Compile by Birendra Chaudhary

biru

Introduction

Date
Page

Review of sets, relation, function

① Sets: collection of objects
→ denoted by capital letters

Eg:
$A = \{1, 2, 3\}$
$B = \{a, e, i, o, u\}$
$a \in B$
$1 \in A$

→ sets can be represented by
① listing its element
Eg: $V = \{a, e, i, o, u\}$

② by describing properties of element
$V = \{x \mid x \text{ is a vowel letter}\}$
$N = \{n \mid n \text{ is natural number}\}$

Types of sets
① Singleton sets : has only one element
② Empty sets : has no element, denoted by ϕ
③ Disjoint set : Two sets having no common elements
④ Overlapping set : having at least one common element

⑦ Subset: if every element of A is in B then $A \subseteq B$

Additional term

① Cardinality: no. of element in set

Eg: $A = \{1, 2, 3\}$

$|A| = 3$

② Cartesian Product:

Cartesian Product of two set A and B is ordered pair of sets

$A \times B = \{ \{x, y\} \mid x \in A \text{ and } y \in B \}$

③ Power set:

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

$P(A) = \{\phi, 1, 2, \{1, 2\}\}$

Powerset

$2^n = 2^2 = 4$

n = no. of element

Some Set Operations

① Union

$A \cup B = \{x : x \in A \text{ or } x \in B\}$

② Intersection

$A \cap B = \{x : x \in A \text{ and } x \in B\}$

③ Set Difference

$A - B = \{x : x \in A \text{ and } x \notin B\}$

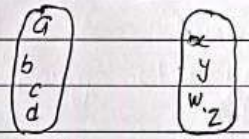
④ Complement

$\overline{A} = \{\text{everything that is not in } A\}$

Relation

A relation is an ordered pair of sets such that to each element of domain, there is an assigned one or more ronge

Suppose S and T are two sets



$R = \{(a, x)\ (b, y)\ (c, w)\ (d, z)\}$

Types of Relation:

1) Reflexive relation:

A relation 'R' is reflexive, if every element of set A is related to itself

Eg: $A = \{1,2,3\}$

$R = \{(1,1), (2,2), (3,3)\}$

2) Symmetric relation:

A relation 'R' is symmetric if $(a,b) \in R$ whenever $(b,a) \in R$

Eg: $A = \{1,2,3\}$

$R = \{(1,2), (2,1), (2,3), (3,2)\}$

3) Transitive relation:

A relation 'R' is transitive if $(a,c) \in R$ whenever $(a,b) \in R$ and $(b,c) \in R$

Eg: $A = \{1,2,3\}$

$R = \{(1,2), (2,3), (1,3)\}$

1/12

* Equivalence relation:

A relation R on set A is equivalence if it is reflexive, symmetric and transitive.

Eg: $A = \{1,2,3\}$

$R = \{(1,1) (2,2) (3,3) (1,2) (2,1) (2,3) (3,2) (1,3) (3,1)\}$

* Function

A function is said to map an element in domain to an element in range.

→ A function is denoted by $f(x)$

→ A function can defined as

$f(x)$ or $f : x \rightarrow y$

where $f(x)$ is called image of x under f.

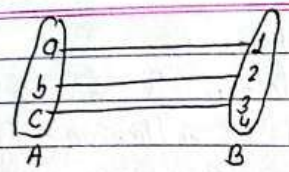
* Types of function

① One to one function

A function $f: A \rightarrow B$ is said to be one to one if every domain in A has different image in B.

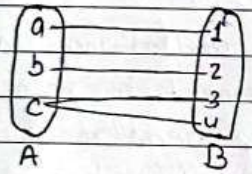
↳ also know as injection function

Eg:



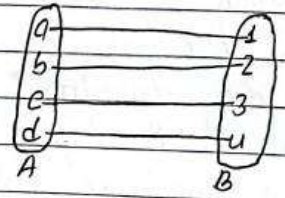
② Onto / Surjection function

A function $f: A \to B$ is said to be onto function if each element of B has some image in A



③ One to One onto function

A function $f: A \to B$ is said to be one to one onto function if each and every element of A is mapped to exactly one element of B with no element left over.



also know as bijection function

* Alphabet and Languages

1) Alphabet:
→ symbol that can be letters or digits
→ denoted by Σ
Eg:
$\Sigma = \{0,1\}$
$\Sigma = \{a,b,c,d\}$
$\Sigma = \{\#, *\}$

2) String or Word:
→ finite sequence of concatenated symbol over alphabet Σ.
Eg:
$\Sigma = \{0,1\}$
Here '00', '1011', '1110', ---- are strings over $\Sigma = \{0,1\}$

3) Length of string
→ no. of symbol in a string W
Eg:
$\Sigma = \{0,1\}$
$W = 0101$
$|W| = |0101| = 4$

4) Empty string
→ string of zero length
→ denoted by e or ε or $\wedge$ or λ (Σ) (ε)

5) Substring
z is substring in w if z appears consecutively in w
Eg:
'01' is substring in '1011'

Concatenation of string
Let ω_1 and ω_2 be two strings, then concatenation of ω_1 and ω_2 is
Here $\omega_1 = abc$
$\omega_2 = xyz$
$\omega_1\omega_2 = abcxyz$

Closure of alphabet
→ defined as set of all strings over an alphabet (Σ) including empty string and denoted by Σ^*.

Eg: $\Sigma = \{0, 1\}$ then
$\Sigma^* = \{0, 1, , \varepsilon, 00, 10, 11, 101, \ldots\}$

Language:
→ subset of all set of possible strings formed from given set of alphabet.
→ formal language 'L' is subset of Σ^*
Eg: $\Sigma = \{0, 1\}$ then
L = {set of all string with equal no. of 0's and 1's}
i.e. L = {01, 10, 0011, 1010, ----}

Grammar
A grammar is a formal system for accepting or rejecting a string.

Date Page

Proof of Techniques

1. proof by contradiction
2. ~~proof~~ pigen hole principle
3. Induction
4. Diagonalization

1. Proof by contradiction

→ given statement is either true or false but not both.

→ mathmetical proof techniques that determine the given statement / proposition is true by showing that assuming proposition is false.

→

Steps involve in proof of contradiction:

a) Assume that statement to be proof be false

b) In the next step we show that our supposition leads to logically contradiction [i.e. our supposition is false]

c) In next step, we conclude that our statement is true because our supposition is false.

Date Page

Eg:

Theorem: There is no greatest integer

In first steps assume the statement is false

Suppose there is a greatest integer N.

Then $n \leq N$ for every integer n.

Let

$$M = N + 1$$

Here, M is an integer since sum of integer.

Also, $M > N$ since $M = N + 1$

Thus M is an integer that is greatest than the greatest integer N, which is a contradiction.

Hence our assume statement / supposition is not true and so there is no greatest integer.

Eg: Give a proof by contradication statement "if n^2 is an even integer then n is an even integer"

Suppose n^2 is an even integer then n is an odd integer.

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

Taking squaring on both side

$$n^2 = (2k+1)^2$$

$$n^2 = 4k^2 + 4k + 1$$

$$n^2 = 2(2k^2 + 2k) + 1 \quad \text{[Taking '2' as common]}$$

$$n^2 = 2r + 1 \quad \text{[general form of odd]}$$

where $r = 2k^2 + 2k$

This shows that n^2 is odd, which is a contradiction to our assumption or suppostion

Eg: If n is an integer and $n^3 + 5$ is odd then n is even using contradiction method

Suppose $n^3 + 5$ is odd then n is an odd integer

Here ~~$n^3 + 5$~~ $n = 2k + 1$

Now,

$$n^3 + 5 = (2k+1)^3 + 5$$

$$= 2(4k^3 + 6k^2 + 3k + 3)$$

Since this expression is multiple of 2 it is even.

However this given informat that $n^3 + 5$ is odd. Therefore our initial assumption that n is odd must be incorrect. Hence n must be even.

Assignment

Date
Page

1. If n and m are odd integer, then n+m is even integer by using contraction

Suppose n and m are odd integers but n+m is not even if n and m are odd they can be repricsented as $n = 2k+1$, $m = 2k+1$

Now lets suppose that n+m is odd as well

$n+m = (2k+1)(2l+1)$

$= 2(k+1+l)$

But this implies that n+m is also even, which contradicts our assumption

Therefore, if n and m are odd integer, then n+m must be even.

2) $\sqrt{2}$ is irrational

Assume $\sqrt{2}$ is rational if can be expressed as a fraction a/b

so $\sqrt{2} = a/b$

Squaring both side we

$2 = \frac{a^2}{b^2}$

Assignment

Date
Page

$a^2 = 2b^2$

Since, a^2 is even (because it is divisible by 2) a must be even as well

let $a = 2k$

Substitute $a = 2k$ into $a^2 = 2b^2$ we get

$(2k^2) = 2b^2$

$4k^2 = 2b^2$

divide by 2

$2k^2 = b^2$

This implies b^2 is even. So b must be even

However if both a and b are even then they have a common factor of 2 which contradicts the assumption that a/b is in it's simplest form.

Therefore our assumption that $\sqrt{2}$ is rational must be false. Thus $\sqrt{2}$ is irrational.

3. If $3n+2$ is odd, then n is odd.

Assume that $3n+2$ is odd, but n is not odd.

If n is not odd then it must be even. So, we can represent n as $n=2k$ where k is an integer

Substituting $n=2k$, we get $3(2k)+2 = 6k+2$

Now analyze $6k+2$. This expression is even because it can be ~~wee~~ written as $2(3k+1)$

But

we assumed that $3x+2$ is odd. This contradict our finding $6k+2$ is even

Therefore, our initial assumption that n is not odd must be false. Thus if $3n+2$ is odd then n must be odd as well.

ii) Mathematical Induction

Steps:

1) Inductive hypothesis (assumption) $\rightarrow k$
2) Use $n=k+1$ in $P(n)$

1) Proposition: The sum of first n positive integer is $\frac{1}{2}n(n+1)$.

Initial steps: For $n=1$, $\frac{1}{2} \times 1(1+1) = 1$

Stage 1: Our assumption (inductive hypothesis) for $n=k$.

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

Stage 2:

For $n=k+1$,

$1+2+3+4+\cdots+k+1 = \frac{1}{2}(k+1)[k+1+1)$

$= \frac{1}{2}(k+1)(k+2)$

Stage 3:

$1+2+3+4+\cdots+k+1 = 1+2+3+4+\cdots+k+k+1$

$= \frac{1}{2}k(k+1) + (k+1)$

$= (k+1)\left[\frac{1}{2}k+1\right]$

$= (k+1) \cdot \frac{(k+2)}{2}$

2) Proposition: For all $n \geq 1$, prove that

$1^2+2^2+3^2+4^2+\ldots+n^2 = n(n+1)(2n+1)/6$

Initial step: For $n=1$ then

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

$= \frac{2 \cdot 3}{6} = 1$

Stage 1: Our assumption for $n=k$

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

Stage 2: For $n=k+1$

$1^2+2^2+3^2+\ldots+(k+1)^2 = \frac{(k+1)(k+1+1)(2(k+1)+1)}{6}$

$= \frac{(k+1)(k+2)(2k+3)}{6}$

Stage 3: ~~For~~

$1^2+2^2+3^2+\ldots+(k+1)^2 = 1^2+2^2+3^2+\ldots+k^2+(k+1)^2$

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

Assignment

$= (k+1)\left[\frac{k(2k+1)}{6} + (k+1)\right]$

$= (k+1)\left[\frac{(6k+6)+(2k+1)\cdot k}{6}\right]$

$= \frac{(k+1)}{6}\left[6k+6+2k^2+k\right]$

$= \frac{(k+1)}{6}\left[2k^2+7k+6\right]$

$= \frac{(k+1)}{6} \cdot (k+2)(2k+3)$

Q. Prove that $1.2.3 + 2.3.4 + \ldots + n(n+1)(n+2) = \frac{n(n+1)(n+2)(n+3)}{4}$

Inital step:

For $n=1$, $\frac{n(n+1)(n+2)(n+3)}{4} = \frac{1(1+1)(1+2)(1+3)}{4}$

$= 6$

Stage 1: Our assumption for $n=k$

$1.2.3+2.3.4+\ldots+k(k+1)(k+2) = \frac{k(k+1)(k+2)(k+3)}{4}$

Assignment

Stage 2 : For $n = k+1$,

$$1\cdot2\cdot3 + 2\cdot3\cdot4 + \dots + (k+1)(k+1+1)(k+1+2) = \frac{(k+1)(k+1+1)(k+1+2)(k+1+3)}{4}$$

Stage 3 :

$$1\cdot2\cdot3 + 2\cdot3\cdot4 + \dots + k(k+1)(k+2) + (k+1)(k+1+1)(k+1+2)$$

$$= \frac{k(k+1)(k+2)(k+3)}{4} + (k+1)(k+1+1)(k+3)$$

$$= (k+1)(k+2)(k+3)\left[\frac{k}{4} + 1\right]$$

$$= (k+1)(k+2)(k+3)\left[\frac{k+4}{4}\right]$$

$$= \frac{(k+1)(k+1+1)(k+1+2)(k+1+3)}{4}$$

Thus $k+1$ is true whenever the statement is true for all natural number.

1/16

Q) For every positive integer n, prove that: $7^n - 3^n$ is divisible by 4

We have $P(n) = 7^n - 3^n$, divisible by 4

For $n=1$,

$P(1) = 7^1 - 3^1 = 4$ which is divisible by 4

Thus $P(n)$ is true for $n = 1$

Let $P(k)$ is true for some positive integer k.

i.e. $P(k) = 7^k - 3^k$ which is divisible by 4

So,

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

$= 4d$, $d \in N$

Now, For some positive integer $k+1$,

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

$= 7^{k+1} - 7\cdot3^k + 7\cdot3^k - 3^{k+1}$

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

$= 7(7^k - 3^k) + 3^k \cdot 4$

$= 7 \cdot 4d + 3^k \cdot 4$ $\quad [7^k - 3^k = 4d]$

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

which is divisible by 4

Thus given statement $P(n) = 7^n - 3^n$ is true for every positive integer n

Q) Prove that for all $n \in N$, $n(n+1)(n+5)$ is a multiple of 3

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

For $n=1$,

$P(1) = 1(1+1)(1+5) = 12$ which is the multiple of 3

Thus $P(1)$ is true for $n=1$

Let $P(k)$ is true for some positive integer k.

ie. $P(k) = k(k+1)(k+5)$ which is multiple of 3

So,

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

$= 3d \quad d \in N$

Now for some positive integer $k+1$)

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

$= (k+1)[(k+2)(k+6)]$

$= (k+1)[k^2+6k+2k+12]$

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

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

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

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

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

$= 3[d + (k+4)(k+1)]$

which is ~~divisible~~ multiple of 3

Thus given statement $P(n) = n(n+1)(n+5)$ is true for all positive $n \in N$.

1/18

Date
Page

Pigeon hole principle:

→ If k+1 object is to be placed in k boxes, then there is at least one box that has more than one object.

→ If N object are to be placed in k boxes then there is at least one box containing N/k object

Theorem: A function f from a set with k+1 element to a set with k element is not one to one.

Proof: Using pigeonhole principle

ⓐ Create a box for each y in the codomain f.

ⓑ put in the box for y of all the element of x such that $f(x) = y$.

ⓒ Since there are k+1 element and only k boxes, at least one box has two or more element.

