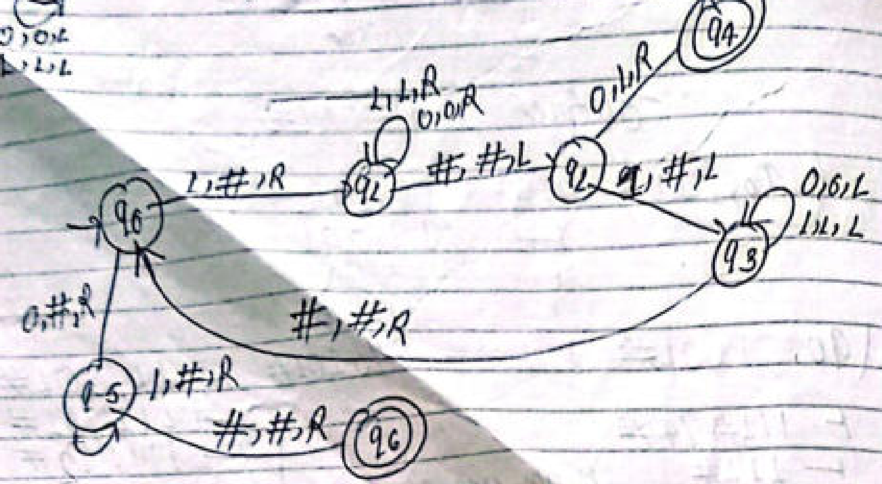
TM for Subtraction :

$f(m,n) = \begin{cases} m-n & \text{if } m > n \\ 0 & \text{if } m \le n \end{cases}$

$f(3,2)$

1 1 1 0 1 1 # #





(ask in exams) (1 imp.) 2019, spring

Q: Design a TM as a right shift machine that transform #w# to ##w# with Σ : {a, b, #}

Solⁿ: w = # | a | b | #

input tape

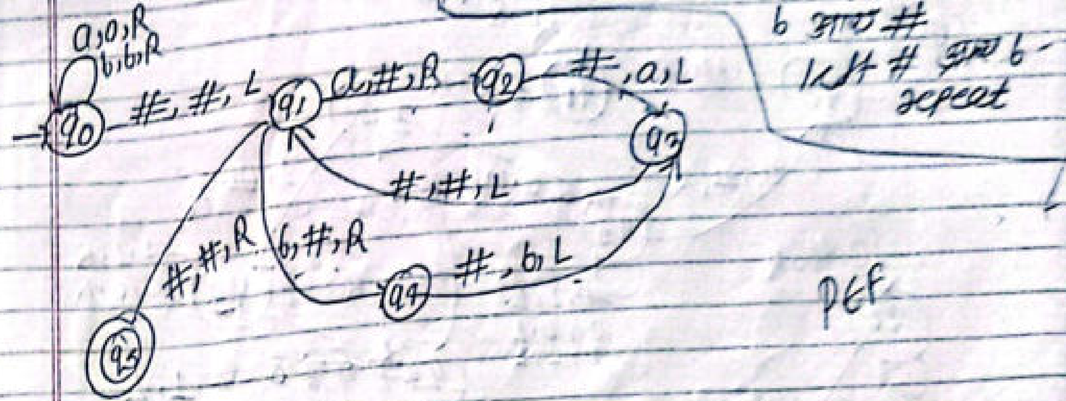
output: # | # | a | b | # |

skip a, a, reach # go left

b आया #

left # आया b

repeat

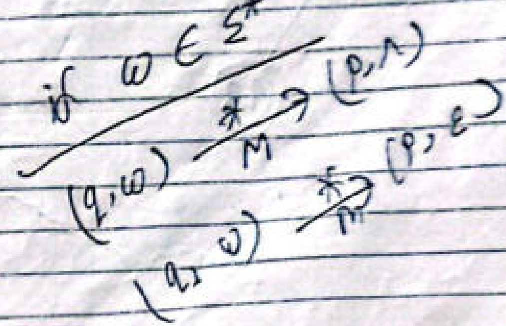


PEF

q0 - q1

a, b as it

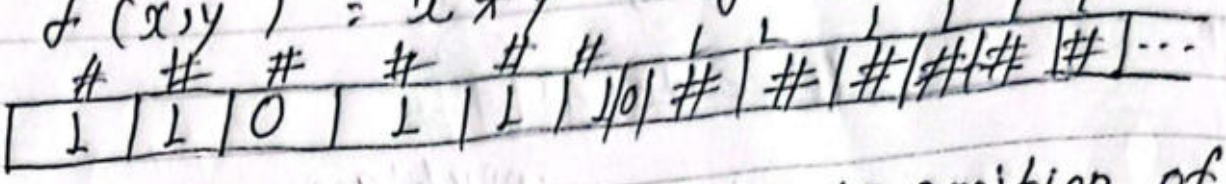
move left if come #



Date:
Page:

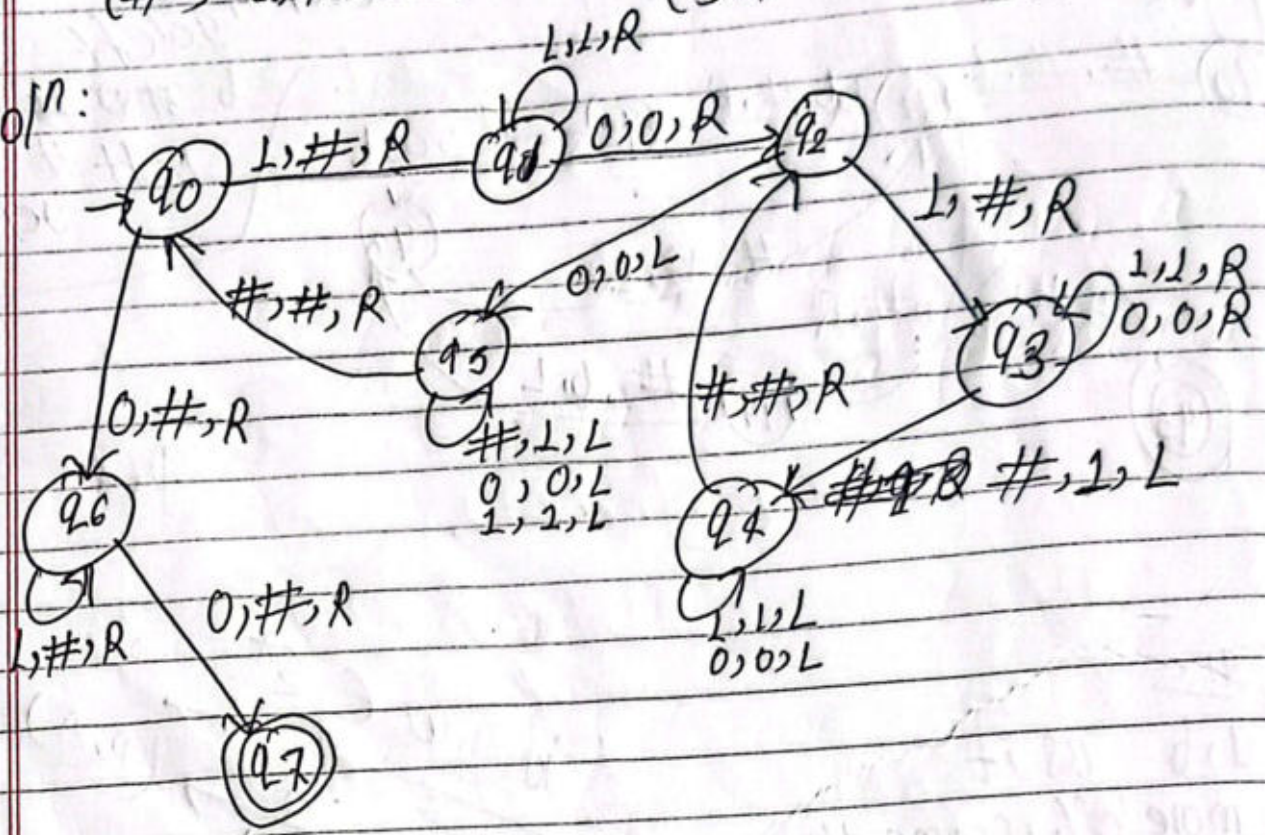
Q: Construct TM for multiplication.