Hence the function is not one to one

1/18

Date
Page

Generalized: pigeon principle:-

If There are n pigeon holes and k·n+1 or more pigeons, then at least one pigeon hole is occupied by k+1 or more pigeon.

Example: How many students are there in a class among which at least four of them are ~~gone~~ born in same months.

⇒ Total no. of month (n) = 12 (Pigeon hole)

i.e. $k+1 = 4 = 3+1$

$\therefore k = 3$

Total no. of student (pigeon)

$= kn + 1$

$= 3 \times 12 + 1$

$= 37$

Q) 25 carats of apples are delivered to a store. The apples are of 3 different sorts and all the apple in each creates are of the ~~sor~~ same sorts. So that among these carats there are at least 9 containing the same sorts of apples.

No. of different creates (n) = 3
Total no. of creates (kn+1) = 25
i.e. $K \cdot 3 + 1 = 25$
$K = \frac{24}{3} = 8$

To Now, $K+1 = 8+1 = 9$

Thus there are at least 9 containing the same sorts of apples.

Diagonalization principle

Statement: Let R be a binary relation on a set A and let D, the diagonal set for R, be $\{a : a \in A \text{ and } (a,a) \notin R\}$. For each $a \in A$, let $R_a = \{b : b \in A \text{ and } (a,b) \in R\}$. Then D is distinct from each R_a.

↳ use to prove undecidability (Halting problem)

In short, diagonalization principle state that the complement of diagonal is different from each row

Example: Let $S = \{a,b,c,d\}$
$R = \{(a,a)\ (b,c)\ (b,d)\ (c,a)\ (c,c)\ (c,d)\ (d,a)\ (d,b)\}$

Matrix form of R

	a	b	c	d
a	x			
b			x	x
c	x		x	x
d	x	x		

From figure,

$R_a = \{a\}$

$R_b = \{c, d\}$

$R_c = \{a, c, d\}$

$R_d = \{a, b\}$

Complement of diagonal is.

$D = \{b, d\}$

i.e. if we compare each of the set R_a, R_b, R_c, R_d with diagonal set D we can see D is different from each row. Thus complement of diagonal is different from each row.

Chomsky Hierarchy

→ hierarchy of grammar
→ defined by Noam chomsky in 1956
→ Grammar is defined by four tuples
$G = \{V, T, P, S\}$
↓ rule to define / generate language.

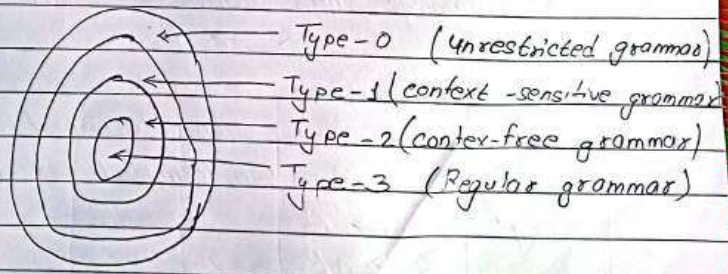
where
V = non-terminal or variables (capital letter)
T = Terminal (small letter)
P = Production rule
S = Starting Symbol

Eg: Production rule

$S \rightarrow AB$

$A \rightarrow a$

$B \rightarrow b$



① Unrestricted grammar generate recursive enumerable language (REL)

② Content sensitive grammar generate context sensitive language (CSL)

③ Content free grammar generate content free language

④ Regular grammar generate regular language (RL)

① Type-0 grammar:
→ includes all formal grammar
→ all language are recognized by TM
→ describe syntax for programming languages
→ Grammar has rule of form
$\alpha \rightarrow \beta$
where
α is non-terminal
β = non-terminal or terminal
Eg: $AB \rightarrow A$
$A \rightarrow aB$
$B \rightarrow a$

② Type 1 Grammar
→ generate context sensitive language
→ all language are recognized by LBA (Linear Bound Automata)
→ grammar has rule of form with restriction that length of $|\alpha| \leq |\beta|$
$\alpha \rightarrow \beta$
Eg: $AB \rightarrow A$ (not acceptable) ✗
$A \rightarrow aB$ (acceptable) ✓
$B \rightarrow a$ (acceptable) ✓

③ Type 2 Grammar
→ generate content free language
→ all language are recognized by PDA (Push down Automata)
→ grammar has rule of form $A \rightarrow \alpha$
where
A = non-terminal
α = non-terminal or terminal
(There will be no context on the left & right on the non-terminal)

Eg: $A \rightarrow ABC$ ✓
$A \rightarrow BCD$ ✓
$a \rightarrow Abc$ ✗

③ Type -3 grammar
→ generate regular language
→ all language are recognized by FA (Finite Automata)
→ grammar has rule of form

$$\alpha \rightarrow \beta$$

where α = non-terminal
β = non-terminal / Terminal or ε

Eg: $S \rightarrow ab \mid S$
$A \rightarrow b$
$B \rightarrow \varepsilon$

Chapter-2 1/24

Finite Automata & Regular Expression

Date Page

Finite Automata : (self-acting)

→ set of states and it's control moves from state to state in response to external input
→ simplest model of computing
→ recognized model of computing

Formal definition:

FA has 5 types defined as

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

where

Q = set of finite state
Σ = input symbol
δ = transition function, $Q \times \Sigma = Q$
q_0 = initial state
F = set of final states / accepted states

Types:

~~① Deterministic FA (DFA): doesnot~~
~~② Non-deterministic~~

1/24

Date Page

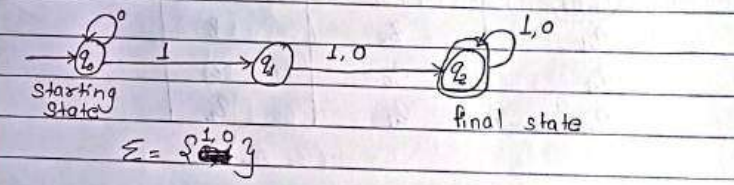
Types:

① Deterministic FA (DFA):
depend on current states and i/p

② Non-deterministic FA (NFA / NDFA):
particularly depend on current state and i/p

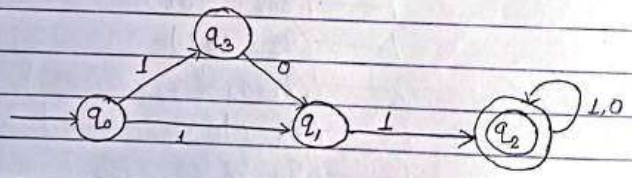
initial state
final state

Example: Deterministic FA



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

Example: Non-deterministic FA



Instantaneous Description (ID)

Q. Given string is 11010 1. check whether the string is accepted or not when transition function (δ) are

State table:

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

Solution:

$q_0, 11010 1 \vdash (q_1, 10101)$
$\vdash (q_0, 0, 101)$
$\vdash (q_2, 101)$
$\vdash (q_3, 01)$
$\vdash (q_1, 1)$
$\vdash (q_0, \varepsilon)$

Since q_0 is final state
So string is accepted.

Assignment

Consider Finitestate Automat in which
$Q = \{q_0, q_1, q_2, q_3\}$
$\Sigma = \{0, 1\}$
$q_0 = \{q_0\}$, initial state
$F = \{q_0\}$ final state
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

a) Write a entire sequence of state for the input string 101101

b) Check whether string is accepted or not
i) 11111
ii) 000000
iii) 101101

Solution

a) $(q_0, 101101) \vdash (q_1, 01101)$
$\vdash (q_3, 1101)$
$\vdash (q_2, 101)$

Assignment

Date
Page

$\vdash (q_3, 01)$
$\vdash (q_1, 1)$
$\vdash (q_0, \varepsilon)$

Here q_0 is final state so it is accepted and

the sequence of state is

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

b) i) $(q_0, 11111) \vdash (q_1, 1111)$
$\vdash (q_0, 111)$
$\vdash (q_1, 11)$
$\vdash (q_0, 1)$
$\vdash (q_1, \varepsilon)$

Here q_1 is final state so it is not accepted.

ii) $(q_0, 00000) \vdash (q_2, 00000)$
$\vdash (q_0, 0000)$
$\vdash (q_2, 000)$
$\vdash (q_0, 00)$
$\vdash (q_2, 0)$
$\vdash (q_0, \varepsilon)$

Here, q_0 is final state so it is accepted.

Assignment

Date
Page

iii) $(q_0, 10101) \vdash (q_1, 0101)$
$\vdash (q_3, 101)$
$\vdash (q_2, 01)$
$\vdash (q_3, 1)$
$\vdash (q_1, \varepsilon)$
$\vdash (q_0, \varepsilon)$

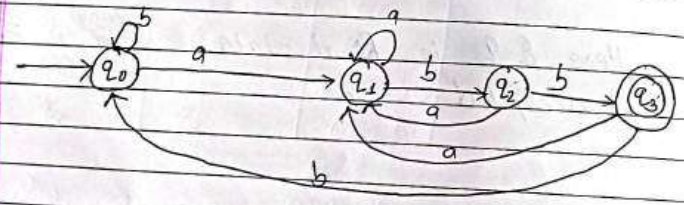
Here q_0 is final state so string is an accepted

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

given, substring = abb

length of substring, $|abb| = 3$

no. of states = 3 + 1 = 4

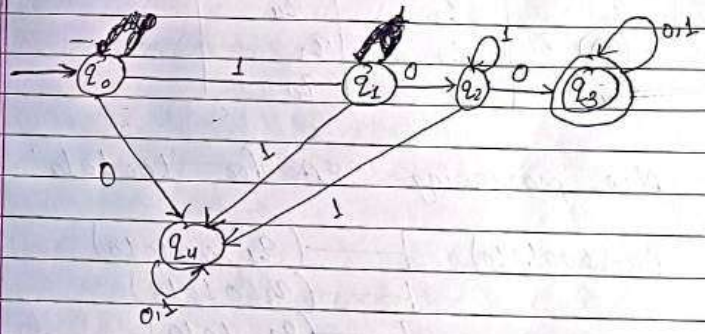


Q. Design a DFA that accept string of 0's and 1's starting with 100 over $\Sigma = \{0, 1\}$

given substring = 100

length of substring $|100| = 3$

no. of state = 3 + 1 = 4



Hence DFA, M is defined as

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

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	Input 1
$\rightarrow (q_0)$	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 DFA for '1001010' as

$(q_0, 1001010) \vdash (q_1, 001010)$
$\vdash (q_2, 01010)$
$\vdash (q_3, 1010)$
$\vdash (q_3, 010)$
$\vdash (q_3, 10)$
$\vdash (q_3, 0)$
$\vdash (q_3, \varepsilon)$

Hence, the string is accepted.

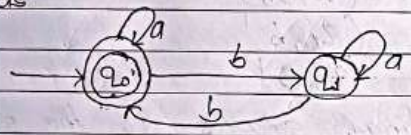
Q. Design a DFA that accepts language which set string in $(a, b)^*$ that have even no. of b's

possible language of even b's = {bb, bab, aabb, bababab}

length of substring $|bb| = 2$

no. of state = 2+1 = 3

Now state transition diagram can be drawn as



Here

DFA, M is defined as

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

where

$Q = \{q_0, q_1\}$

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

$q_0 = \{q_0\}$

$F = \{q_0\}$

δ is given by

State	Input	
	a	b
q_0	q_0	q_1
q_1	q_1	q_0

Now processing DFA for 'aabb' as

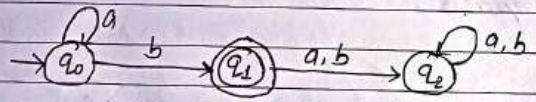
$\delta(q_0, aabb) \vdash (q_0, abb)$
$\vdash (q_0, bb)$
$\vdash (q_1, b)$
$\vdash (q_0, \varepsilon)$

Here q_0 is final state so the string is accepted.

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

Possible language, $L = \{b, ab, aab, \ldots\}$
Length of string = $|b| = 1$
no. of state = $1+1 = 2$

State transition diagram:



Here DFA, M is defined as
$M = \{Q, \Sigma, \delta, q_0, F\}$
where
$Q = \{q_0, q_1, q_2\}$
$\Sigma = \{a, b\}$
$q_0 = \{q_0\}$
$F = \{q_1\}$

δ is given by

State	Input	
	a	b
q_0	q_0	q_1
q_1	q_2	q_2
q_2	q_2	q_2

Assignment

Date
Page

Now, processing DFA for 'aaab' as

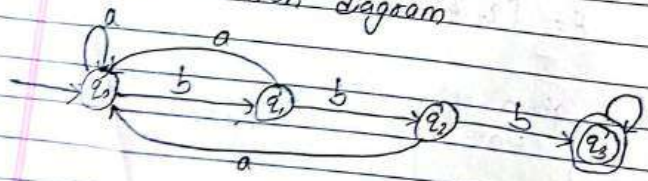
$(q_0, aaab) \vdash (q_0, aab)$
$\vdash (q_0, ab)$
$\vdash (q_0, b)$
$\vdash (q_1, \varepsilon)$

Since q_1 is final state so it is accepted.

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

Possible language, $L = \{bbb, abbb, \ldots\}$
length of substring, $|bbb| = 3$
no. of state $= 3 + 1 = 4$

State transition diagram



Assignment

Date
Page

We know $M = \{Q, \Sigma, \delta, q_0, F\}$
where
$Q = \{q_0, q_1, q_2, q_3\}$
$\Sigma = \{a, b\}$
$q_0 = \{q_0\}$
$F = \{q_3\}$

State Transition table (δ)

State	Input	
	a	b
q_0	q_0	q_1
q_1	q_0	q_2
q_2	q_0	q_3
q_3	q_3	q_3

Now, processing DFA for 'abbbab' as

$(q_0, abbbab) \vdash (q_0, bbbab)$
$\vdash (q_1, bbab)$
$\vdash (q_2, bab)$
$\vdash (q_3, ab)$
$\vdash (q_3, b)$
$\vdash (q_3, \varepsilon)$

Since q_3 is final state So, the string is accepted!

Assignment

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

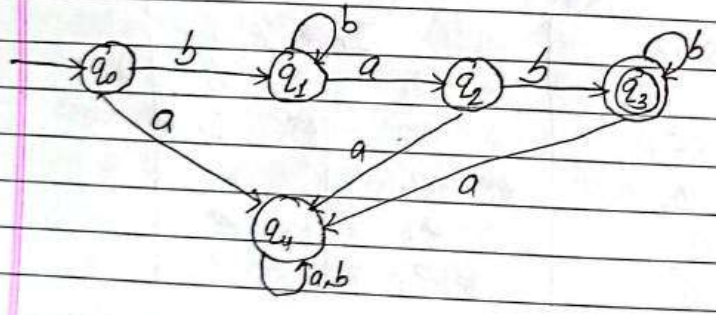
Solution:

Possible language, $L = \{bab, bbab, \ldots\}$

Length of substring $|bab| = 3$

no. of state $= 3 + 1 = 4$

State Transition Diagram:



We know, M is defined as

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

where,

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

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

$q_0 = \{q_0\}$

$F = \{q_3\}$

State transition table (δ) is

State	Input	
	a	b
q_0	q_4	q_1
q_1	q_2	q_1
q_2	q_4	q_3
q_3	q_4	q_3
q_4	q_4	q_4

Now processing DFA for 'bbbab'

$(q_0, bbbab) \vdash (q_1, bbab)$

$\vdash (q_1, bab)$

$\vdash (q_1, ab)$

$\vdash (q_2, b)$

$\vdash (q_3, \varepsilon)$

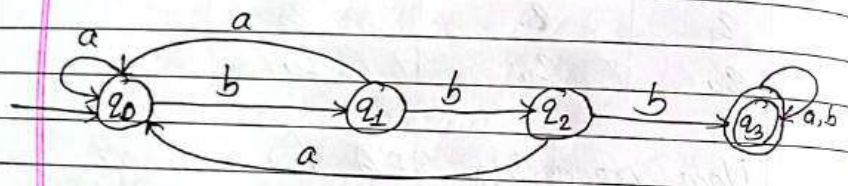
Since q_3 is final state

So this string is accepted.

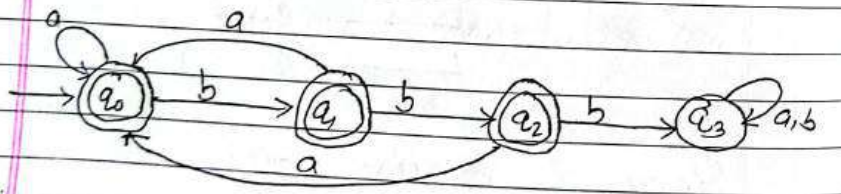
1/26

Q. Design DFA that accept all string that doesnot have three consuctive b's over $\Sigma = \{a, b\}$

firstly state transition diagram for having three consecutive b's



Now,
state transition diagram for doesnot have three consecutive b's



Here DFA, M is defined as
$M = \{Q, \Sigma, \delta, q_0, F\}$
where,
$Q = \{q_0, q_1, q_2, q_3\}$
$\Sigma = \{a, b\}$
$q_0 = \{q_0\}$

1/26

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

State transition table (δ) :

State	Input	
	a	b
q_0	q_0	q_1
q_1	q_0	q_2
q_2	q_0	q_3
q_3	q_3	q_3

Now checking for DFA 'ababaabb'

$(q_0, ababaabb) \vdash (q_0, babaabb)$
$\vdash (q_1, abaabb)$
$\vdash (q_0, baabb)$
$\vdash (q_1, abb)$
$\vdash (q_0, bb)$
$\vdash (q_1, b)$
$\vdash (q_2, \varepsilon)$

Here q_2 is final state so, the string is accepted.

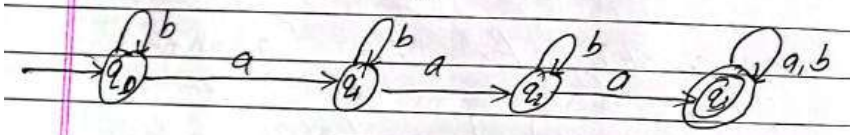
Q. Design DFA that accept all string not having more that two a's over $\Sigma = \{a, b\}$

firstly for ~~none~~ string have more than two a possible substring, $L = \{aa, baa, baab, \ldots\}$

length of substring $|aa| = 2$

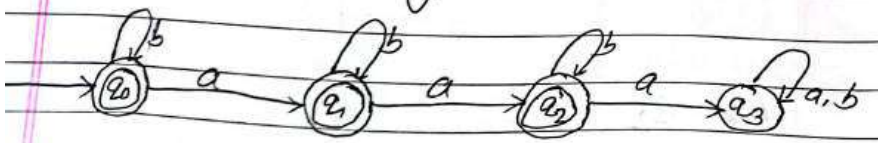
no. of ~~sta~~ state $= 2+1 = 3$

Transition state diagram:



Now For string not having more than two a's

State transition diagram:



Here, M is defined as

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

where

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

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

$q_0 = \{q_0\}$

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

State transition table (δ)

State	Input	
	a	b
q_0	q_1	q_0
q_1	q_2	q_1
q_2	q_3	q_2
q_3	q_3	q_3

Now, processing for DFA 'abbaba'

$(q_0, abbaba) \vdash (q_1, bbaba)$

$\vdash (q_1, baba)$

$\vdash (q_1, aba)$

$\vdash (q_2, ba)$

$\vdash (q_2, a)$

$\vdash (q_3, \varepsilon)$

Here q_3 is final state so the string is acceptable

Equivalence of DFA and DFA

Q. Construct DFA equivalent to NFA, $M=(\{q_0,q_1\}, \{0,1\}, \delta, \{q_0\}, \{q_0\})$ and state table is

State	Input	
	0	1
q_0	q_0	q_1
q_1	q_1	q_0q_1

For NFA,

Here initial state = q_0

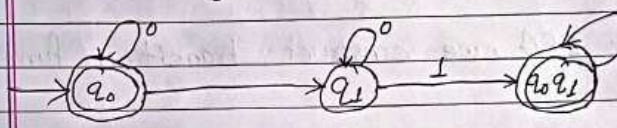
final state = q_0

Now for DFA we construct transition function δ as

State	Input	
	0	1
q_0	q_0	q_1
q_1	q_1	q_0q_1
q_0q_1	q_0q_1	q_0q_1

Note: Initial state of NFA is same as DFA

So, state diagram for DFA is



Here final state of DFA is

$F' = 2^{q-1} = 2^{2-1} = 2$

$F^1 = \{q_0, q_0q_1\}$

Q. Construct a DFA equivalent to $M' = (\{q_0,q_1,q_2,q_3\}, \{0,1\}, \delta, \{q_0\}, \{q_3\})$ and δ is

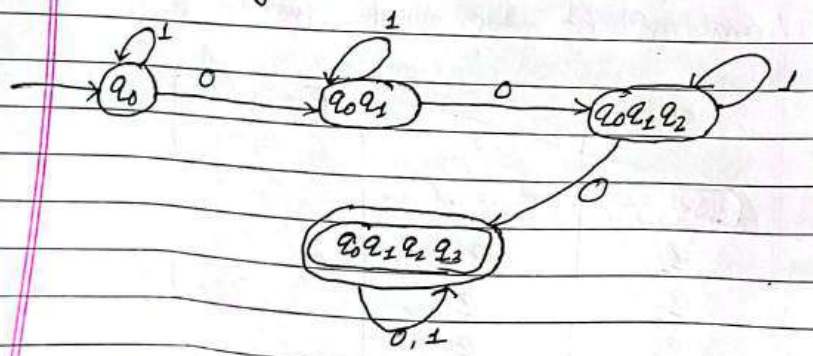
State	Input	
	0	1
q_0	q_0q_1	q_0
q_1	q_2	q_1
q_2	q_3	q_2
q_3	q_3	q_3

Here for NFA
initial state = q_0
final state = q_3

Now for DFA, we construct transition function δ as

State	Input	
	0	1
q_0	q_0q_1	q_0
q_0q_1	$q_0q_1q_2$	q_0q_1
$q_0q_1q_2$	$q_0q_1q_2q_3$	$q_0q_1q_2$
$q_0q_1q_2q_3$	$q_0q_1q_2q_3$	$q_0q_1q_2q_3$

So state diagram for DFA is



∴ final stat of DFA is $\{q_0\ q_1\ q_2\ q_3\}$

Construct a DFA equivalent to $M = (\{q_1, q_2, q_3\}, \{0, 1\}, \delta, \{q_1\}, \{q_3\})$ and δ is given by

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

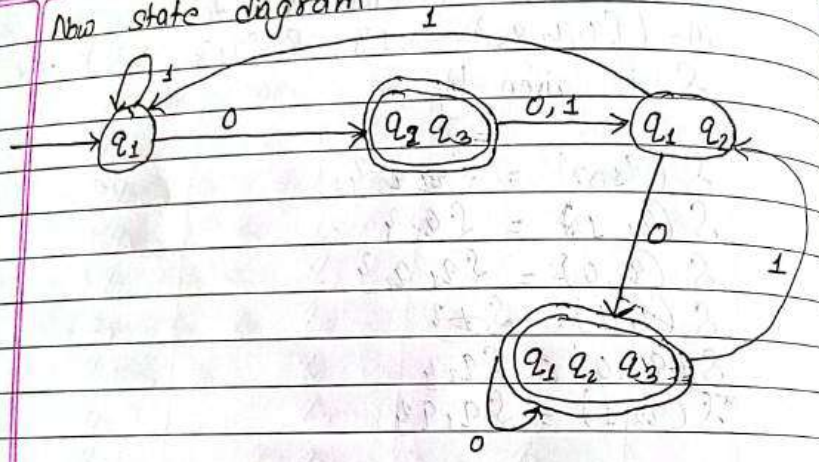
Solution:
For NFA,
initial state = q_1
final state = q_3

For DFA,
initial state = q_1 (same as NFA)

State transition table (δ) for DFA is

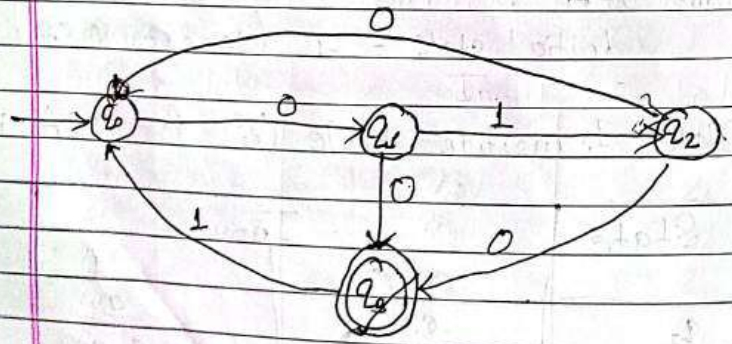
State	Input	
	0	1
q_1	q_2q_3	q_1
q_2q_3	q_1q_2	q_2q_1
q_1q_2	$q_1q_2q_3$	q_1
$q_1q_2q_3$	$q_1q_2q_3$	q_1q_2

Now state diagram



∴ The final state for DFA: $q_2 q_3$, $q_1 q_2 q_3$

2. Construct a DFA equivalent to given NFA



Solution For NFA,

initial state = q_0

finial state = q_3

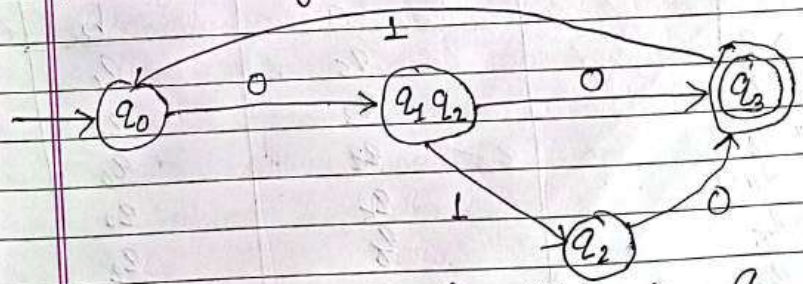
For DFA

initial state = q_0 (same as NFA)

Now state transition for DFA is

State	Input	
	0	1
q_0	$q_1 q_2$	ϕ
$q_1 q_2$	q_3	q_2
q_3	ϕ	q_0
q_2	q_3	ϕ

Now, Stat diagram:



∴ The finial state for DFA is q_3.

Minimization of DFA

Example:

	Current State	Input Symbol a	Input Symbol b
(initial)	$\rightarrow q_0$	q_5	q_1
	q_1	q_2	q_6
(final)	$*q_2$	q_2	q_0
	q_4	q_5	q_7
	q_5	q_6	q_2
	q_6	q_4	q_6
	q_7	q_2	q_6
	q_3	q_6	q_2

Step 1: Eliminate state that can't be reached from starting state.

Here q_3 is not reachable, so remove it we get.

Current State	Input Symbol a	Input Symbol b
$\rightarrow q_0$	q_5	q_1
q_1	q_2	q_6
$*q_2$	q_2	q_0
q_4	q_5	q_7
q_5	q_6	q_2
q_6	q_4	q_6
q_7	q_2	q_6

Step 2: Now divide the table into two sets as

a) set 1: containing row which start from non-final state.

$\rightarrow q_0$	q_5	q_1	Row 1
q_1	q_2	q_6	Row 2
q_4	q_5	q_7	Row 3
q_5	q_6	q_2	Row 4
q_6	q_4	q_6	Row 5
q_7	q_2	q_6	Row 6

set 1

b) set 2: containing row which start from final state.

$*q_2$	q_2	q_0

Step 3: Considering set 1

Here row 2 and row 6 transit to same state on input a and b

So, we remove one of them (for instance q_7) and replace q_7 with q_1

→ q_0	q_5	q_1
q_1	q_2	q_6
q_2	q_5	q_1
q_5	q_6	q_2
q_6	q_4	q_6

Again row 1 and row 3 is same.
So we remove q_4 and replace with q_0

→ q_0	q_5	q_1
q_1	q_2	q_6
q_5	q_6	q_1
q_6	q_0	q_6

Now there are no similar rows

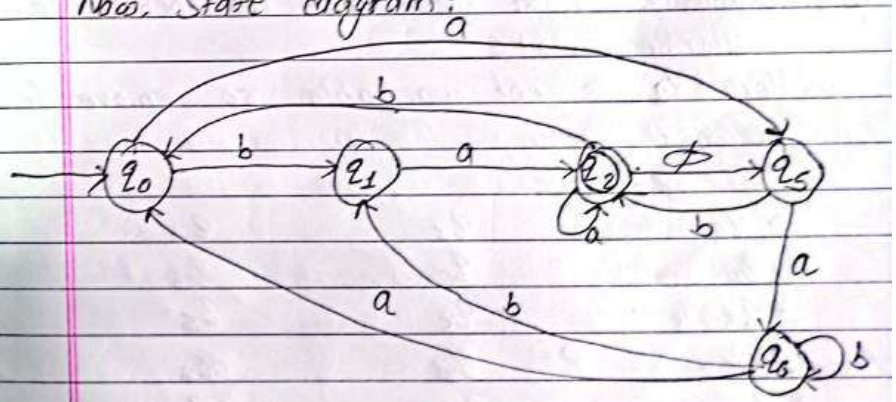
Consider set 2
Here there is only one row. so it is minimized

* q_2	q_1	q_0

Step 4: Now combine minimized set 1 and set 2 as

Current State	Input Symbol a	Input Symbol b
→ q_0	q_5	q_1
q_1	q_2	q_6
* q_2	q_1	q_0
q_5	q_6	q_2
q_6	q_0	q_6

Now, State diagram:



Assignment

Date
Page

Q. Minimize the DFA

Current State	Input Symbol a	Input Symbol b
→ q_0	q_1	q_3
q_1	q_0	q_3
q_2	q_1	q_4
* q_3	q_5	q_5
q_4	q_3	q_3
* q_5	q_5	q_5

Step 1: Eliminate state that can't be reached from starting state

Here q_2 is not reachable so remove it we get

→ q_0	q_1	q_3
q_1	q_0	q_3
* q_3	q_5	q_5
q_4	q_3	q_3
* q_5	q_5	q_5

Assignment

Date
Page

Again q_4 is not reachable so also remove it

→ q_0	q_1	q_3
q_1	q_0	q_3
* q_3	q_5	q_5
* q_5	q_5	q_5

Step 2: Now divide the table into two set as

a) Set 1: containing row which start from non-final state

→ q_0	q_1	q_3
q_1	q_0	q_3

b) Set 2: containing row which stard from final state

* q_3	q_5	q_5
* q_5	q_5	q_5

Assignment

Step 3: Considering set 1
Here the row of set 1 is already minimized

→	q_0	q_1	q_3
	q_1	q_0	q_3

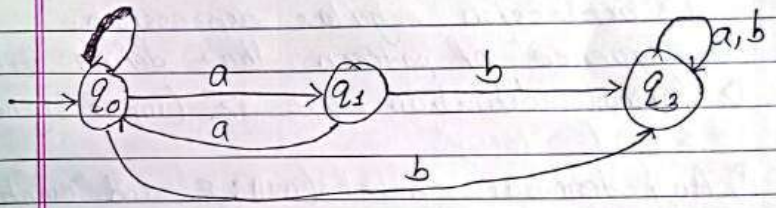
Considering set 2
Here row 1 and row 2 are same. So we remove q_3 and replace with q_3

*	q_3	q_3	q_3

Step 4: Now combining the minimized set 1 and set 2 as

Current State	Input Symbol a	Input Symbol b
→ q_0	q_1	q_3
q_1	q_0	q_3
* q_3	q_3	q_3

State diagram:



Date Page

Regular Expression

↳ language accepted by FA can be represented as expression regular expression
↳ sequence of pattern that define string
↳ formal definition ove Σ (input signal),

① Any terminal symbol (null: ε and empty set: ϕ) are regular expression.