$f(x,y) = x * y$ eg: 2 * 3



for 1st 1, copy 3 1's in position of blank (#) after 0. (skip 1)

Soln:



Assignment:

Q1. Design TM for :

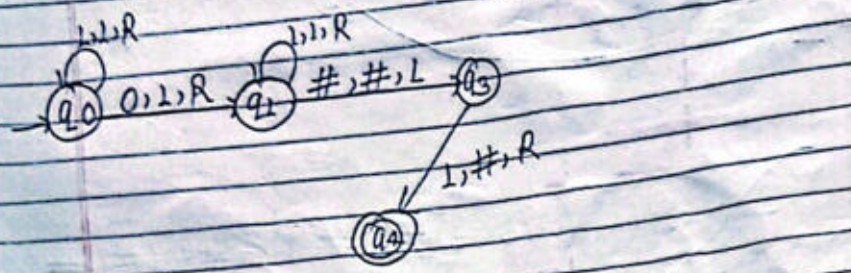
$f(n) = n+1$

$f(n) = n+2$

① $f(n) = n+1$

Input: 1 1 0 1 # (n = 2)

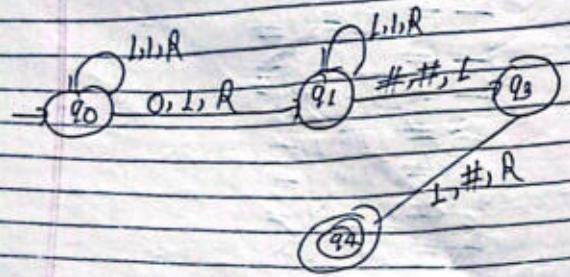
Output: 1 1 1 # #



2019 Spring

⑪ $f(n) = n+2$

input: 1 1 0 1 # # (n = 2)



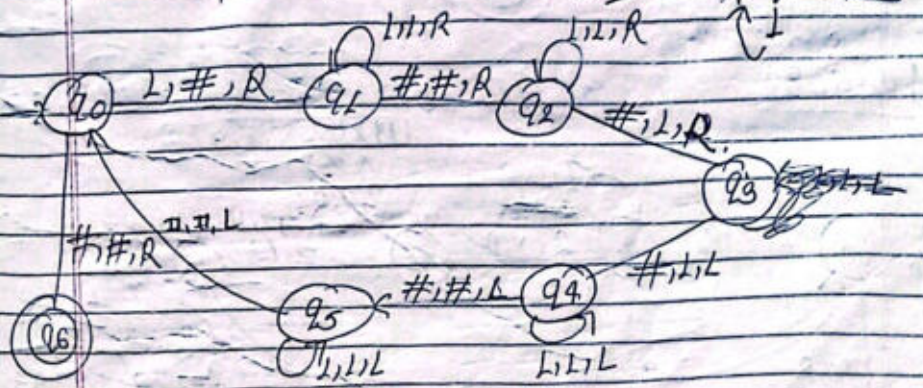
let, we simulate for 1101#

$(q_0, 1101\#) \vdash 1q_0101\# \vdash 11q_001\# \vdash 110q_01\# \vdash 1101q_0\# \vdash 1101\#q_1 \vdash 1101q_2\# \vdash 110q_21\# \vdash 11q_211\# \vdash 1q_2111\# \vdash q_21111\# \vdash 1111q_3\# \vdash 111q_4\#$

Hence, accepted.

Q3. Construct TM for $f(n) = 2n$

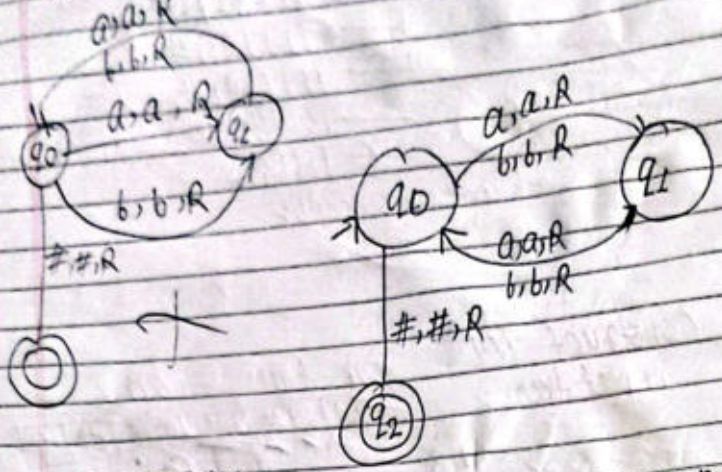
input tape: # 1 1 # 1 1 # # # #



2021 Pschand

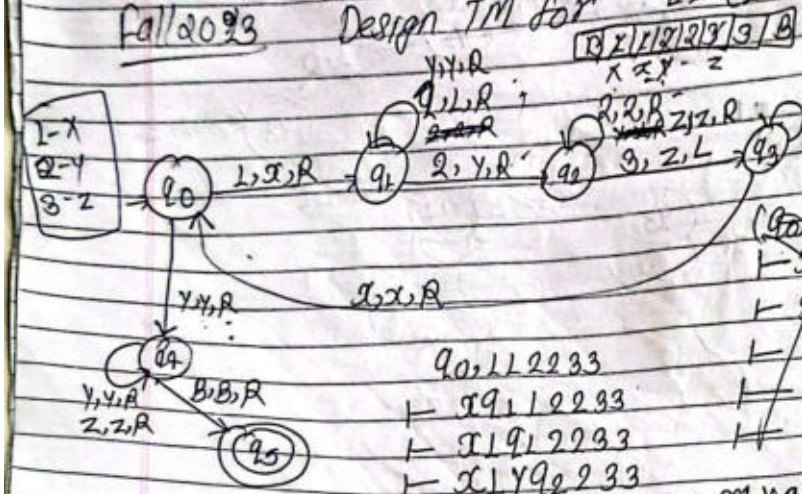
5a) Construct a Turing machine that accepts the language of strings over {a,b} with each string of even length. Also show it accepts string abab

Sol^n:



2018, Spring

Fall 2023 Design TM for $L = \{1^n 2^n 3^n, n \geq 0\}$

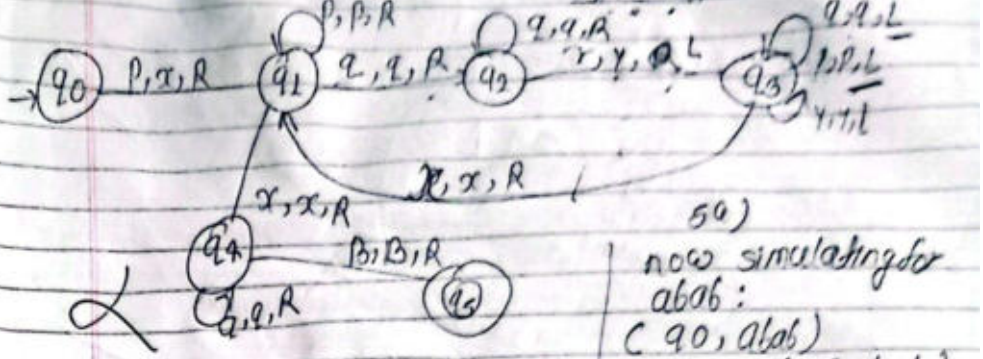


$q_0, 1122 33$
$\vdash X q_1 1 22 33$
$\vdash X 1 q_1 22 33$
$\vdash X 1 Y q_2 2 33$
$\vdash X 1 Y 2 q_2 33$
$\vdash X 1 Y q_3 2 Z 3$
$\vdash X 1 q_3 Y Z 3$
$\vdash X q_3 1 Y Z 3$

$\vdash X 1 q_3 2 Z 3$
$\vdash X q_3 1 2 Z 3$
$\vdash X X q_1 2 Z 3$
$\vdash X X 2 q_2 Z 3$
$\vdash X X 2 Z q_2 3$
$\vdash$

2019 Fall

Design TM for $L = \{ p^n q^m r^n : m, n \geq 0 \}$

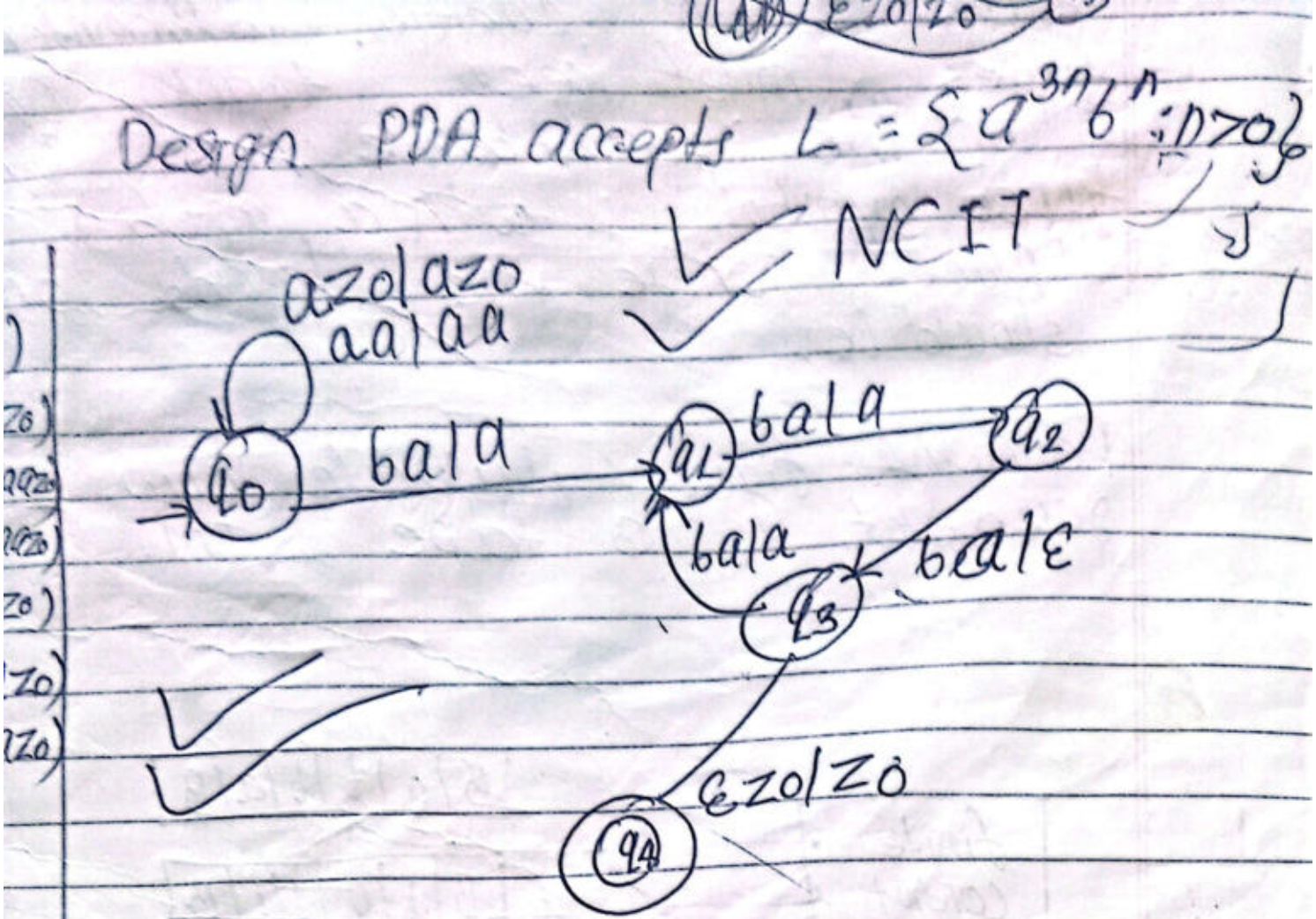


50)
now simulating for abab:
$(q_0, abab)$
$\vdash (a q_1 bab)$
$\vdash ab q_2 ab$
$\vdash ab a q_3 b$
$\vdash ab q_4 ab \#$
$\vdash ab q_2 b \#$
$\vdash aba q_2 \#$
$\vdash abab q_2$
Halt, Hence, accepted.

2014 Spring Types of TM:

a) Multi-tape
b) multi-head
c) non-deterministic
d) K-dimensional TM
e) Universal TM
f) Restricted TM

Design PDA accepts $L = \{a^{3n} b^n : n \geq 0\}$



Multi-tape Turing Machine

Cont' TM.

Theorem: for any K-tape (multiple), there is an equivalent single tape TM.

Proof: Let, $K >= 1$, be any K-tape TM. The key idea is to convert multitape TM in single tape TM.

Let, M has K-tape, then simulate the effects of K tape by storing these information on a single tape.

It uses new symbols as a delimeter to seperate the content of different tape. In addition to content of those tapes, must keep track of the location of the read/write head by writing a tape symbol with a dot(.) to make the place where the heads on that tape should be.

Following figure illustrate how 1 tape use to represent 3 tape TM.

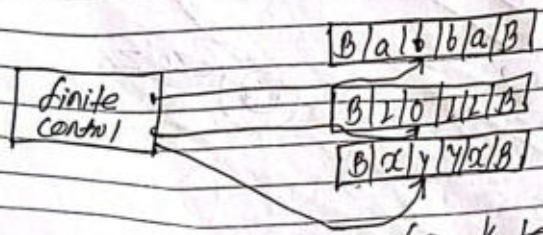


fig: K-tape (K=3)

abbbba

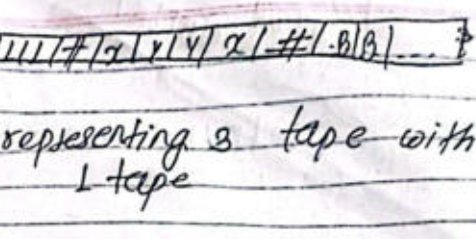


fig: representing 3 tape with 1 tape

To simulate a transition, from state q, we must scan our tape to see which symbols are under K-tape head, i.e. scan the dots.

Once we determine this and are ready to make the transition, we must scan across the tape again to update the cell and move the dots.

→ Whenever one heads move to the right end, we must shift our tape, so we can insert blank symbol.

Cont'

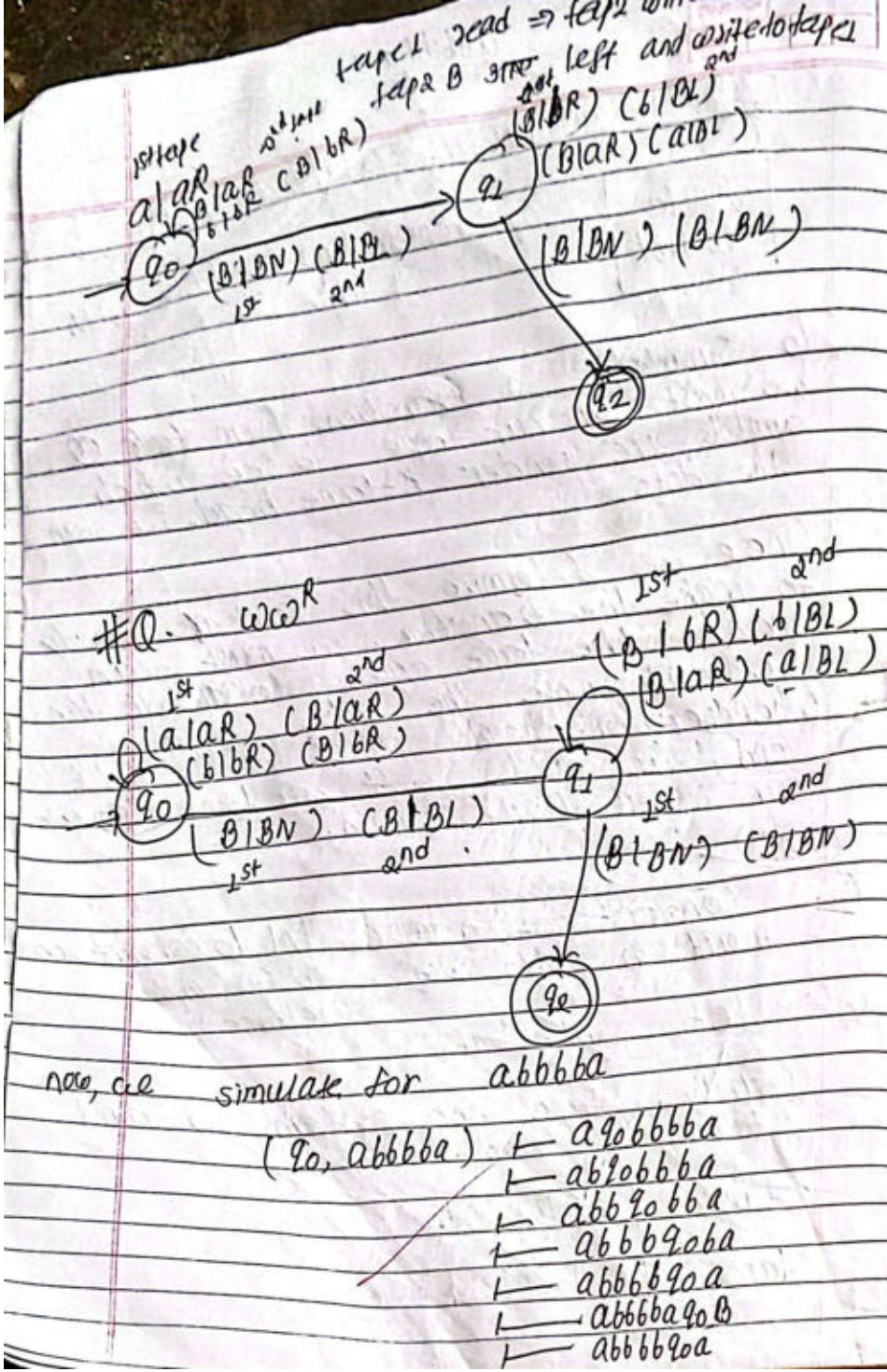
Q. Construct a two tape TM to convert an input w into ww^R, $w \in (a+b)$ (reverse)

Soln: let, w be 'abb'

Initially, tape one contains w and tape2 is blank.