② The union of two regular expression R_1 and R_2 written as $R_1 + R_2$ is also a regular expression.

③ Concatination of two regular expression $R_1 . R_2$ is also a regular expression.

④ The ~~in~~iteration (kleene closure) of regular expression written as r_1^* is also a regular expression.

Eg: $\Sigma = \{a\} = a^* = a$ को repetation
$L = \{a^*\}$
$L = \{\varepsilon, a, aa, aaa, \dots\}$
$R.E = a^*$

Date Page

$L = \{a, aa, aaa, \dots\}$
$R.E = a^+$

$* \rightarrow \varepsilon$ include सुन्न
$+ \rightarrow \varepsilon$ include सुन्न बाहेक

Example 1: Let $\Sigma = \{a, b\}$. Find regular expression for language L whose length is exactly 2.

Here $L = \{aa, ab, ba, bb\}$
Now,
union of all string
$= aa + ab + ba + bb$
$= a(a+b) + b(a+b)$
$= (a+b)(a+b)$

Example 2: For at least length 2

$L = \{aa, ab, bb, ba, aaa, aab, bab, \dots\}$

For exactly length 2
$R.E = (a+b)(a+b)$

Now for at least

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

1/30

Regular Expression

Eg: Write R.E for language accepting all string containing any no. of a's and b's.

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

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

2. Write R.E for strings over $\Sigma = \{0,1\}$ starting with 1 and ending with 0.

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

3. Write R.E for language all strings starting and ending with a and having any combination of b's in between.

$RE = ab^*a$

4. Write R.E for string over $\Sigma = \{0,1\}$ having even length of string.

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

$= \{(0+1)^2\}^*$

$= \{(0+1)^{2*}\} = (0+1)^{2n}, \quad n \geq 0$

5. Write R.E for string over $\Sigma = \{0,1\}$ that does not contain substring 01.

$R.E = \{1^*0^*\}$

Identities of R.E

1. $\phi + R = R$
2. $\phi R + R\phi = \phi$
3. $\varepsilon R = R\varepsilon = R$
4. $\varepsilon^* = \varepsilon$ and $\phi^* = \varepsilon$
5. $R + R = R$
6. $R^*R^* = R^*$
7. $RR^* = R^*R$
8. $(R^*)^* = R^*$
9. $\varepsilon + RR^* = \varepsilon + R^*R = R^*$
10. $(PQ)^*P = P(QP)^*$

11. $(P+Q)^* = (P^*Q^*)^* = (P^* + Q^*)^*$

12. $(P+Q)R = PR + QR$

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

Eg: Prove that $(1+00^*1) + (1+00^*1)(0+10^*1)^* (0+10^*1) = 0^*1(0+10^*1)^*$

LHS $= (1+00^*1) + (1+00^*1)(0+10^*1)^* (0+10^*1)$

$= (1+00^*1)\{\varepsilon + (0+10^*1)^* (0+10^*1)\}$

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

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

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

$=$ RHS #

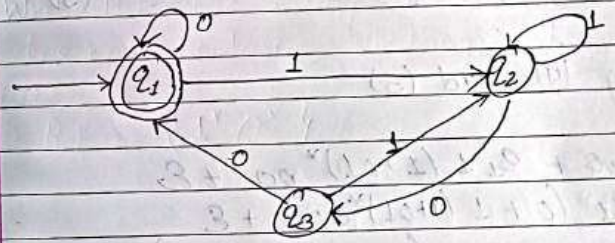
Imp

Arden's Theorem

If P, Q, R are R.E and $R = Q + RP$ then

$R = QP^*$

Q. Find R.E from the following state diagram



Soln Here,

$q_1 = q_10 + q_30 + \varepsilon$ —— (1)

$q_3 = q_20$ —— (3)

$q_2 = q_11 + q_21 + q_31$ —— (2)

Solving (1) and (3)

$q_1 = q_10 + q_200 + \varepsilon$ —— (4)

Solving eqn (2) and (3)

$$q_2 = q_1 1 + q_2 1 + q_2 01$$

$$\underset{R}{q_2} = \underset{Q}{q_1 1} + \underset{R}{q_2}\underset{P}{(11+01)}$$

Now using arden's theorem, we get

$$q_2 = q_1 1 (1+01) \quad \text{——— (5)}$$

Solving (4) and (5)

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

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

$$\underset{R}{q_1} = \underset{Q}{\varepsilon} + \underset{R}{q_1}\underset{P}{(0+1(1+1)^*00)}$$

Using arden's theorem

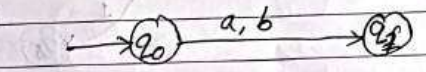
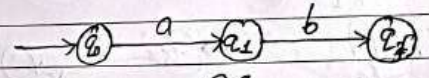
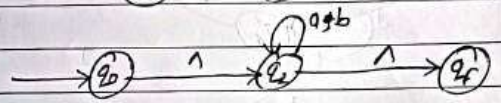
$$q_1 = \varepsilon (0+1 (1+01)^* 00)^*$$

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

So the RE is $(0+1(1+01)^* 00)^*$

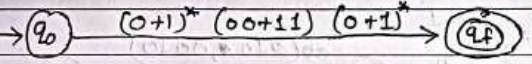
2/1

Construction of FA from R.E

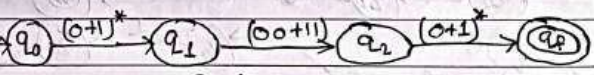
	R.E	F.A
1.	a+b	
2.	ab	
3.	a^*	
4.	$(a+b)^*$	

Q. Construct FA equivalent to RE:
$(0+1)^* (00+11) (0+1)^*$

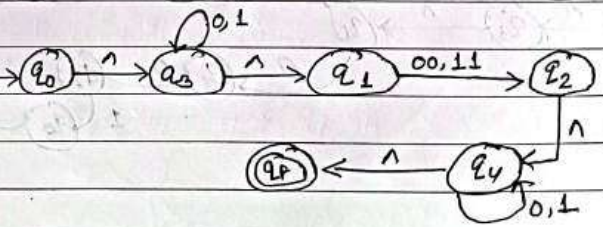
Step 1:



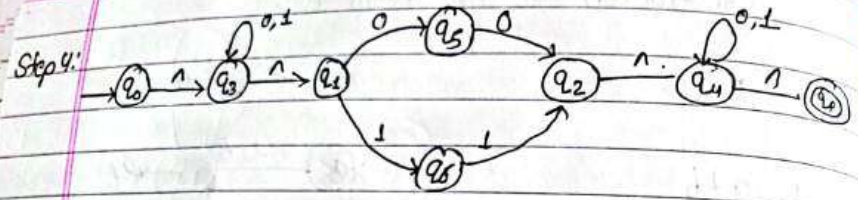
Step 2:



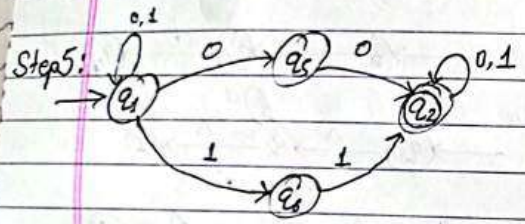
Step 3:



Step 4:

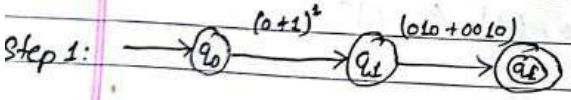


Step 5:

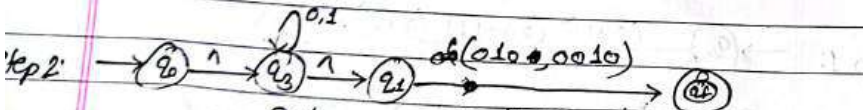


Q- Construct FA equivalent RE
$(0+1)^*\ (010 + 0010)$

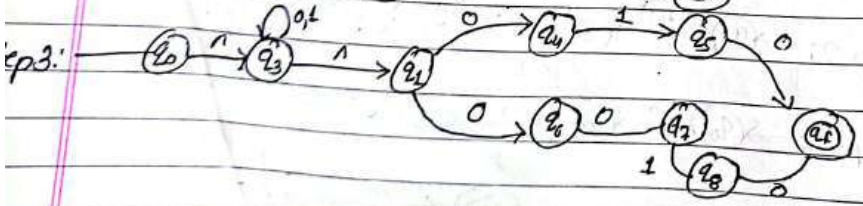
Step 1:



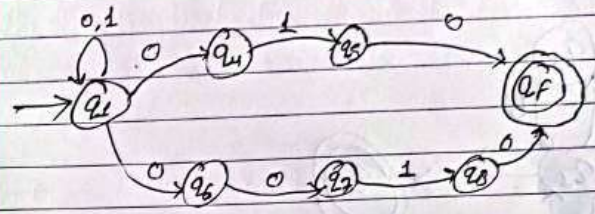
Step 2:



Step 3:



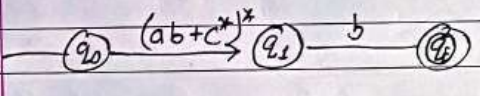
Step 4:



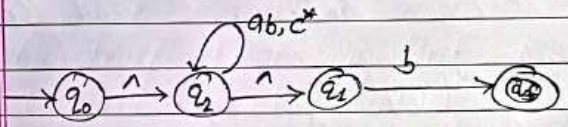
Assignment

Q. Construct FA equivalent to RE
$(ab + c^*)^*\ b$

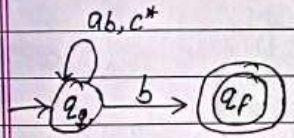
Step 1:



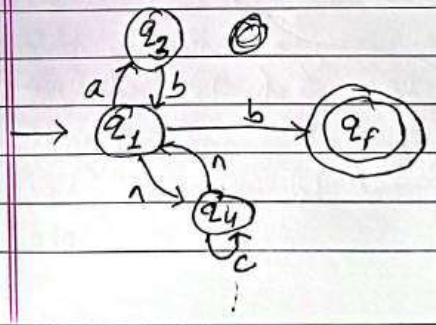
Step 2:



Step 3:

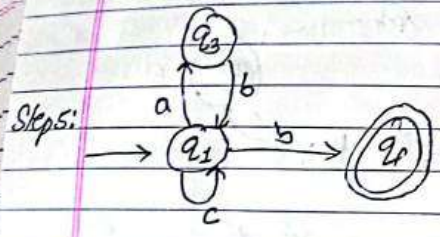


Step 4:



Assignment

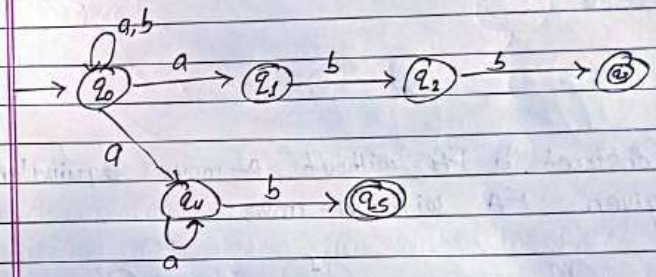
Steps:



2/2

→ or

Q) (a|b) (abb|a*b)



Elimination ~~for~~ of null move (∧-move)

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

Process:

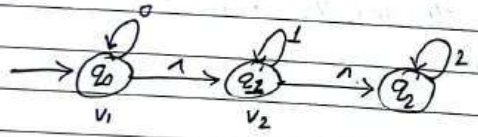
~~Replace~~

1) Find all the edges starting from V_2
2) Duplicate all edges of V_2 from V_1 without changing the edge ~~level~~ label.
3. If V_1 is initial state make V_2 as initial state.

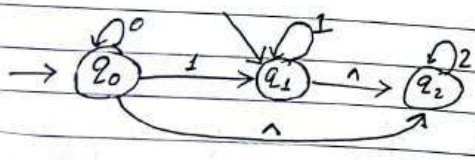
4. If V_2 is finial state make V_1 as final state

Q. Construct a FA without $\wedge$-move equivalent to given FA with $\wedge$-move.

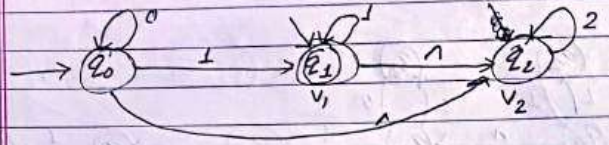
Step 1: Suppose $\wedge$-move between q_0 and q_1



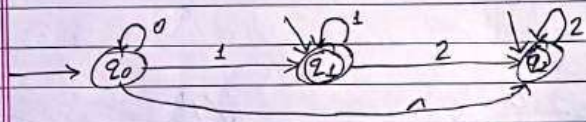
Step 2:



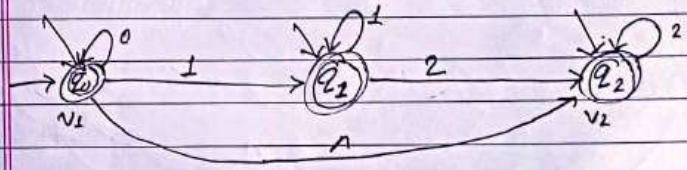
Step 3: Considering $\wedge$-move between q_1 and q_2



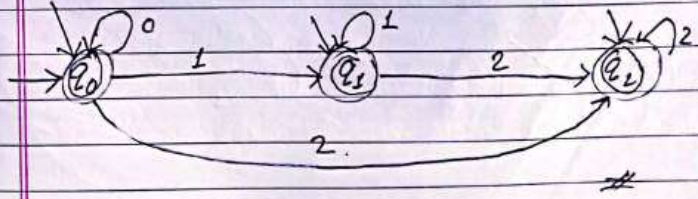
Step 4:



Step 5: Suppose $\wedge$-move between q_0 and q_2



Step 6:

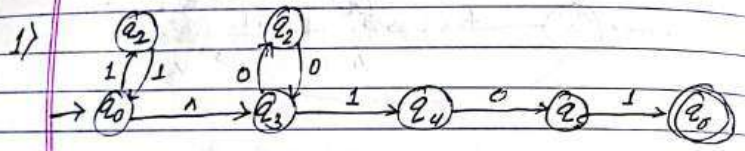


Assignment

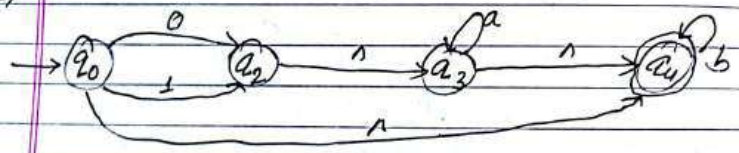
Date Page

Q) Construct a FA without ∧-move equivalent to given FA with ∧-move

1)



2)



2/2

Date Page

Q. Reduce the given NFA with NULL to without null move.

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

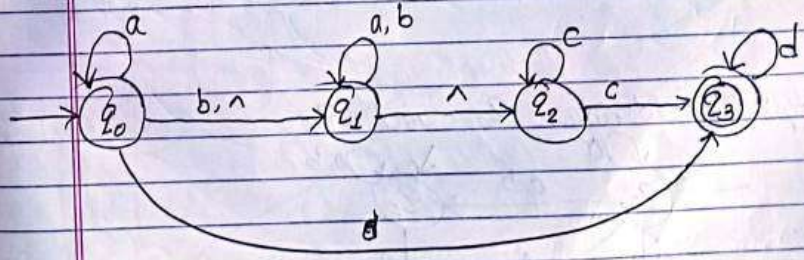
State table

State	Input a	b	c	d	∧
q_0	q_0	q_1	ϕ	q_3	q_1
q_1	q_1	q_1	ϕ	ϕ	q_2
q_2	ϕ	ϕ	$[q_2 q_3]$	ϕ	ϕ
q_3	ϕ	ϕ	ϕ	q_3	ϕ

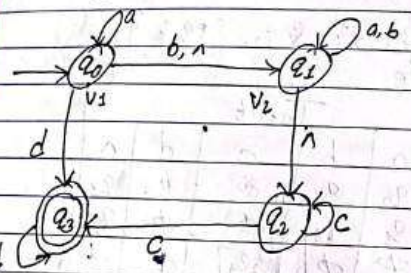
where $Q = \{q_0, q_1, q_2, q_3\}$

Solution

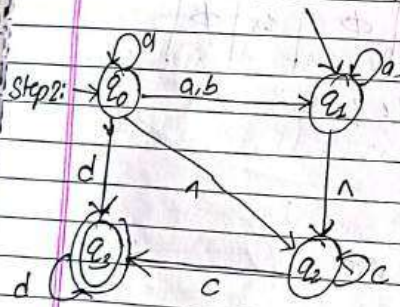
State diagram:



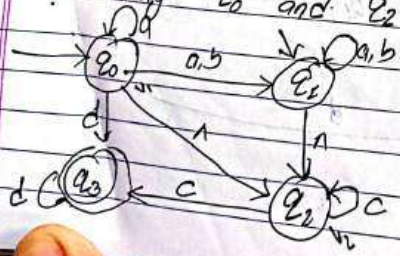
Step 1: Considering q_0 and q_1



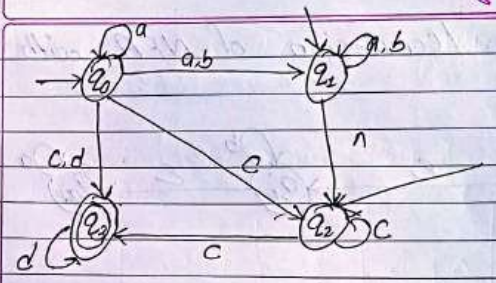
Step 2:



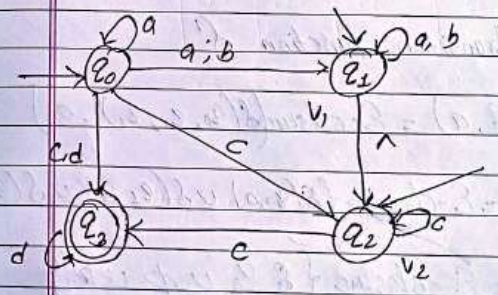
Step 3: Considering q_0 and q_2



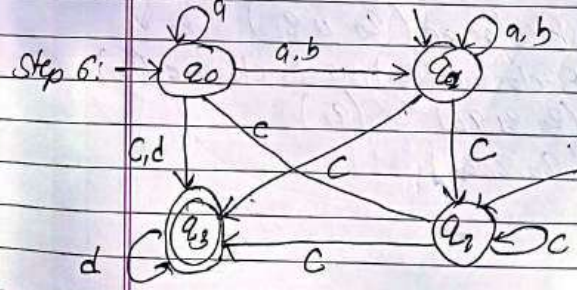
Step 4:



Step 5: Considering q_1 and q_2

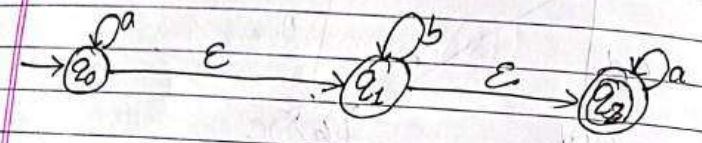


Step 6:



Note:
ε-closure $(\phi) = \phi$

Consider / Conversion of NFA with ε to DFA.



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

Now, DFA transition function δ',

1. $\delta'(\{q_0, q_1, q_2\}, a) = \varepsilon\text{-closure}(\delta(q_0, q_1, q_2), a)$
$= \varepsilon\text{-closure}\{\delta(q_0,a) \cup \delta(q_1,a) \cup \delta(q_2,a)\}$
$= \varepsilon\text{-closure}\{q_0 \cup \phi \cup a_2\}$
$= \varepsilon\text{-closure}(q_0 \cup q_2)$
$= \varepsilon\text{-closure}(q_0) \cup \varepsilon\text{-closure}(q_2)$
$= (q_0, q_1, q_2) \cup (q_2)$
$= (q_0, q_1, q_2)$

2. $\delta'(\{q_0, q_1, q_2\}, b) = \varepsilon\text{-closure}(\delta(q_0, q_1, q_2), b)$
$= \varepsilon\text{-closure}\{\delta(q_0,b) \cup \delta(q_1,b) \cup \delta(q_2,b)\}$
$= \varepsilon\text{-closure}\{\phi \cup q_1 \cup \phi\}$
$= \varepsilon\text{-closure}\{q_1\}$
$= \{q_1, q_2\}$ [New state]

3. $\delta'(\{q_1, q_2\}, a) = \varepsilon\text{-closure}(\delta(q_1,a) \cup \delta(q_2,a))$
$= \varepsilon\text{-closure}(\phi \cup q_2)$
$= \varepsilon\text{-closure}(q_2)$
$= \{q_2\}$ [New state]

4. $\delta'(\{q_1, q_2\}, b) = \varepsilon\text{-closure}\{\delta(q_1,b) \cup \delta(q_2,b)\}$
$= \varepsilon\text{-closure}\{q_1 \cup \phi\}$
$= \varepsilon\text{-closure}\{q_1\}$
$= \{q_1, q_2\}$

5. $\delta'[\{q_2\},a] = \varepsilon\text{-closure}(\delta(q_2,a))$

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

$= \{q_2\}$

6. $\delta'[\{q_2\},b] = \varepsilon\text{-closure}(\delta(q_2,b))$

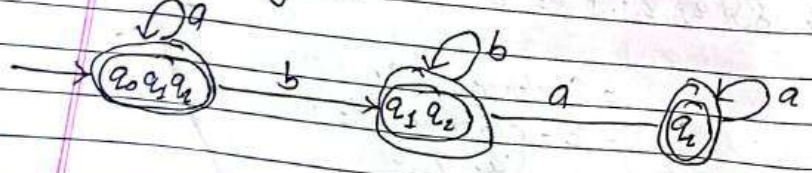
$= \varepsilon\text{-closure}(\phi)$

$= \phi$

Now transition state

State	Input a	Input b
$q_0\,q_1\,q_2$	$q_0\,q_1\,q_2$	$q_1\,q_2$
$q_1\,q_2$	q_2	$q_1\,q_2$
q_2	q_2	ϕ

∴ State diagram:



Conversion of NFA with ε to DFA

Q. Transition table for NFA, ε is

State	Input a	Input b	Input ε
$\to Q_0$	Q_0	$\{Q_0,Q_2\}$	Q_1
Q_1	Q_1	$\{Q_0,Q_1\}$	ϕ
Q_2	Q_3	ϕ	ϕ
(Q_3)	ϕ	ϕ	Q_4
(Q_4)	Q_3	ϕ	ϕ

Sol"

$\varepsilon\text{-closure}(Q_0) = \{Q_0, Q_1\}$

$(Q_1) = \{Q_1\}$

$(Q_2) = \{Q_2\}$

$(Q_3) = \{Q_3, Q_4\}$

$(Q_4) = \{Q_4\}$

Now transition function δ' for DFA

(1) $\delta'[\{Q_0,Q_1\},a]$

Note: Q_0 is starte of NFA so we take ε-closure of Q_0 as starting state of DFA

$= \varepsilon\text{-closure}(\delta\{(Q_0,Q_1),a\})$

$= \varepsilon\text{-closure}(Q_0 \cup Q_1)$

$= \varepsilon\text{-closure}(Q_0) \cup \varepsilon\text{-closure}(Q_1)$

$= \{Q_0,Q_1\} \cup \{Q_1\}$

$= \{Q_0, Q_1, Q_4\}$ [new state]

② $\delta'(\{Q_0, Q_1\}, b)$
$= \varepsilon\text{-closure}(\delta\{(Q_0, Q_1), b\})$
$= \varepsilon\text{-closure}(\delta(Q_0, b) \cup \delta(Q_1, b))$
$= \varepsilon\text{-closure}(\{Q_0, Q_2\} \cup \{Q_2, Q_4\})$
$= \varepsilon\text{-closure}(Q_0) \cup \varepsilon\text{-closure}(Q_2) \cup \varepsilon\text{-closure}(Q_2) \cup \varepsilon\text{-closure}(Q_4)$
$= \{Q_0, Q_1\} \cup \{Q_2\} \cup \{Q_2\} \cup \{Q_4\}$
$= \{Q_0, Q_1, Q_2, Q_4\}$ [new state]

③ $\delta'[\{Q_0, Q_1, Q_4\}, a]$
$= \varepsilon\text{-closure}(\delta(Q_0, Q_1, Q_4), a))$
$= \varepsilon\text{-closure}(Q_0 \cup Q_4 \cup Q_3)$
$= \{Q_0, Q_1\} \cup \{Q_4\} \cup \{Q_3, Q_4\}$
$= \{Q_0, Q_1, Q_3, Q_4\}$ [new state]

④ $\delta'(\{Q_0, Q_1, Q_4\}, b)$
$= \varepsilon\text{-closure}(\delta\{(Q_0, Q_1, Q_4), b\})$
$= \varepsilon\text{-closure}(\{Q_0, Q_2\} \cup \{Q_2, Q_4\} \cup \phi\}$
$= \{Q_0, Q_1, Q_2, Q_4\}$

⑤ $\delta'(\{Q_0, Q_1, Q_2, Q_4\}, a)$
$= \varepsilon\text{-closure}(\delta\{(Q_0, Q_1, Q_2, Q_4), a\})$
$= \varepsilon\text{-closure}(Q_0 \cup Q_4 \cup Q_3 \cup Q_3)$
$= \{Q_0, Q_1\} \cup \{Q_4\} \cup \{Q_3, Q_4\} \cup \{Q_3, Q_4\}$

$= \{Q_0, Q_1, Q_2, Q_4\}$

⑥ $\delta'(\{Q_0, Q_1, Q_2, Q_4\}, b)$
$= \varepsilon\text{-closure}\cdot\{\delta(Q_0, Q_1, Q_2, Q_4), b)$
$= \varepsilon\text{-closure}(\{Q_0, Q_2\} \cup \{Q_2, Q_4\} \cup \phi \cup \phi\}$
$= \{Q_0, Q_1, Q_2, Q_4\}$

⑦ $\delta'(\{Q_0, Q_1, Q_2, Q_4\}, a)$
$= \varepsilon\text{-closure}(\delta\{(Q_0, Q_1, Q_2, Q_4), a\})$
$= \varepsilon\text{-closure}(\{Q_0\} \cup \{Q_4\} \cup \phi \cup Q_3)$
$= \{Q_0, Q_1, Q_3, Q_4\}$

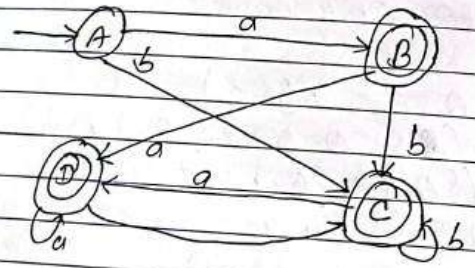
⑧ $\delta'(\{Q_0\ Q_1\ Q_3\ Q_4\}, b\}$
$= \varepsilon\text{-closure}(\delta\{(Q_0, Q_1, Q_3, Q_4), b\})$
$= \varepsilon\text{-closure}(\{Q_0, Q_2\} \cup \{Q_4, Q_4\} \cup \phi \cup \phi\}$
$= \{Q_0\ Q_1\ Q_2\ Q_4\}$

Transition table for DFA, δ' is

State	Input	
	a	b
→ $\{Q_0, Q_1\}$	$\{Q_0\ Q_1\ Q_4\}$	$\{Q_0\ Q_1\ Q_2\ Q_4\}$
* $\{Q_0\ Q_1\ Q_4\}$	$\{Q_0\ Q_1\ Q_3\ Q_4\}$	$\{Q_0\ Q_1\ Q_2\ Q_4\}$
* $\{Q_0\ Q_1\ Q_2\ Q_4\}$	$\{Q_0\ Q_1\ Q_3\ Q_4\}$	$\{Q_0\ Q_1\ Q_2\ Q_4\}$
* $\{Q_0\ Q_1\ Q_3\ Q_4\}$	$\{Q_0\ Q_1\ Q_3\ Q_4\}$	$\{Q_0\ Q_1\ Q_2\ Q_4\}$

Let $\{Q_0 Q_1\}^* = A$, $\{Q_0 Q_1 Q_4\}^* = B$,
$\{Q_0 Q_1 Q_2 Q_4\}^* = C$ and
$\{Q_0 Q_1 Q_3 Q_4\}^* = D$

DFA is



Imp The pumping lemma for regular sets (repitition)
→ to prove not regular

Theorem: Let $m = (Q, \Sigma, \delta, q_0, F)$ be a finite automata with 'd' state that accept regular language 'L'. Let w be the string such that $w \in L$ and $|w| \geq m$. If $m \geq n$, then there exists x, y, z (substring) such that
i) $w = xyz$
ii) $y \neq \Lambda$ (y cannot be null)
iii) $xy^i z \in L$ for each $i \geq 0$

Assignment

Properties of regular expression or language
1. Commutative
2. Associativity
3. Identities
4. Annihilator
5. Idempotent law
6. Distributive law
7. Law of closure

Algebric Rule / law for regular expression.