tape1: B a b b B ...
H1

tape2: B B B B B ...
H2



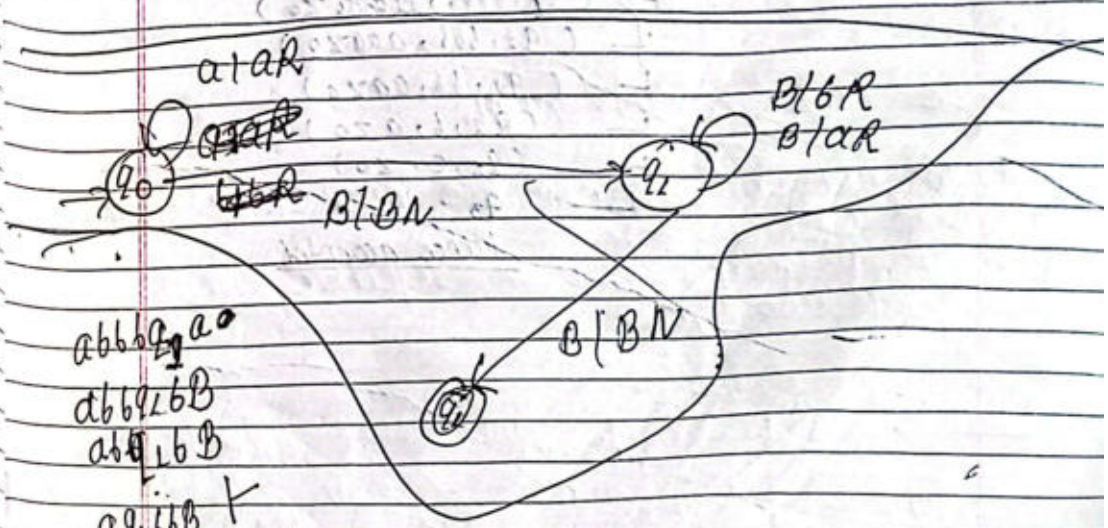
#Q. WcW^R

now, we simulate for abbbba

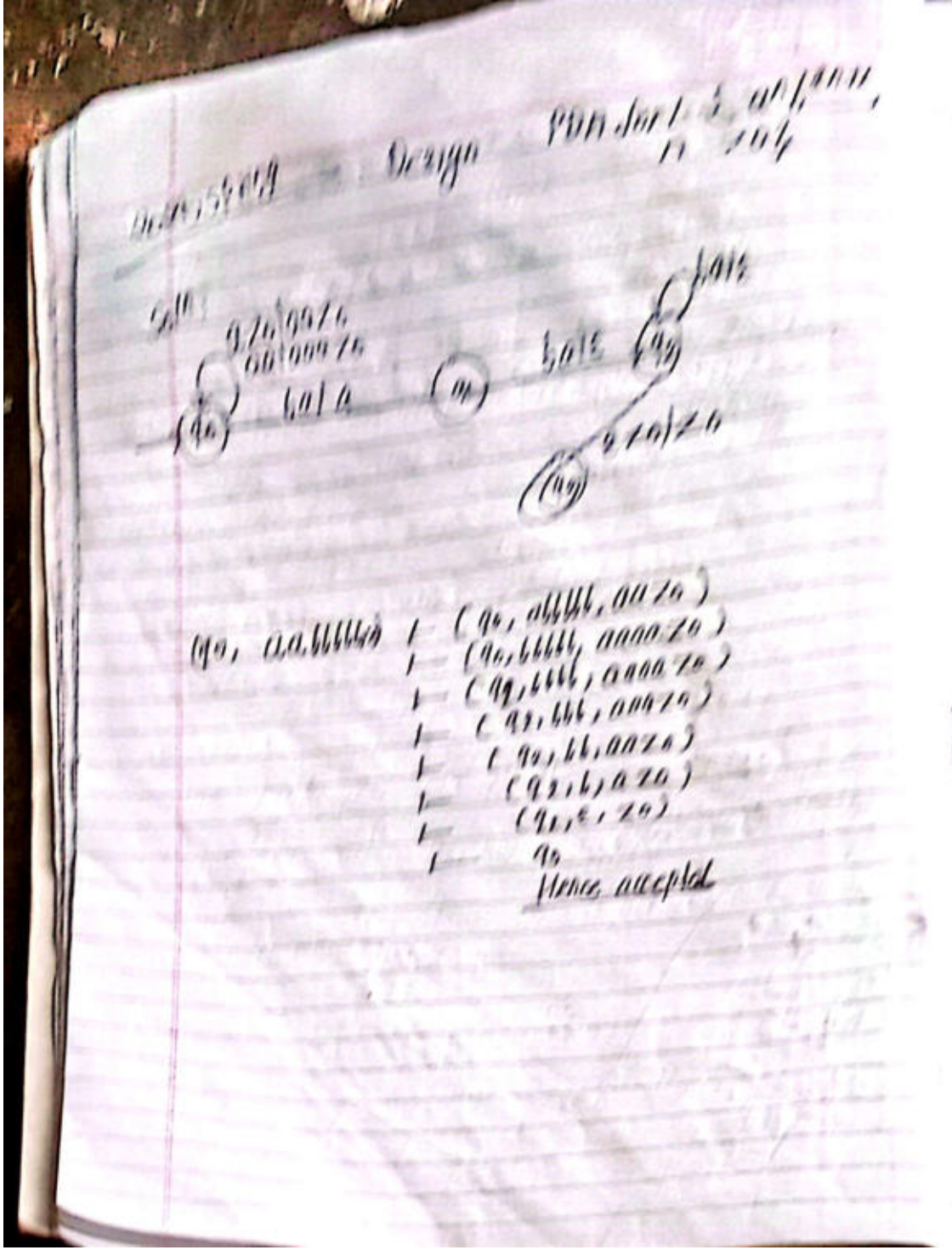
$(q_0, abbbba) \vdash aq_0bbbba$
$\vdash abq_0bbba$
$\vdash abbq_0bba$
$\vdash abbbq_0ba$
$\vdash abbbbq_0a$
$\vdash abbbbaq_0B$
$\vdash abbbbq_0a$

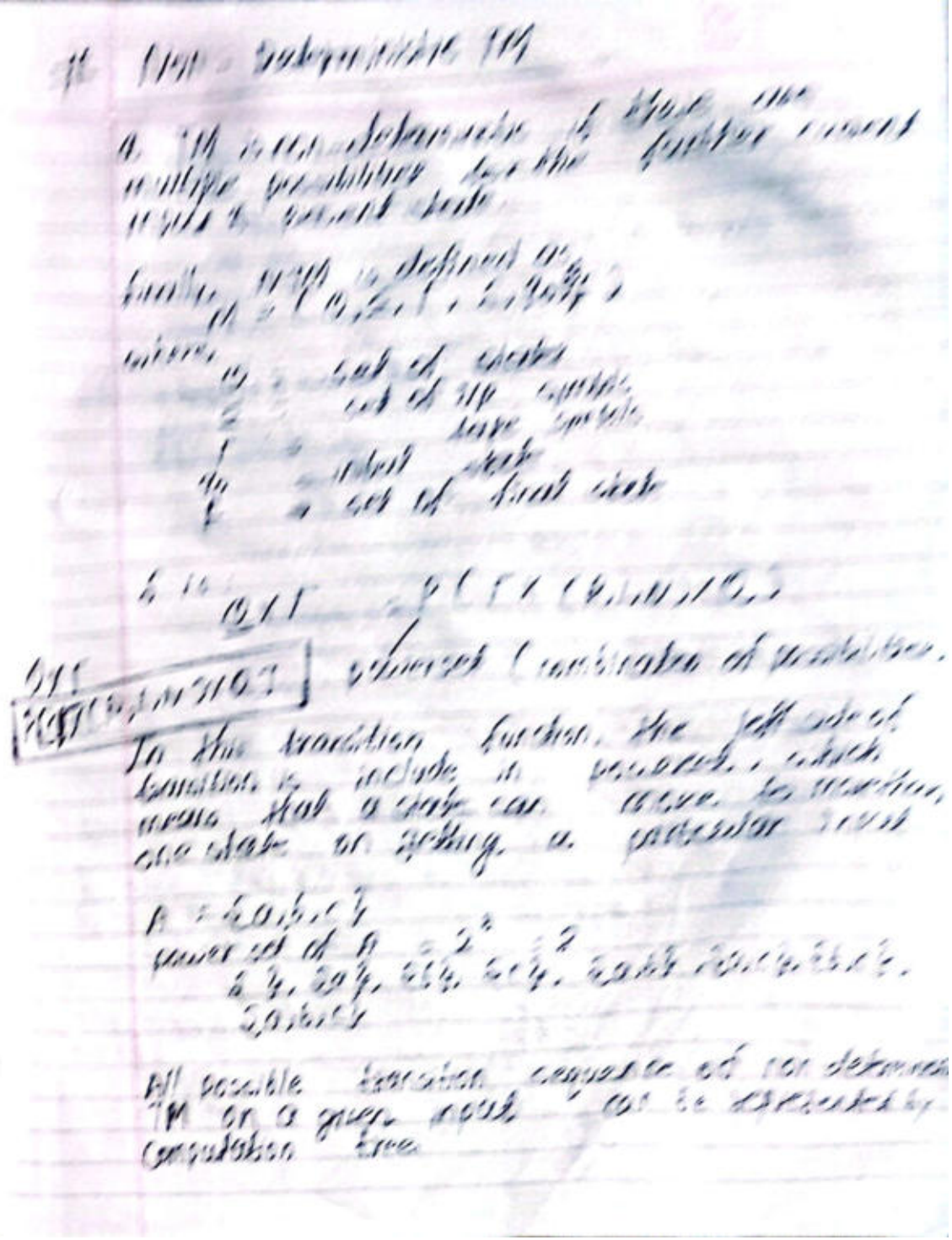
Q(iii)
→ Why TM is called ultimately calculating mechanism.

Soln: → most fundamental and powerful model of computation.
→ UTM can simulate other TM can perform any computation that any algorithm can solve do.
- Aligns with Church-Turing Thesis.
- Simple design (infinite tape, read/write head, finite set of states and transition rules).
- every today's computer is essentially advance form of TM.
- defines what is computable or not



abbbbq1a
abbq1bB
abq1bB
aq1bbB
q1aBbB





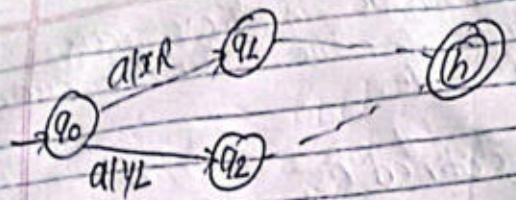
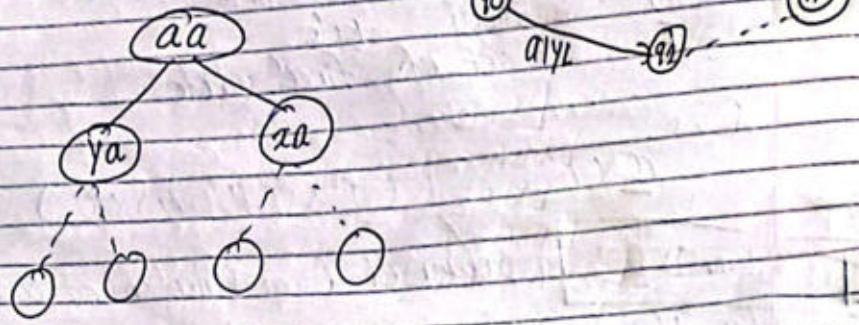


fig: NTM

We draw computation tree for string 'aa',

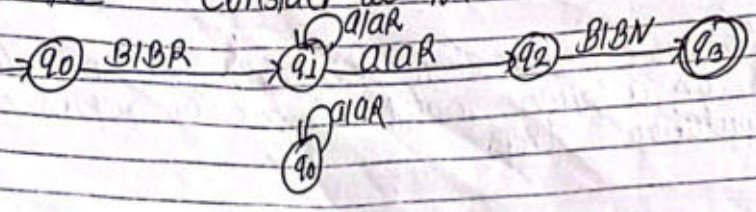


The outcome of NTM,

→ if any branch accept the input string, then NTM also accepts.

→ if all branches halt or reject the input string the NTM also rejects.

Example: Consider a NTM that accepts a^+. $\Sigma\{a, aa, aaa, \ldots\}$



Unrestricted Grammar:

An Unrestricted Grammar is most general in Chomskey hierarchy. It is called unrestricted because no other restriction is made on this except each of layer left hand side of the rule can contain terminal & non-terminal, but the condition is at least one of them most be non-terminal.

Each of production is of the form,

$$\alpha \rightarrow \beta$$

where,

α is $(V \cup T)^* V (V \cup T)^*$

β is $(V \cup T)^*$

formally, Gramma is,

$G = \{V, T, P, S\}$

(non-terminal, terminal ($\leq$))

The language genesated by unrestricted grammar is called Recursively Enumerable language. (REL).

The REL is accepted by TM.

example: Given grammar for $L = \{a^{2n} \mid n \geq 0\}$

production is,

$S \rightarrow TaU$

$U \rightarrow \varepsilon | AU$

$aA \rightarrow Aaa$

$TA \rightarrow T$

$T \rightarrow \varepsilon$

$S \rightarrow TaU$

$T \rightarrow a$

$U \rightarrow \varepsilon$

Derive a^4.

$S \to TaU$
$\to EaU$
$\to aAa$
$\to Aaaa$

$S \to TaU$
$\to TaAU$
$\to TAaaU$
$\to TAaaAU$
$\to TaaaAU$
$\to EaaaaE$
$= aaaaa$

$S \to TaU$

$S \to TaU$
$\to TaAU \quad [U \to AU]$
$\to TAaaU \quad [aA \to Aaa]$
$\to TAaaAU \quad [U \to AU]$
$\to TAaAaaU \quad [aA \to Aaa]$
$\to TAAaaaU \quad [aA \to Aaa]$
$\to TAaaaaU \quad [TA \to T]$
$= TaaaaU \quad [TA \to T]$
$= aaaa \quad [T \to E, U \to E]$
$\to aaaa \#$

Q. Given grammar for $L = \{a^n b^n c \mid n \geq 0\}$ production is,

$S \to UT$
$U \to E \mid aUbC$
$Eb \to bC$
$CT \to Tc$
$T \to E$

Derive $a^3b^3c^3$.

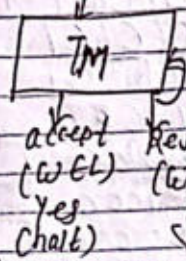
Sol: $S \to UT$
$\to aUbCT \quad [U \to aUbC]$
$\to aaUbCbCT \quad [U \to aUbC]$
$\to aaaUbCbCbCT \quad [U \to aUbC]$
$\to aaabCbCbCT \quad [U \to E]$
$\to aaabbCbCCT \quad [Cb \to bC]$
$\to aaabbbCCCT \quad [CT \to Tc]$
$\to aaabbbCCTc \quad [CT \to Tc]$
$\to aaabbbCTcc \quad [CT \to Tc]$
$\to aaabbbccc \quad [T \to E]$
$\#$

bCbC
bCCbC
bCbCCT
bCbC
TC
bCCbC
bC

Recursively Enumerable Language (REL)

- Language generated by unrestricted grammar.
→ Language accepted by TM.

Definition: A language 'L' is said to be recursively EL if there exists a TM which will accept an halts for all the input strings which are in language L but may or maynot halt for all the input string which are not in L. It means that TM can loop forever for the string which are not in L.



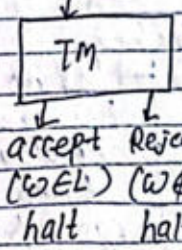
It is also known as type 0 or Turing Recognizable language.

- All regular, context free and recursive languages are recursively enumerable language.

TM P

Recursive Language (RL)

- A language 'L' is said to be recursive if there exists a TM which will accept all the strings in L and reject all the strings not in L.



A TM will halt every time and give an answer (accepted or rejected) for each & every input strings.

for example:

$L = \{a^n b^n c^n \mid n \geq 1\}$ is recursive language because we can construct a TM which will move to final state if the string is of the form $a^n b^n c^n$ else move to nonfinal state. So TM will halt in this case. which is also known as 'Turing Decidable language.

Units

Undecidability (4hr = 8marks)

Problem

Solvable Unsolvable

Yes / No → decidable | undecidable

decidable: has algorithm, it will stop in finite time

undecidable: has no algorithm, but has method to solve problem, time is not fix. Eg: halting problem (imp)

Decidable Language

A language 'L' is decidable if it is recursive language. All decidable language are recursive & vice versa. RL, TM → decidable

Partially Decidable Language

A Language 'L' is partially decidable language if it is Recursively Enumerable Language. REL, TM → recognizable

Undecidable Language

If a language is not even partially decidable then there exists no TM for that languages.

Decidable Problem:

A problem is decidable if we can construct a TM which will halt in finite amount of time for every input and give answer as yes or no. A decidable problem has an algorithm to determine the answer for the given input.

Example: equivalence of two regular language,
finiteness of regular language,
emptiness of context free language.

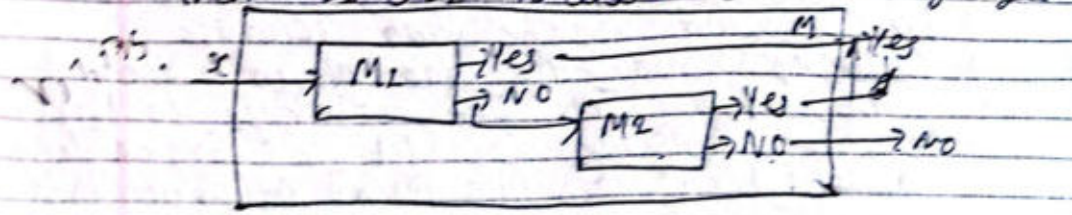
Undecidable Problem

A problem is undecidable if there is no TM which will always halt in finite amount of time to give answer as yes or no. An undecidable problem has no algorithm to determine the answer for given input.

Eg: ambiguity of Context Free Language,
equivalence of two CFL

Properties of Recursive Language: (imp)

1) Union: If L_1 and L_2 are two recursive languages then $L_1 \cup L_2$ is also recursive language



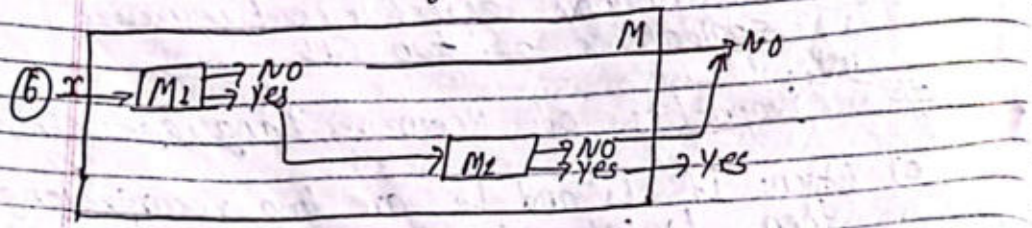
Let, L_1 and L_2 be two recursive language and M_1 and M_2 be corresponding TM that halt. Let $x \in (L_1 \cup L_2)$ be a string to M. We design $M = M_1 \cup M_2$ such that

→ 1. if M_1 accepts x then M accepts x.
→ 2. if M_1 rejects x then x is passed to M_2.
3. if M_2 accepts x then M accepts x.
4. if M_1 and M_2 both rejects x then M rejects x.

Thus $L_1 \cup L_2$ is also a recursive language.

Properties of Recursive Language (TM)

b) Intersection: If L_1 and L_2 are two recursive languages, then $L_1 \cap L_2$ is also recursive language.

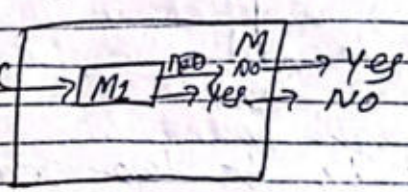


Let, L_1 and L_2 be two recursive language and M_1 and M_2 be their corresponding TM. Let x belongs to $L_1 \cap L_2$ be a string to M. So we design a TM, $M = M_1 \cap M_2$. such that

1. if M_1 rejects x then M rejects x
2. if M_1 accepts x then pass x is passed to M_2.
3. if M_2 also accepts x then M also accepts x.
4. if M_2 rejects x then M also rejects x.

Thus, $L_1 \cap L_2$ is also recursive language.

c) Complement: If L_1 is recursive language, then $\overline{L_1}$ is also recursive language.