(1) Commutativity
This law means we can switch the order of its operand and get the same result. The union of regular expression is commutative but concatenation of regular expression is not commutative i.e. If r and s are regular expression representing like language $L(r)$ and $L(s)$ then $r+s = s+r$ i.e. $r \cup s = s \cup r$ but $r.s \neq s.r$

(2) Associativity
The union as well as concatenation of regular expression are associative i.e. if t, r, s are regular expression representing regular language

L(t), L(r) and L(s) then

$t + (r+s) = (t+r)+s$

and $t \cdot (r.s) = (t.r).s$

(3) Distributive law

For any regular expression r, s, t representing regular language L(r), L(s) and L(t) then

$r(s+t) = rs + rt$ — left distribution

$(s+t)r = sr + tr$ — right distribution

(4) Identity law

ϕ is identity for union i.e. for any regular expression representing regular expression L(r)

$r + \phi = \phi + r = r$ i.e. $\phi \cup r = r$

ε is identity for concatenation

i.e. $\varepsilon . r = r = r . \varepsilon$

(5) Annihilator

It is a value such that when the operator is applied to the annihilator and some other value, the result is annihilator

ϕ is annihilator for concatenation

i.e. $\phi . r = r . \phi = \phi$

(6) Idempotent law of union

→ For any regular expression r representing the regular language L(r), $r + r = r$. This is idempotent law of union

(7) Law of closure

– For any regular expression r, representing the regular language L(r) then

- $(r^*)^* = r^*$
- Closure of $\phi = \phi^* = \varepsilon$
- Closure of $\varepsilon = \varepsilon^* = \varepsilon$
- positive closure of r, $r^+ = rr^*$

Proof: Let $w = a_1, a_2, \ldots a_m$, $m \geq n$

$\delta(q_0, a_1 a_2 \ldots a_m) = q_i$ for $i = 1, 2, 3, \ldots m$

$Q = (q_0, q, q_2, \ldots q_m)$

Here Q is the sequence of states in the path with path value $w = a_1 a_2 \ldots a_m$

Now input string w can be decomposed into substring as

$x = a_1 a_2 \ldots a_j$

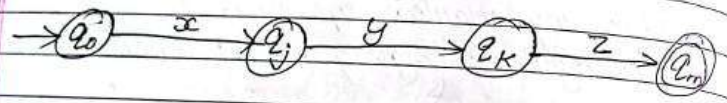
$y = a_{j+1} \ldots a_k$

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

Since we know that for any regular

expression there exists a path from initial state to final state with path value in the given regular expression.

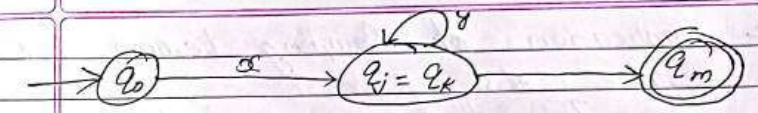
For ω, the finite automata can be constructed as shown in figure.



Here, length of string, $|\omega| = 3$ i.e. $m = 3$
no. of states, $n = 4$

On reading the string xyz, a new state are added on existing states. But by definition of 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 distinct state defined, but unapplying the input string, the state become n+1. Thus among various pair of repeated state, we take first as q_j and q_k for merging and hence the path with ω in the transition diagram of FA is shown in figure below:



Here length of string $|\omega| = 3$ i.e. $m = 3$
no. of state, $n = 3$
i.e. $m \geq n$ (True)

Since,
$\omega = xyz$
$= a_1 a_2, \ldots a_j a_{j+1} \ldots a_k a_{k+1} \ldots a_m a_{m+1}$

So we can write $0 \leq j \leq k \leq n$ and this implies that $|xy| \leq n$

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

* When $i = 0$, $\omega = xz$ (accepted by FA)
* When $i = 1$, $\omega = xyz$ (accepted by FA)
* when $i = 2$, $\omega = xy^2z$
$= xyyz$ (accepted by FA)

and so on.

Hence we can conclude that this condition i.e. $xy^i z \in L$ for each $i \geq 0$ is valid for all regular expression.

Thus, the string of y can pump in several number of time to generate large string. Hence $xy^i z \in L$. As every state in Q is obtained by applying an input symbol, $y \neq \varepsilon$.

Application of Pumping Lemma for regular Sets:

This theorem is used to prove that certain set of string are not regular. The steps needed for proving that a given set is not regular are:

Step1: Assume that L is regular. Let n be no. of state in FA.

Step2: Choose a string w such that $|w| \geq n$. Use pumping lemma to write, $w = xyz$, $|xy| \leq n$ and $|y| > 0$ or $|y| \geq 1$

Step3: Find a suitable i for which $xy^iz \notin L$. This contradicts our assumption. So, L is not regular.

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

Step1: Assume L is regular

Step2: Let $ww = \overbrace{a^n b}\ \overbrace{a^n b}$. Then

By pumping lemma, we can write $ww = xyz$, $|xy| \leq n$, $|y| > 0$

Suppose $n = 7$

$ww = a^7 b\, a^7 b$

$= aaaaaaab\ aaaaaaab$

Case 1: Y has only part of a

$\underbrace{aa}_{x}\ \underbrace{aaaaa}_{y}\ \underbrace{baaaaaaab}_{z}$

Case 2: Y has only b's

$\underbrace{aaaaaaab}_{x}\ \underbrace{aaaaaaab}_{y}\ \underbrace{}_{z}$

Case 3: Y has both a and b

$\underbrace{aaaaaaab}_{x}\ \underbrace{aaaaaaab}_{y}\ \underbrace{}_{z}$

Step3: We want to find i such that $xy^iz \notin L$

Take $i = 2$,

In case 1, xy^2z

i.e. $xy^2z = xyyz$

$= aaaaaaaaaaaaaa\ b\ aaaaaaab$

$= a^{12} b\, a^7 b$

This is not in pattern

Thus $xy^iz \notin L$ for $i = 2$

Also, in case 1, $|xy| = 7$, $n = 7$, $|xy| \leq n$

Take $i = 3$

In case 2, xy^3z

$= xyyyz$

$= a^7 b^3 a^7 b$

This is not in pattern

Thus $xy^iz \notin L$

Also

In case 2,

$|xy| = 8$, $n = 7$, $|xy| \leq n$ (false)

Q. Show that $L = \{0^n 1^n \mid n \geq 1\}$ is not regular

Soln

Step 1: Assume L is regular

Step 2: Let $w = 0^n 1^n$

Then by pumping lemma, we can write $w = xyz$ with $|xy| \leq n$, $|y| > 0$

Suppose, $n = 5$

$w = 0^5 1^5$

$= 0000011111$

Case I: y has 0's parts

i.e. 00 000 11111

x y z

Case 2: y has only 1's

i.e 00000 11111

x y z

Case 3: y has both 0's and 1's

i.e. 0000 01 1111

x y z

Step 3: We want to find i, such that $xy^iz \notin L$

Take $i = 2$

In case 1, xy^2z

$= xyyz$

$= 000000001111$

$= 0^8 1^5$

Here no. of 0's and 1's are not equal

i.e. not in pattern

So, $xy^iz \notin L$ for $i = 2$

Also, $|xy| = 5 \leq 5$ (True)

Take $i = 2$

In Case 2, xy^2z

$= xyyz$

$= 00000 1111111$

$= 0^5 1^7$

not in pattern

so, $xy^iz \notin L$ for $i = 2$

$|xy| = 7$, $n = 5$ (false)

Take $i = 2$

In case 3, xy^2z

$= xyyz$

$= 0000010111111$

$= 0^6 1^6$

not in pattern so, $xy^iz \notin L$ for $i=3$

$|xy| = 6$, $n = 5$ (false)

Thus in all cases, we get contradiction so given language.

$L = \{0^n 1^n \mid n \geq 1\}$ is not regular.

Assignment

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

Soln

Step 1: Assume 'L' is a regular language

Step 2: Let $w = a^p$, then by pumping lemma. We can write

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

Suppose,

$n = 7$

$w = a^7$

$= aaaaaaa$

case 1: y has only a i.e. $aaaaaaa$

Step 3: we want to find i such that $xy^iz \notin L$ for $i = 2$

In case 1, $xy^2z = aaaaaaaaaaaaaa$

$= a^{14}$

This is not prime. So not in pattern

$\therefore xy^iz \notin L$ also $|xy| = 7$, $n = 7$

$|xy| \leq n$ (true)

Closure properties of regular set / language

(2) If L_1 and L_2 are regular, then $L_1 \cdot L_2$ is also regular.

Proof:

Let $M_1 = (Q_1, \Sigma_1, \delta_1, q_1, F_1)$ be NFA that accept language L_1, i.e. $L_1 = L_1(M_1)$ and

$M_2 = (Q_2, \Sigma_2, \delta_2, q_2, F_2)$ be another NFA that accept language L_2, i.e. $L_2 = L_2(M_2)$

Now we construct NFA, $M = (Q, \Sigma, \delta, q, F)$ such that it we accept $L_1 \cdot L_2$ i.e.

$$L(M) = L_1(M_1) \cdot L_2(M_2)$$

where $Q = Q_1 \cup Q_2$

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

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

$q = q_1$

$F = F_2$

Formally,

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

if and only if, there exist $\omega_1 . \omega_2 \in \Sigma^*$ and $p_1 \in F_1$, $p_2 \in F_2$ such that

$w = w_1 \cdot w_2$, $(q_1, w_1) \xrightarrow[M_1]{*} (p_1, \varepsilon)$

and

$(q_2, w_2) \xrightarrow[M_2]{*} (p_2, \varepsilon)$

Hence $L(m) = L_1(m_1) \cdot L_2(m_2)$

3. If L_1 and L_2 are regular, the $L_1 \cap L_2$ is also regular.

Let $M_1 = (Q_1, \Sigma_1, \delta_1, q_1, F_1)$ be DFA that accept language L_1, i.e $L_1 = L_1(m_1)$ and

$M_2 = (Q_2, \Sigma_2, \delta_2, q_2, F_2)$ be DFA that accept language L_2 i.e. $L_2 = L_2(m_2)$.

We assume that alphabet of both automata are same and its Σ is union of alphabets of L_1 and L_2, If they are different

Now we construct FA, $M = \{Q, \Sigma, \delta, q, F\}$ such that it accept $L_1 \cap L_2$, i.e.

$L(m) = L_1(m_1) \cap L_2(m_2)$

where

$Q = Q_1 \times Q_2$ $\quad q = (q_1, q_2)$

$\Sigma = \Sigma_1 \cup \Sigma_2$ $\quad F = F_1 \times F_2$

$\delta = (\delta_1, \delta_2)$

If M_1 ~~else~~ goes from state P to state S on reading 'a' and M_2 goes from state Q to P on reading 'a' then M will go from state (P, q) to (S, t) on reading 'a'.

If $(P, a) \xrightarrow[M_1]{*} (S, \varepsilon)$ and $(q, a) \xrightarrow[M_2]{*} (t, \varepsilon)$ then

$(\{P, q\}, a) \xrightarrow[M]{*} (\{S, t\}, \varepsilon)$

i.e.

$\delta(\{P, Q\}, a) = \{\delta_1(P, a), \delta_2(q, a)\}$

Thus string is accepted by M if and only if w is accepted by M_1 and M_2.

Hence $L_1 \cap L_2$ is regular.

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

Proof

Let L be 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 have become non-final state of B and vice-versa.

Then,

w is in $L(B)$ if $\delta(q_0, w)$ is in $Q \setminus F$, which occurs if and only if w is not in $L(A)$

5. The closure operation (kleene closure) on a regular language is also regular.

Let $M_1 = \{Q_1, \Sigma_1, \delta_1, q_1, F_1\}$ be an NFA that accept regular language $L = L(M_1)$

Now

we construct NFA, $M = \{Q, \Sigma, \delta, q, F\}$ such that it can accept $L = L(M) = L(M_1)^*$

where

$Q = Q_1 \cup \{q_1\}$

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

$\delta = \delta_1 \cup [(q, \wedge) \to q_1, (q_1, \wedge) \to F_1, (F_1, \wedge) \to q]$

$q = \text{start state} = q$

$F = F_1 \cup \{q\}$

Here M consists of the state of M_1 and all transition of M_1, also any final state of M_1 is also final state of M. The new inital state is also final state so that Σ (or $\wedge$) is accepted.

If flows the inspection of M that if $\omega \in L(M)$ then either $\omega = n$ or $\omega = \omega_1\omega_2 \ldots \omega_k$ for $k \geq i$, $i = 1, 2, 3, \ldots k$

There is $F_i \in F$ such that $(q_1, \omega) \xrightarrow{*}_{M_1} (F_i, n)$

6. If L and M are regular then difference L/M is also regular.

Here $L/M = L \cap \overline{M}$

We already know that regular language goes under complement and intersection

Decision Algorithm for regular sets / language

Algorithm for regular language that tells whether or not following property hold.

Types:

1) The membership problem
2) The Emptiness problem
3) The infiniteness problem

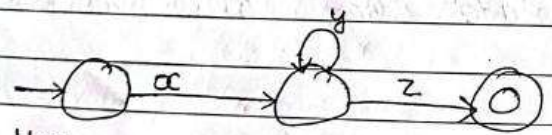
1) The membership problem
→ Is string ω in regular language L?
→ Assume L is represented by a DFA M
→ Processes the action of M on the sequence of input symbol forming ω
→ If processing the ω, M reaches the final state then ω is member of L.

2) The Emptiness problem
→ Given a regular language, those the language contain any string at all.
→ Assume a DFA for language L
→ Construct a transition graph
→ Compute the set of state reachable from the start state.