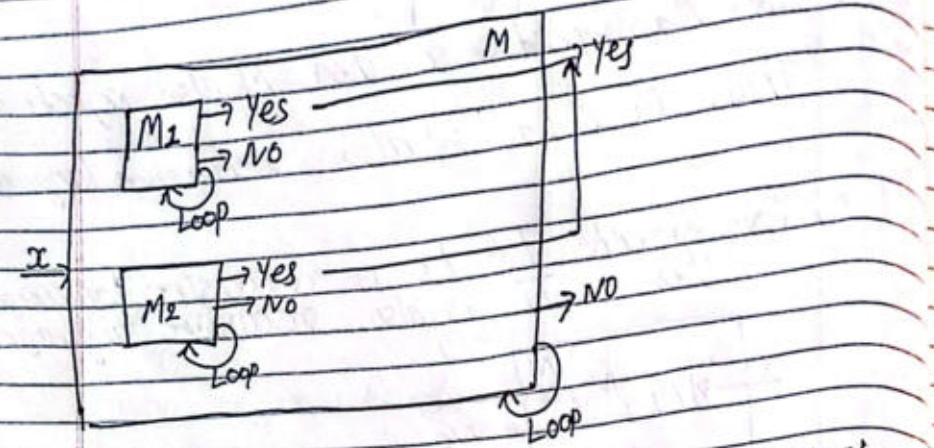
Let, L_1 be a recursive language and M_1 be corresponding TM. Let $x \in L_1$ be a string to M. So we design a TM, $M = \overline{M_1}$, such that

1. if M_1 rejects x then M accepts x
2. if M_1 accepts x then M rejects x.

Thus, $\overline{L_1}$ is also recursive language.

Properties of Recursively Enumerable Language

① Union: If L_1 and L_2 are two recursively enumerable language, then $L_1 \cup L_2$ is a recursively enumerable language.

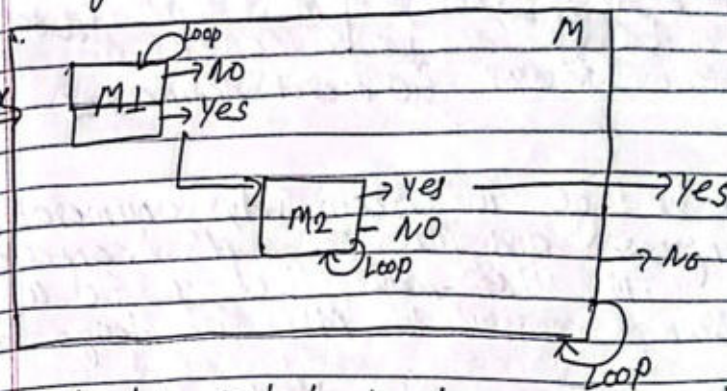


Let, L_1 and L_2 be two recursively Enumerable language. And M_1 and M_2 are their corresponding TM that may or maynot halt. Let $x \in (L_1 \cup L_2)$ be a string to M, then we design a TM

$M = M_1 \cup M_2$, such that

1. if M_1 accepts x then M accepts x.
2. if M_2 accepts x then M accepts x.
3. if M_1 and M_2 reject or goes infinite loop, then M may reject or may goes infinite loop.

② Intersection: If L_1 and L_2 are two recursively enumberable language, then $L_1 \cap L_2$ is also recursively enumerable language.

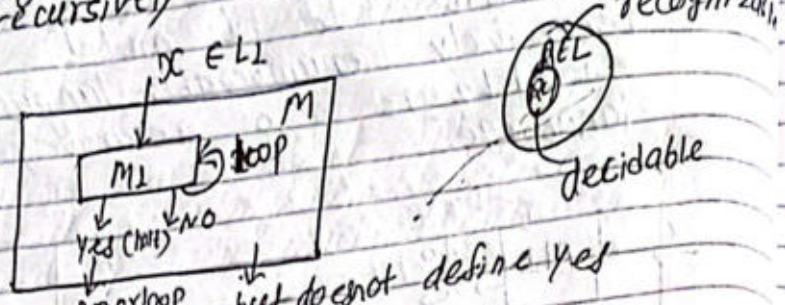


Let, L_1 and L_2 be two recursively enumerable language and M_1 and M_2 are their corresponding TM that may or maynot halt. Let $x \in (L_1 \cap L_2)$ be a string to M, then we design a TM, $M = M_1 \cap M_2$, such that:

1. if M_1 accepts x then it is passed to M_2.
2. if M_2 accepts x then M also accepts x.
3. if M_2 rejects x or goes infinite loop then M rejects or goes infinite loop.
4. if M_2 rejects x or goes infinite loop, then M may reject or may goes infinite loop.

Complement

(iii) If L_2 is a recursively enumerable language then L_1 is not recursively enumerable language.



Let L_1 be a recursively enumerable language and M_1 be it's corresponding TM. that may Let x be a string passed to M. We design a TM $M = \overline{M_1}$, such that:

a) if M_2 accepts x and halt then M may reject or goes on infinite loop. M_1 accepts

b) if x does $\notin L_1$ then M_2 may goes on loop or may rejects, for this we can't decide accept state in TM M, i.e. can't design $M = \overline{M_1}$.

So $\overline{L_1}$ is not recursively enumerable language.

Church-Turing Machine Thesis (Page 24)

In 1936, two scientists gave papers on definition of algorithm.

Alonza Church (Lambda Calculas)	Alan Turing (Turing Machine)
→ He said that, we can design Lambda cuculos for any problem is solvable / computable (i.e. which has algorithm)	→ He said that, we can design TM for any problem that is computable (i.e. which has algorithm)

It turns out that both definition are equivalent.

Alonza Church

The connection between the informal concept of algorithm and formal definition is called Church-Turing Thesis.

informal concept of algorithm

Church-Turing Thesis which states that given any problem that can be solve within effective algorithm, these is a Turing Machine that can solve the problem. If we don't consider algorithm, then these doesn't exist TM.

the connection between the inform.

Nothing will be consider an algorithm if it cann't be rendered as TM. It is universally accepted by computer scientists said that TM is a mathematical model

of an algorithm. This thesis is just a hypothesis, which is not proven and doesn't consider wrong by anyone. It means nobody prove this hypothesis wrong till today.

The thesis implies that TM model is universal, any algorithm that can be performed by any machine can also be performed by TM.

Halting Problem : (v.imp)

Given a program, will it Halt?

→ we can't design a TM for this.
→ Undecidable problem.
→ proposed by Alan Turing

Defination : Halting problem is the problem of finding or determining whether the program will finish running or continue to run forever, form a description of an arbitary computer program and input.

~~arbitary~~

form a description of an arbitary computer program and input.

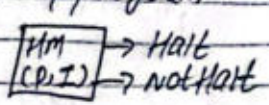
Proof by Contradiction

problem statement: Can we design a machine which if given a program can find out if that program will always halt or not halt.

Solution:

1. Assumption

Suppose we can design ~~search~~ such a machine, called HM (P, I) where
→ HM : Hypothetical Machine / program
→ P : given program
→ I : input



when, provided with these inputs, HM will determines whether program P halt or not halt.

2. Constructing Contradiction

Using HM, we can create another program called CM (X), where X is any program passed as input.

```
CM (X) {
  if (HM (X,X) == Halt)
      forever loop (not halt)
  else
      Halt
}
```

TM = {Q, Σ, δ, s}

3. Contradiction:

Now, consider what happens if we pass CM itself as an argument.

```
CM (CM)
{
  if (HM (CM, CM) == Halt)
    forever loop (not halt)
  else
    Halt
}
```

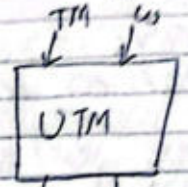
i) if HM predicts that CM(CM) halts, CM will go on forever loop.
ii) if HM predicts CM(CM) doesn't halt, CM will halt.

in both cases we face contradiction. Therefore our initial assumption that such a machine HM could exist must be false. Hence, we can conclude that halting problem is undecidable. No general algorithm can determine whether every program will halt or not.

Universal Turing Machine

A TM is said to be universal if it accepts
- input data
- description (algorithm) for computing

(TM must be pass no only ∞)



- Infinite memory (infinite input tape)
- if problem has TM, it also has UTM.

→ UTM is recognizer, not decider.

Assignment

Q. why UTM is called recognizer, not decider?

Soln: UTM is called recognizer because it can accept strings belonging to a language but it may not halt or reject for all inputs, including strings not in the language.

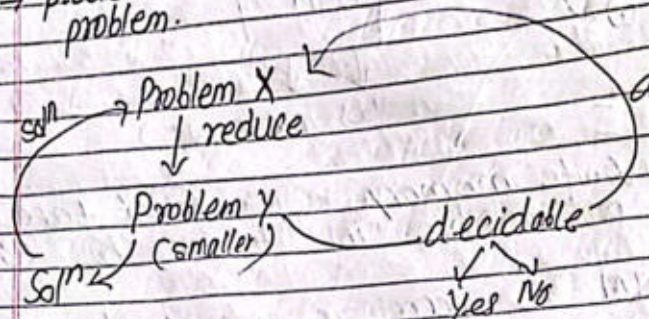
UTM is a TM that can simulate an arbitary TM on arbitary input.

Assignment

Undecidable Problem about TM (Halting problem)

↳ No TM to decide 'yes' or 'No'.
- no algorithm.

* Reduciability:
→ problem that can be reduced to smaller problem.



→ if X is undecidable, then problem Y is undecidable.

→ An undeciable problem is one for which no TM can be constructed that will always correctly determine, in finite time, whether any given instance of the problem is a yes or no

→ Halting problem is also an undecidable problem

eg: emptiness problem, equivalence, totality problem etc.

Unit 6

Undecidable problems mostly contain Recursive Enumerable Language.

Reducibility (imp) 2q. Pattern)

Reducibility is a fundamental concept in computability and Complexity theory that helps compare the difficulty of problems
- if problem A can be reduced to problem B, it means that an algorithm for B can be used to solve A.
- How one problem can be transformed to another problem

emptiness
equivalence

emptiness problem, equivalence, tota

reducibility is a fundamental concept in computability and complexity theory

Unit 6 Computational Complexity Theory (4hr = 8mar)

computable? → which has TM.

Complexity Theory
→ Time complexity
→ space complexity

imp How efficiently problem are solvable?
Difference b/w (I) & (II).

* Problems are categorized on the basis of time complexity:

shortnote (imp)
(I) Tractable: solve in finite time, polynomial time (has TM)
(II) Intractable: solve in infinite", exponential" (has non-deterministic TM)
(factorial time) (non-polynomial time)
polynomial time : $O(n), O(n^2)$
exponential time : $O(2^n), O(n!)$ (not deterministic TM)

Tractable example: addition, sorting
recursive language

Intractable: Chase problem, Recursively Enumerable Language (undecidable problem)

2a+call
imp Computability: describes which problem can be solve and which has algorithm.

Complexity Classes: (imp)

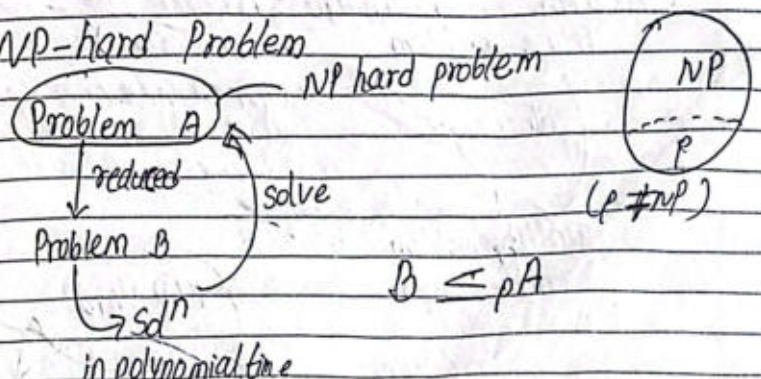
// Polynomial Time

Problem are classified on the basis of easy and hard.

(I) P class Problem: easy → can be solved in polynomial time. (Deterministic Polynomial Time) decidable Tractable.
(II) NP class Problem: hard. (Non-Deterministic Polynomial time) Intractable.
Solvable (NTM)
can be make DTM, but it can only verify solution. (in polynomial time)

NTM, that can solve problem in polynomial time.

a) NP-hard Problem



eg: TSP (Travelling Salesman Problem), Hamilton path (cyle baneko)
possibility: $\frac{(n-1)!}{2}$

Boolean Satisfiability Problem (SAT)

→ P vs NP problem

b) NP-Complete (Q4, Fall)
if both NP hard and NP problem.

TSP → Hamilton solⁿ (solⁿ)
↓ solⁿ
NP hard
→ NP

	decidable	verify
P	✓	✓
NP	X	✓

eg: SAT (3-circuit SAT)
Tiling problem (undecidable)
~~यानि~~ undecidable हो।
(reduced)
↓
Halting problem (undecidable) हो सके

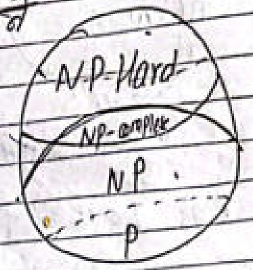


fig: NP-Complete problem
P vs NP

Assignment

Case study 1: Tiling problem is undecidable

1. Introduction:

The Tiling Problem (Domino Problem) asks whether a given finite set of tile types can cover an infinite plane without gaps or overlaps, while matching specified edge constraints. In 1966, Robert Berger proved that this problem is undeciable. No algorithm can determine, for all possible tile sets, whether a valid tiling exists. This result connects computation theory, geometry and mathematical logic, demonstrating how undecidability arises in non-computational domains.

2. Background and Definitions

2.1 The Tiling Problem:
Tiles - square tiles with labeled edges (e.g. colors, symbols).
Tiling Rules: Adjacent tiles must have matching edge labels.
Valid Tiling: An infinite covering of the plane respecting the rules.

2.2 Undecidability in Computation
- A problem is undecidable if no algorithm can solve all instances (e.g. the Halting Problem).
- Reduction: Proving undecidability by transforming one undecidable problem into another.

3. Berger's Proof: Reducing the Halting Problem to Tiling

3.1. High-Level-Idea
Berger showed that any TM can be encoded as a set of tiles such that:
- The TM halts → No valid tiling exists
- The TM runs forever → A valid tiling exists.

Since the Halting Problem is undecidable, the Tiling Problem must also be undecidable.

3.2 Key steps in the Proof
1. Tile Simulation of a TM
- each tile represents a TM state, tape symbol and head position.
- Adjacency rules enforce TM transitions.

2. Infinite Plane ↔ Infinite Computation
- A tiling corresponds to a TM's execution history.
- A halting state creates a local mismatch, breaking the tiling.

3. Undecidability Conclusion
- If we could decide whether a tile set admits a tiling, we could solve the Halting Problem which is impossible.

4. Implications and Applications
4.1 Limits of Algorithm
→ no general procedure exists to check arbitrary tile sets for tilability.
→ Undecidability extends to related problems in geometry and logic

4.2 Aperiodic Tilings and Physics
- Penrose Tilings: Aperiodic tilings (no repeating pattern) inspired by undecidability
- Quasicrystals: Real-world materials with aperiodic atomic structures (Nobel Prize 2011).

5. Conclusion
Berger's proof established the Tiling Problem as undecidable by reducing it to the Halting Problem. This result reveals deep connections between computation and geometry influencing mathematics, Physics and Computer Science

6. References

→ Berger, R. (1966). The Undecidability of the Domino Problem.
→ Sipser, M. (2012). Introduction to the Theory of Computation (3rd ed.)
→ Goodman-Strauss, C. (2020). Aperiodic Order and Tiling Theory

(Hamilton path)

Case study: TSP is NP-Complete Problem.

1. Introduction

The Traveling Salesman Problem (TSP) is one of the most famous and well-~~cite~~ studied problems in computer science and operations research. It asks:
Given a list of cities and the distances between them, what is the shortest possible route that visits each city exactly once and returns to the origin city?
In 1972, Richard Karp proved that TSP in NP-Complete, meaning:
- It belongs to the class NP (solutions can be verified quickly).
- It is at least as hard as the hardest problems in NP (any NP problem can be reduced to it).

This result has profound implications for optimization, logistics and computational complexity theory.

2. Background and Definitions

2.1. The Travelling Salesman Problem
→ Input: A weighted graph (cities as nodes, distances as edges).
→ Output: A Hamiltonian cycle (a closed loop visiting each node exactly once) with the minimum total weight.
→ Decision Version: Does a tour with total distance $\leq k$ exist?

2.2. Computational Complexity Classes
→ P: Problems solvable in polynomial time (e.g. sorting).
→ NP: Problems where solutions can be verified in polynomial time (e.g. TSP, Boolean satisfiability).
→ NP-Complete: The hardest problems in NP; solving one in polynomial time would solve all NP problems.

2.3. NP-Completeness Proof Requirements
To prove TSP is NP-Complete, we must show:
1. TSP $\in$ NP (a given solution can be checked quickly).
2. TSP is NP-hard (some known NP-complete problem reduces to TSP).

3. Proof that TSP is NP-Complete

3.1. TSP $\in$ NP (easy part)
Given a candidate tour, we can verify in $O(n)$ time:
- it visits every city exactly once
- it's total distance $\leq k$
Thus, TSP is in NP.

3.2 TSP in NP-Hard (Reduction form Hamiltonian Cycle)
Karp's proof reduces the Hamiltonian cycle problem (HCP) - a known NP-complete problem to TSP:

1. Hamilton Cycle Problem (HC):
Given an unweighted graph, does a cycle exist that visits every node exactly once?

2. Reduction Strategy:
Convert an HC instance (graph G) into a TSP instance:
→ cities = nodes of G
→ Distances:
• if an edge exists in G, set distance = 1
• if no edge exists, set distance = ∞
• set K = no. of cities (n)

3- Key Sight
- A Hamiltonian cycle in G exists ⇔ A TSP tour of length ≤ *n* exists.
- Thus, solving TSP would solve HC, proving TSP is at least as hard as HC

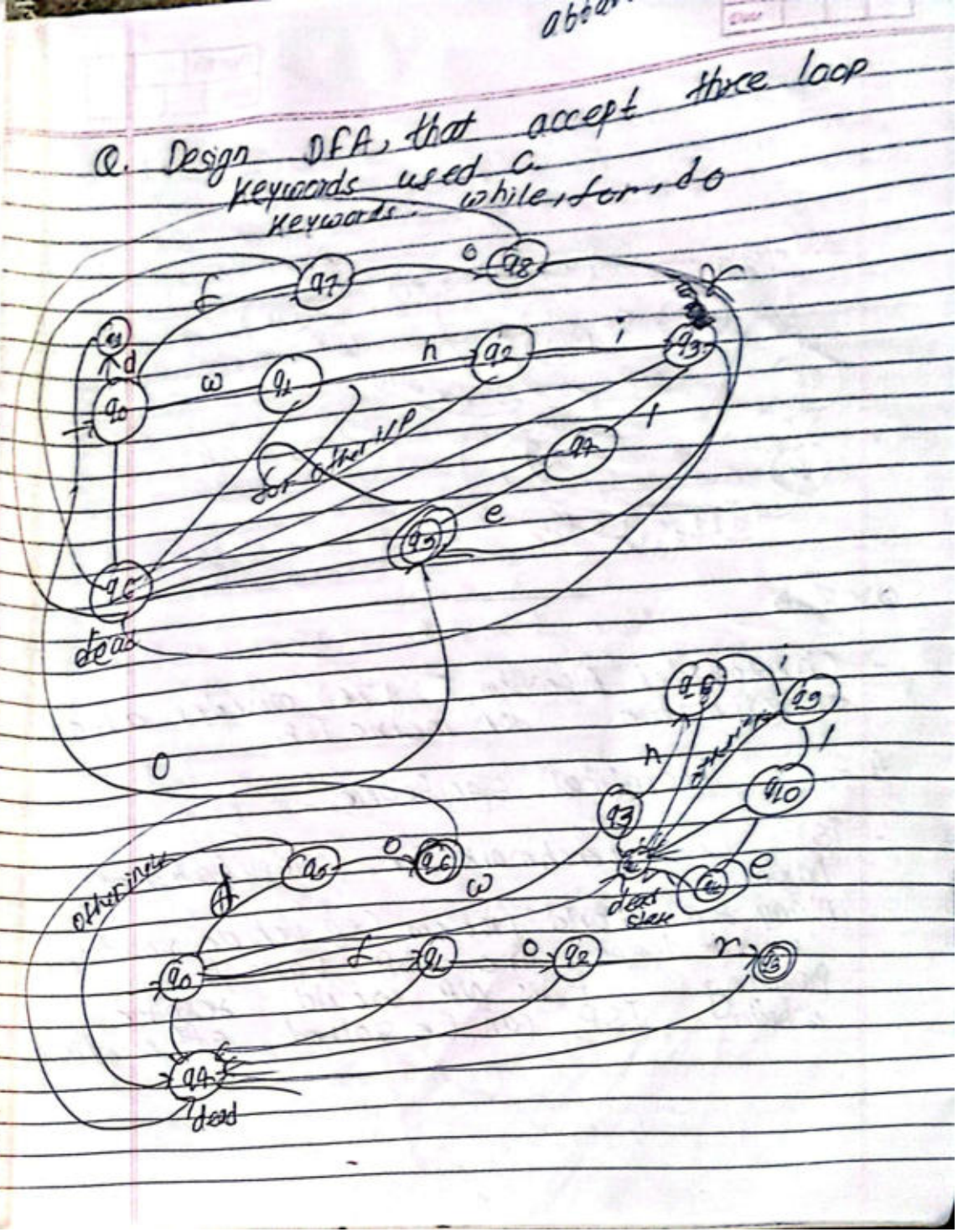
Since HC is NP-complete, TSP must be NP hard and since TSP ∈ NP, it is NP-complete

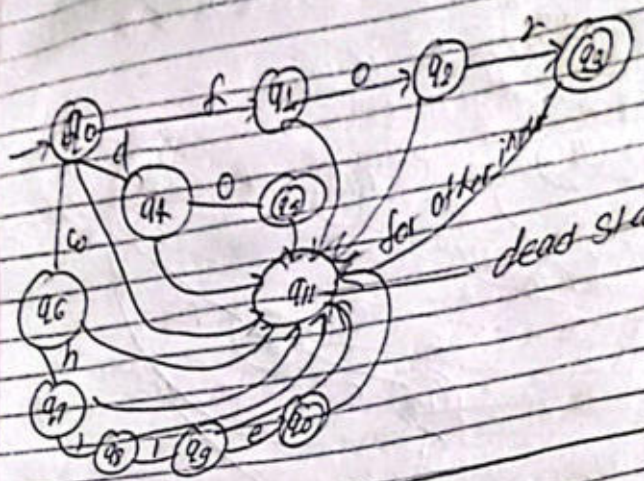
4. Implications of TSP's NP-Completeness

4.1 Pratical Consequences
• No known efficient exact algorithm for large-scale TSP takes O(n!) time

• Heuristics & Approximations:
Nearest-neighbour genetic algorithms and dynamic Programming are used.

Q. Design DFA that accept three loop keywords used C.
keywords... while, for, do





case study:

- Christofides Algorithm (1976) provides a 1.5-approximation for metric TSP.

4.2 Theoretical Significance

- TSP is a benchmark for computational hardness.
- Many real-world problem (eg. vehicle routing, circuit design) are TSP variants.
- Proving P vs NP would resolve whether TSP can be solved efficiently.

5. Conclusion

Karp's 1972 proof established TSP as NP-complete, linking it to the core of computational complexity theory. While exact solutions remain intractable for large inputs, approximation methods and heuristics are widely used in logistics, manufacturing and AI.

TSP remains a fundamental problem in computer science, illustrating the challenges and beauty of NP-completeness.

6. References

- Karp, R.M. (1972). Reducibility Among Combinatorial Problems.
- Garey, M.R. & Johnson, D.S. (1979). Computers and Intractability: A Guid to NP-Completeness
- Christofides, N. (1976). Worst-case analysis of a new heuristic for the TSP.

03/25

Q. Fall

is P = NP ?
No, this is not proven yet.
Experts believe P ≠ NP (i.e. solving is harder than verifying)

if P = NP, problems whose solutions are easy to check would easy to solve.

if P ≠ NP, problems
- Cryptography would collapse
- no security
- AI advance dramatically
- Mathematics could be automated.

Q. Fall - How P & NP problems reflect the notion of deterministic & non-deterministic algorithms.

→ P: deterministic follows single well-defined computation path, next action is uniquely determined by current state
P solvable at polynomial time. verify step by step
eg: sorting: O(nlogn)
not compute everything sequentially.

NP: nondeterministic: guess a solution & verify it.
explores multiple computation paths simultaneously
verified in P.

Q. Spring - difference between FA, PDA & TM according to string acceptance.

FA	PDA	TM
→ reads input from left to right, sequentially	→ reads left to right + uses stack memory	→ reads/writes/moves bidirectionally on an infinite tape
→ decisions on only current state. → regular language only	→ current state + stack contents → context-free language only	→ current state + tape content → Recursively enumerable language only
→ fails if no transition defined for input.	fails if no transitions or stack can't be manipulated to accept.	fails if it halts in non-accepting state or loop infinitely

→ TM is considered functionally stronger among all of these why?

Sol: → unrestricted memory access and universal computation capability.
- unlimited input tape
- recursively enumerable
- can compute all problems
- full random access
- TM can simulate any algorithm
- TM can simulate FA/PDA

29, spring

6b) Differentiate the time complexity of a Turing Machine and time complexity of a language decision problem.

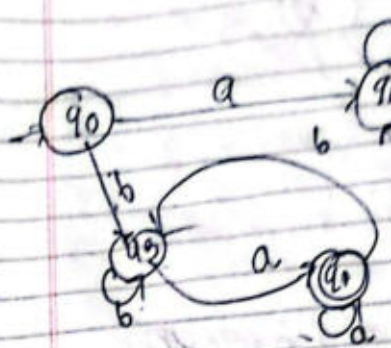
Time Complexity of a TM	Time complexity of language Decision Problem
→ maximum no of steps a TM takes to process input.	→ minimum time required by TM to decide all instances of a language.
→ Machine specific, depends on the implementation of a particular TM.	→ Problem specific: inherent to language itself.
→ can vary between different TMs solving the same problems.	→ fixed for a given language.
→ a poorly designed TM for palindrome might take $O(n^2)$.	→ palindrome has time complexity $O(n)$
→ analyzes the efficiency of a particular algorithm/ machine.	→ characterizes the intrinsic difficulty of a computational problems.
→ contributes to understanding the problem's complexity.	→ Represents the lower bound across all possible TMs for that problem

abab

DPP 09, spring

starts and end with different symbols, $\Sigma = \{a, b\}$

a.b / ba



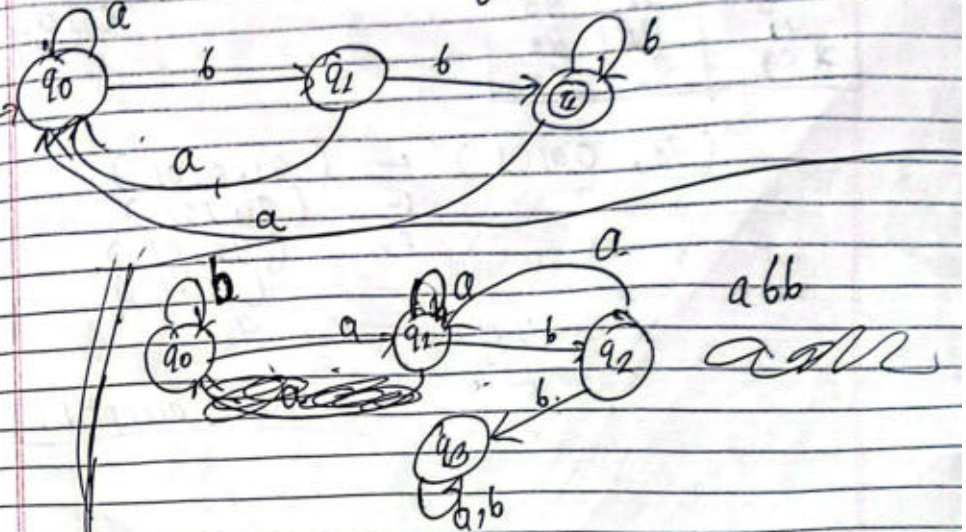
a bb

a bb

24) pseboard

$L = \{\omega \in (a,b)^*, \omega \text{ ends with } bb\}$

babb



a bb