→ If for any string the final state is reached then the string is present in L else no.

g) The infinitness problem
→ Is a given regular language infinite?
→ Start with a DFA for the language L.
→ If the DFA has 'n' state and language contain any string of length n or more then the ~~String is present in L, else no.~~ language is infinite otherwise the language is finite.
→ If n-state DFA accept a string w of length n or more, then there must be a state that appears twice on the path labelled 'w' from start to final state

→



Here, $w = xyz$
Then, $xy^iz \in L$ for $i \geq 0$

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

Unit-3

Context-Free Language and Pushdown Automata

* Context free Language
↳ language generated by context free grammar
↳ Language accepted by PDA

* Context-Free Grammer
↳ A context-free grammar G is a quadruple (4-tuple) defined as $G = \{V, \Sigma, P, S\}$ where
V = set of variable or non-terminal symbol
Σ = set of terminal symbol
P = production rule
S = Starting Symbol
A CFG has production rule in the form, $A \rightarrow \alpha$
where $\alpha = \{V \cup \Sigma\}^*$ and $A \in V$

or

non-terminal → non-terminal (Capital letter)
non-terminal → terminal (represented by small letter/ digits / symbol)

Why context free grammar

① It provide a simple and mathematically precise mechanism for describing the method by which phrases in some natural language are built from smaller block

② Its simplicity make the formalism amenable to rigorous mathematical study

Eg: Consider $S \to aaAb$

A is non-terminal and a, b are terminal

Sentence → Noun Verb

Noun → Ram / Sita

Verb → goes / writes / sings

Eg: Construct a CFG for the language having any number of a's over $\Sigma = \{a\}$

⇒ we know, R.E for this

$RE = a^*$

Now production rule

$S \to \varepsilon$

$S \to aS$

If we want 'aaaa' string to be derived we can start with start symbol

$S \to aS$

$\to aaS \quad (S \to aS)$

$\to aaaS \quad (S \to aS)$

$\to aaaa\varepsilon \quad (S \to \varepsilon)$

$\to aaaa$

CFG for given problem is

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

where,

$V = \{S\}$

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

$S = \{S\}$

P is, $S \to \varepsilon \mid aS$

2. Construct a CFG for RE $(0+1)^*$

⇒ Here R.E. $= (0+1)^*$

Possible string are $\{\varepsilon, 0, 1, 01, 10, \ldots\}$

Production rule is

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

3. Construct a CFG for string having at least two a's over $\Sigma = \{a, b\}$

⇒ Here $RE = (a+b)^* a (a+b)^* a (a+b)^*$

Possible string are $\{aa, aba, baaa, \ldots\}$

Now,

Production rule

$S \to AaAaA$

$A \to \varepsilon \mid aA \mid bA$

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

⇒ Here R.E = $0(0+1)^* 1$

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

Possible string are $\{01, 10, 001, 110, \ldots\}$

Thus production rule,

$S \rightarrow 0A1$

$S \rightarrow 1A0$

$A \rightarrow \varepsilon \mid 0A \mid 1A$

5. Construct a CFG for language $L = \omega c \omega^T$, $\omega \in (a,b)^*$, ω^T = reverse

⇒

Production rule;

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

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

⇒ Possible string are $L = \{ab, aabb, abb, \ldots\}$

Now, Production Rules

$A \rightarrow a \mid aA$

$B \rightarrow b \mid bB$

$S \rightarrow AB$

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

⇒ Possible string, $L = \{a, ab, aab, aabb, \ldots\}$

Now, Production Rules

$A \rightarrow a \mid aA$

$B \rightarrow \varepsilon \mid bB$

$S \rightarrow AB$

8. Construct a CFG for $L = \{a^n b^m \mid (n+m) \text{ is even}\}$

⇒ As we know,

$a^* = \{\varepsilon, a, aa, aaa, \ldots\}$ [not all even]

$(aa)^* = \{\varepsilon, aa, aaaa, \ldots\}$ [all even]

$a(aa)^* = \{a, aaa, aaaaa, \ldots\}$ [all odd]

Also even + even = even

~~even~~ odd + odd = even

R.E for $L = \{a^n b^m \mid n+m \text{ is even}\}$ is

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

Production Rule for $(aa)^*$ is

$A \rightarrow \varepsilon \mid aaA$

Production rule for $(bb)^*$ is

$B \rightarrow \varepsilon \mid bbB$

Now, CFG, For $L = \{a^n b^m \mid n+m \text{ is odd}\}$ is

$S \rightarrow AbB \mid aAB$

$A \rightarrow \varepsilon \mid aaA$

$B \rightarrow \varepsilon \mid bbB$

9. Construct a CFG for $L = \{a^n b^m \mid n \neq m\}$ is odd
⇒ [Same as 8]

Assignment

Q. Construct CFG for string whose length is odd over $\Sigma = \{a, b\}$

Q. Construct CFG for string having same starting and ending symbol over $\Sigma = \{a, b\}$

Derivation of CFG
↳ process of deriving strings from grammar

Eg: Derive a^4 from grammar
$S \to aS \mid \varepsilon$

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

Hence, a^4 is accepted by grammar
The language accepted by grammar is
$L = \{a^n / n \geq 0\}$

Q. Let $G = \{(S, C), (a, b), P, S\}$, where P is
$S \to aCa$
$C \to aCa \mid b$
And $L(G)$ i.e language accepted by grammar (CFL)

⇒
$S \to aCa$
$\to abaa$ [$C \to b$]

$abaa \in L(G)$
Also, $S \to aCa$
$\to aaCaa$ [$C \to aCa$]

→ aabaa [C→b]

∴ aabaa ∈ L(G)

Similarly,

$S \xrightarrow{*} aaabaaa \in L(G)$

$S \xrightarrow{*} aaaabaaaa \in L(G)$

and so on.

So the language is

$L = \{a^n b a^n \mid n \geq 1\}$

Q. Find L(G) for S→ as|bs|a|b

⇒ S→ a

S→ b

→ as

→ aa [S→a]

S→ bS

→ bb [S→b]

Similarly, $S \xrightarrow{*} aba$

$S \xrightarrow{*} aabaa$

and so on

So, the language is $L = \{a^n b^m \mid (n+m) \geq 1\}$

Derivation can be classified as:

1. Left Most Derivation
   If we apply rule at left most variable every time

2. Right Most Derivation
   If we apply rule at left most variable every time

3. Mixed Most Derivation
   Combination of left and right most

Derivation Tree
→ representation of derivation using tree
→ also called parse tree

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

1. Every vertex has a label which is a variable or a terminal or null (ε or ∧)
2. The root has label S
3. The label of an internal vertex is a variable.

Assignment

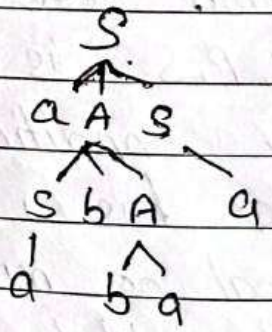
Date
Page

4. If the vertices $n_1, n_2, \ldots n_k$ written with label $x_1, x_2, \ldots x_k$ are the child nodes of vertex n with label 'A', then
$A \rightarrow x_1 x_2 \ldots x_k$ is a production rule.

Q. Consider a grammar G which has
$S \rightarrow aAS \mid a$
$A \rightarrow SbA \mid SS \mid ba$
Construct a parse tree which yeild 'aabbaa' where S is starting symbol

⇒ $S \rightarrow aAS$
$\rightarrow aSbAS$ [A → .SbA]
$\rightarrow aabAS$ [S → a]
$\rightarrow aabbaS$ [A → ba]
$\rightarrow aabbaa$ [S → a]

Parse Tree:-



Date Page

(2) Derive the string 'aabbabba'

$S \rightarrow aB \mid bA$

$A \rightarrow a \mid aS \mid bAA$

$B \rightarrow b \mid bS \mid aBB$

Soln

(1) Left most derivation

$S \rightarrow aB$ [S → aB]

$\rightarrow aaBB$ [B → aBB]

$\rightarrow aabSB$ [B → bS]

$\rightarrow aabbAB$ [S → bA]

$\rightarrow aabbaB$ [A → a]

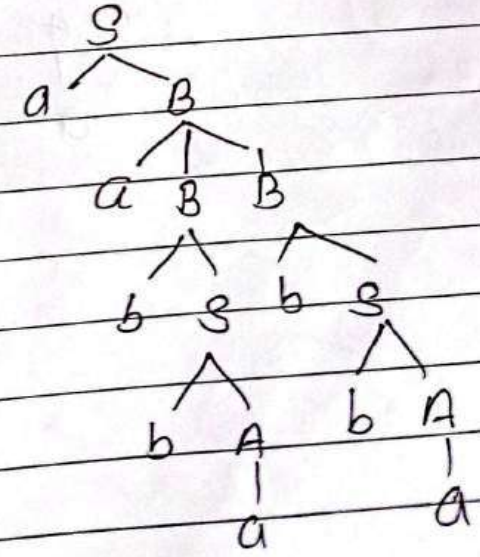
~~$\rightarrow aabbabAB$ [S → bA]~~

$\rightarrow aabbabS$ [B → bS]

$\rightarrow aabbabbA$ [S → bA]

$\rightarrow aabbabba$ [A → a]

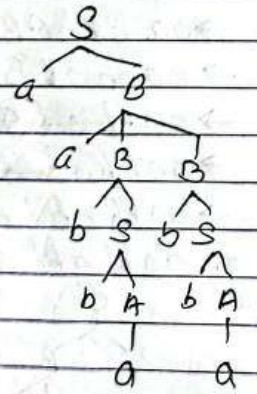
Parse tree:



② Right most derivation

$S \to aB$ [S → aB]
$\to aaBB$ [B → aBB]
$\to aaBbS$ [B → bS]
$\to aaBbbA$ [S → bA]
$\to aaBbba$ [A → a]
$\to aabSbba$ [B → bS]
$\to aabbAbba$ [S → bA]
$\to aabbabba$ [A → a]

Parse tree:



Q. Derive the string "00110101"

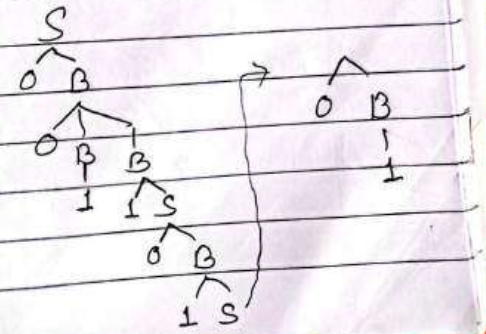
$S \to 0B \mid 1A$
$A \to 0 \mid 0S \mid 1AA$
$B \to 1 \mid 1S \mid 0BB$

by
① Left most derivation
② Right most derivation
③ Mixed most derivation
④ make parse tree of each derivation

① Left most derivation

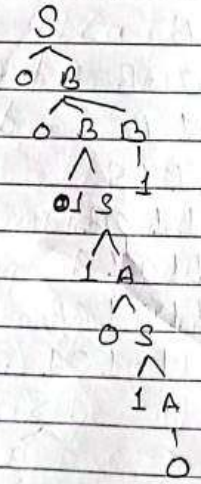
$S \to 0B$ [S → 0B]
$\to 00BB$ [B → 0BB]
$\to 001B$ [B → 1]
$\to 0011S$ [B → 1S]
$\to 00110B$ [S → 0B]
$\to 001101S$ [B → 1S]
$\to 0011010B$ [S → 0B]
$\to 00110101$ [B → 1]

Parse tree:



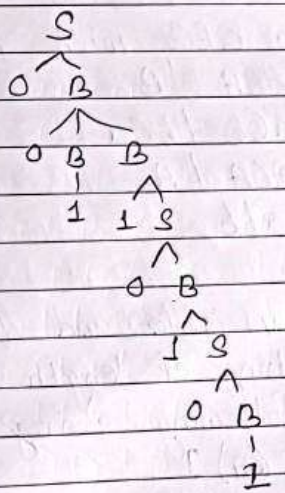
2. Right Most derivation

$S \rightarrow 0B$ [S → 0B]
$\rightarrow 00BB$ [B → 0BB]
$\rightarrow 00B1$ [B → 1]
$\rightarrow 001S1$ [B → 1S]
$\rightarrow 0011A1$ [S → 1A]
$\rightarrow 00110S1$ [A → 0S]
$\rightarrow 001101A1$ [S → 1A]
$\rightarrow 00110101$ [A → 0]



(B) Mixed Most derivation:

$S \rightarrow 0B$ [S → 0B]
$\rightarrow 00BB$ [B → 0BB]
$\rightarrow 001 1S$ [B → 1, B → 1S]
$\rightarrow 0011 0B$ [S → 0B]
$\rightarrow 00110 1S$ [B → 1S]
$\rightarrow 001101 0B$ [S → 0B]
$\rightarrow 0011010 1$ [B → 1]



Assignment

Q. Derive the string "aaabbabbba" by
$S \to aB \mid bA$
$A \to aS \mid bAA \mid a$
$B \to bS \mid aBB \mid b$
① Left most derivation
② Right most derivation
③ Mixed most derivation
④ Make parse tree of each derivation

Q. Consider a CFG with following production
$S \to ASA \mid B$
$B \to aCb \mid bCa$
$C \to ACA \mid A$
$A \to a \mid b$

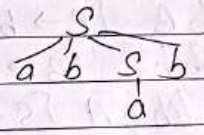
a) What are variable and terminal in G?
b) Give the string of length '7' in L(G)
c) Are the following string in Language of G i.e. L(G)?
i) aaa ii) bbb iii) aba iv) abb
d) True or False, $C \to bab$
e) True or False, $C \xrightarrow{*} bab$
f) True or False, $C \xrightarrow{*} AAA$

Ambiguity in CFG

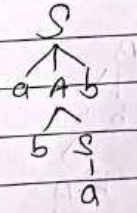
→ A terminal string $\omega \in L(G)$ is ambiguous if there exist more than two derivation tree for same string i.e. ω

Eg: Show that the grammar
$S \to a \mid abSb \mid aAb$
$A \to bS \mid aAAb$ is ambiguous.

1st method
$S \to abSb$ [$S \to abSb$]
$\to abab$ [$S \to a$]



2nd method
$S \to aAb$ [$S \to aAb$]
$\to abSb$ [$A \to bS$]
$\to abab$ [$S \to a$]



Since more than two derivation for "abab"
So, grammar is ambiguous.

Assignment

Q. Derive the string "aaabbabbba" by

$S \rightarrow aB \mid bA$

$A \rightarrow aS \mid bAA \mid a$

$B \rightarrow bS \mid aBB \mid b$

i) Left most derivation

$S \rightarrow aB$ [S→aB]

$\rightarrow aaBB$ [B→aBB]

$\rightarrow aaaBBB$ [B→aBB]

$\rightarrow aaabBB$ [B→b]

$\rightarrow aaabbB$ [B→b]

$\rightarrow aaabbaBB$ [B→aBB]

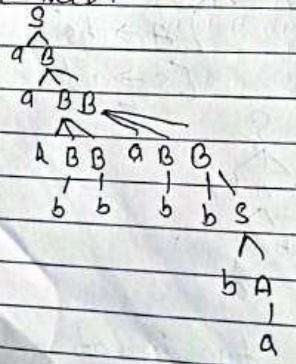
$\rightarrow aaabbabB$ [B→b]

$\rightarrow aaabbabbS$ [B→bS]

$\rightarrow aaabbabbbA$ [S→bA]

$\rightarrow aaabbabbba$ [A→a]

Parse tree:



ii) Right most derivation

$S \rightarrow aB$ [S→aB]

$\rightarrow aaBB$ [B→aBB]

$\rightarrow aaBaBB$ [B→aBB]

$\rightarrow aaBaBbS$ [B→bS]

$\rightarrow aaBaBbbA$ [S→bA]

$\rightarrow aaBaBbba$ [A→a]

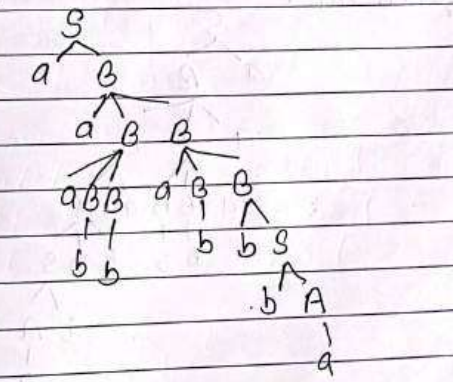
$\rightarrow aaBabbba$ [B→b]

$\rightarrow aaaBBabbba$ [B→aBB]

$\rightarrow aaaBbabbba$ [B→b]

$\rightarrow aaabbabbba$ [B→b]

Parse tree:

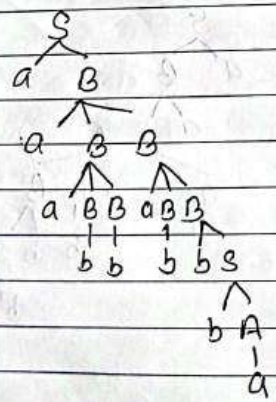


Assignment

iii) Mixed most Derivation

$S \to aB$ [$S \to aB$]
$\to aaBB$ [$B \to aBB$]
$\to aabbS$ [$B \to b$, $B \to bS$]
$\to aabbaB$ [$S \to aB$]
$\to aabbaaBB$ [$B \to aBB$]
$\to aabbaa$
$\to aaaBBbaBB$ [$B \to aBB$, $B \to aBB$]
$\to aaabbabbS$ [$B \to b$, $B \to b$, $B \to b$, $B \to bS$]
$\to aaabbabbbA$ [$S \to bA$]
$\to aaabbabbba$ [$A \to a$]

parse tree:



Q. Consider a CFG with following production
$S \to ASA \mid B$
$B \to aCb \mid bCa$
$C \to ACA \mid A$
$A \to a \mid b$

a) What are the variable and terminal in G?
⇒ Variable : A, S, B, C,
Terminal : a, b

b) Give the string of length '7' in L(G)?
⇒ aabbabb

c) Are the following string in language of G i.e. L(G)?
i) aaa ii) bbb iii) aba iv) abb
⇒ Yes, there is a string in language
i.e. abb

2/14

Q. Show that grammar G is ambiguous

$S \to aB \mid ab$

$B \to ABb \mid b$

$A \to aAB \mid a$

1st method:

$S \longrightarrow aB \quad [S \to aB]$

$\longrightarrow aABb \quad [B \to ABb]$

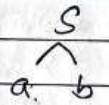
$\longrightarrow aaABBb \quad [A \to aAB]$

$\longrightarrow aaaBBb \quad [A \to a]$

$\longrightarrow aabb \quad [B \to b]$

2nd method:

$S \longrightarrow ab \quad [S \to ab]$



Since two different derivation tree. So grammar is ambiguous

2/14

Q. If G is the grammar, $S \to SbS \mid a$ Show that grammar is ambiguous for abababa string

1st method:

$S \to SbS \quad [S \to SbS]$

$\to abS \quad [S \to a]$

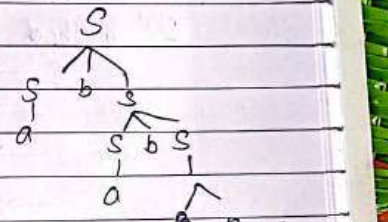
$\to abSbS \quad [S \to SbS]$

$\to ababS \quad [S \to a]$

$\to ababSbS \quad [S \to SbS]$

$\to ababaBS \quad [S \to a]$

$\to abababa \quad [S \to a]$



2nd method:

$S \to SbS \quad [S \to SbS]$

$\to SbS \quad [S \to a]$

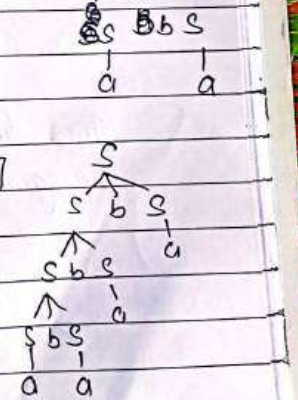
$\to SbSba \quad [S \to SbS]$

$\to SbabSba \quad [S \to a]$

$\to SbabaSba \quad [S \to SbS]$

$\to SbabaSba \quad [S \to a]$

$\to abababa \quad [S \to a]$



Q. Show that grammar G is ambiguous

$S \rightarrow as \mid asb \mid X$

$X \rightarrow Xa \mid a$

1st method

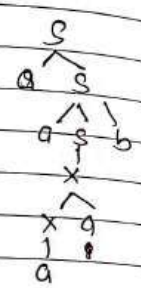
$S \rightarrow as$ [S → as]

$\rightarrow aasb$ [S → asb]

$\rightarrow aaXb$ [S → X]

$\rightarrow aaXab$ [X → Xa]

$\rightarrow aaaab$ [X → a]



2nd method

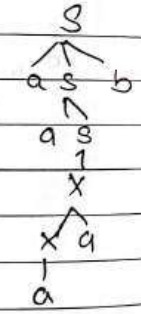
$S \rightarrow asb$

$\rightarrow aaSb$ [S → as]

$\rightarrow aaXb$ [S → X]

$\rightarrow aaXab$ [X → Xa]

$\rightarrow aaaab$ [X → a]



Since two different derivation tree

So grammar is ambiguous

Q. If G is the grammar for palindrom

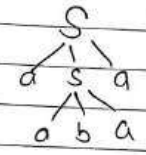
$S \rightarrow aSa \mid bSb \mid a \mid b \mid \varepsilon$

Check for ambiguity

1st method

$S \rightarrow asa$ [S → asa]

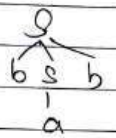
$S \rightarrow aba$ [S → b]



2nd method

$S \rightarrow bSb$ [S → bSb]

$\rightarrow bab$ [S → b]



Here ~~three~~ no different derivation tree

So, grammar is no ambiguity

Assignment

Date
Page

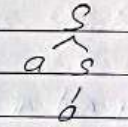
1 Q) Show that grammar G is ambiguous.

$S \rightarrow as \mid Sa \mid a$

1st method:

$S \rightarrow as$

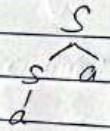
$\rightarrow aa \quad [S \rightarrow a]$



2nd method:

$S \rightarrow Sa$

$\rightarrow aa \quad [S \rightarrow a]$



Since more than two derivation form. So, grammar is ambiguous.

Assignment

Date
Page

2. Check for ambigity.

$S \rightarrow iCtS \mid iCtSeS \mid a$

$C \rightarrow b$

$S = iCtS$

$\Rightarrow ibta \quad [C \rightarrow b, \; S \rightarrow a]$

It can't be derived from any other way. So it is not ambiguous.

Simplification of Grammar

↳ removing all unnecessary symbol
1. Removal of useless symbol
2. Removal of NULL Symbol
3. Removal of Unit Symbol

1) Removal of useless Symbol (Reduction of CFG)

CFG are reduced in two phases:

Phase 1:

Derivation of CFG, G' from G, such that each variable derive some terminal string

Procedure:

Step1: Include all symbols W_i that derived some terminal and $i=1$
Step2: Include all symbols W_{i+1} that derives W_i
Step3: Implement increment i and repeat Step2 until $W_{i+1} = W_i$

Step4: Include all production rule that have W_i in it

Phase 2:

Derivation of CFG, G'' from G' such that each symbol appears in sentential form (derives from starts Symbol)

Procedure:

Step1: Include the start symbol in Y_i and $i=1$
Step2: Include all symbol Y_{i+1} that can be derived from Y_i and includ all production rule that have been applied
Step3: Increment i and repeat step2 until $Y_{i+1} = Y_i$

1. Removal of useless Symbol

Q. Find a reduced grammar equivalent to G having production rule P as

$S \rightarrow AC \mid B$
$A \rightarrow a$
$C \rightarrow c \mid BC$
$E \rightarrow aA \mid e$

Phase 1:

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

$W_1 = \{A, C, E\}$

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

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

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

where

$P = S \to AC,\ A \to a,\ C \to c,\ E \to aA/e$

Phase 2:

$Y_1 = \{S\}$

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

$Y_3 = \{S, A, C, ac\}$

$Y_4 = \{S, A, C, ac\}$

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

where

$P: S \to AC$

$A \to a$

$C \to c$

Hence is reduced grammar

Q Find a reduced grammar equivalent to G having production rule P as

$S \to ABa/BC,\ A \to BCC/\epsilon,\ C \to c,\ B \to bcc$

$D \to E,\ E \to d,\ F \to e$

Phase 1:

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

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

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

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

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

where

$P = S \to ABa/BC$

$A \to BCC/\epsilon$

$C \to c$

$B \to bcc$

$D \to E$

$E \to d$

$F \to e$

Now Phase 2:

$Y_1 = \{S\}$

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

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

$V_4 = \{S, A, B, C, a, b, C\}$

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

where

$P = S \to ABa \mid BC$

$A \to aC \mid BCC$

$C \to a$

$B \to bcc$

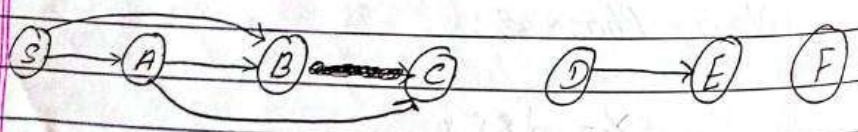
So this is the reduced grammar.

Alternate method:

Step 1: Eliminate non-generating symbol, Here all symbol are generating.

Step 2: Eliminat non-reachable symbol (variable)

Draw dependency graph



Assignment

Here D, E and F are non-reachable from S

After removing useless symbol, we get

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

$B \to bcc$

Q. Find a reduced grammar equivalent to G having production rule P as

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

$S \to BC \mid AB \qquad C \to aB \mid b$

Phase 1:

$T = \{a, b\}$

$W_1 = \{A, C\}$

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

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

$W_4 = \{A, C, S\}$

where

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

where

$P = S \to AC$

$A \to a$

$C \to aB \mid b$

Phase 2:

$Y_1 = \{S\}$

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

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

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

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

where

$P = S \to AC$

$A \to a$

$C \to aB \mid b$

So this is reduced grammar.

Find a reduced grammar equivalent to G having production rule P as

$S \to aAa$

$B \to ab$

$A \to bBB$

$C \to ab$

Here C is non-reable from S. So, remove it. we get,

$S \to aAa$

$A \to bBB$

$B \to ab$

$S \to as \mid A \mid BC$

$A \to a$

$B \to aa$

$C \to aCb$

Here C is useless as it does not derive any string.

After removing C. we get

$S \to as \mid A$

$A \to a$

$B \to aa$

Again

B is non-reachable from S. So remove it. we get

$S \to as \mid A$

$A \to a$

2. Reduction of Unit Production ($A \to B$)

↳ Any production rule of the form $A \to B$ where $A, B \in$ non terminal are called Unit production.

↳ Procedure

Step1: To remove production $A \to B$, add $A \to x$ to grammar rule whenever $B \to x$ ($x \in$ terminal or $x \in$ null)

Step2: Remove $A \to B$ from grammar

Step3: Repeat ~~Set~~ step 1 and 2 until all unit production are removed

(2). Remove unit production from given grammar

P: $S \to XY$, $X \to a$, $Y \to z|b$, $z \to m$
$m \to N$, $N \to a$

Sol" Here, unit production are
$Y \to z$, $z \to m$, $m \to N$

(1) Since $N \to a$, add $m \to a$, we get

P: $S \to XY$, $X \to a$, $Y \to z|b$, $z \to m$, $m \to a$,
$N \to a$

(2) Since $m \to a$, add $z \to a$, we get

P: $S \to XY$, $X \to a$, $Y \to z|b$, $z \to a$, $m \to a$,
$N \to a$

(3) Since $z \to a$, add $Y \to a$, we get

P: $S \to XY$, $X \to a$, $Y \to a|b$, $z \to a$, $m \to a$
$N \to a$

Also remove non-reachable Symbol
Here M, N, Z are non-reachable. So reduced Grammar is

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

Date Page

Q. Remove unit production from given grammar
$S \rightarrow A/bb$
$A \rightarrow B/b$
$B \rightarrow S/a$

⇒ Here unit production are
$S \rightarrow A$, $B \rightarrow S$, $A \rightarrow B$

① Since $B \rightarrow a$ add $A \rightarrow a$ we get
P: $S \rightarrow A/bb$
$B \rightarrow S/a$
$A \rightarrow a/b$

② Since $S \rightarrow bb$ add $B \rightarrow bb$ we get
P: $S \rightarrow A/bb$
$B \rightarrow bb/a$
$A \rightarrow a/b$

③ Since $A \rightarrow a/b$ add $S \rightarrow a/b$ we get
$S \rightarrow a/b/bb$
$A \rightarrow a/b$
$B \rightarrow bb/a$

Here B, A are not reachable so remove it the the grammar is
$S \rightarrow a/b/bb$

Date Page

Elimination of NULL production

In CFG or non-terminal symbol 'A' is null production if $A \rightarrow \varepsilon$ (or $A \rightarrow \wedge$) or a production starting at A leads to ε or $\wedge$ (i.e. $A \rightarrow \cdots \varepsilon$ or $A \xrightarrow{*} \varepsilon$)

Procedure:

1. To remove $A \rightarrow \varepsilon$, look for all production whose RHS contain A.
2. Replace each occurance of A in each of those production with ε
3. Add the resultant production to the Grammar.

Q. Eliminate null production from grammar
$S \rightarrow ABAC$, $A \rightarrow aA/\varepsilon$, $B \rightarrow bB/\varepsilon$
$C \rightarrow c$

Sol^n Here null production $A \rightarrow \varepsilon$, $B \rightarrow \varepsilon$

① Remove $A \rightarrow \varepsilon$

$S \rightarrow ABAC$
$\rightarrow \varepsilon BAC$
$\rightarrow BAC$

$S \rightarrow ABAC$
$\rightarrow ABEC$
$\rightarrow ABC$

$S \rightarrow ABAC$
$\rightarrow \varepsilon B \varepsilon C$
$\rightarrow BC$

(iv) $A \to aA$
$\to aE$
$\to a$

now, new production rule are
$S \to ABAC \mid BAC \mid ABC \mid BC$
$A \to aA \mid a$, $B \to bB \mid E$, $C \to c$

① Remove $B \to E$

① $S \to ABAC$
$\to AEAC$
$\to AAC$

② $S \to BAC$
$\to EAC$
$\to AC$

③ $S \to ABC$
$\to AEC$
$\to AC$

④ $S \to BC$
$\to EC$
$\to C$

⑤ $B \to bB$
$\to bE$
$\to b$

now, production rule are
$S \to ABAC \mid AAC \mid BAC \mid BC \mid bAAC \mid AC \mid C$
$A \to aA \mid a$
$B \to bB \mid b$
$C \to c$

Assignment

Q. Eliminate null production from grammar
$S \to ABaC$, $A \to BC$, $B \to b \mid E$, $C \to D \mid E$
$D \to d$

Here null production.
$B \to E$
$C \to E$

① Remove $B \to E$
$S \to ABaC$
$\to AaC$

$A \to BC$
$\to C$

New P: $S \to ABaC \mid AaC$
$A \to BC \mid C$
$B \to b$
$C \to D \mid E$
$D \to d$

② Removing $C \to E$

P: $S \to ABa \mid Aa \mid ABaC \mid AaC$
$B \to b$
$A \to B \mid E \mid BC \mid C$
$C \to D$
$D \to d$

Assignment

Again Removing $A \to \varepsilon$

Pi

$S \to ABaC \mid AaC \mid ABa \mid Aa \mid BaC \mid ac \mid Ba \mid a$

$A \to BC \mid C \mid B$

$B \to b$

$C \to D$

$D \to d$

Q. Simplify the grammar

$S \to aA \mid aBB$

$A \to aAA \mid \varepsilon$

$B \to bB \mid bbC$

$C \to B$

(1) Remove

(a) NULL

(b) Unit

(c) Useless

(1) Eliminating ε-production given result grammar as

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

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

$B \to bB \mid bbC$

$C \to B$

(2) Removing unit production gives resulting grammar

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

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

$B \to bB \mid bbC$

$C \to bB \mid bbC$

(c) Removing useless symbol gives resulting grammar as

Here B and C are useless Because B and C do not drive any string. After removing useless, symbol we get.

$S \rightarrow aA \mid a$

$A \rightarrow aAA \mid aA \mid a$

This is simplified grammar.

Normal Form

↳ production rule that has certain restriction

Types:

① Chomsky Normal Form (CNF)
② Greibach Normal Form (GNF)

Imp

① Chomsky Normal Form (CNF)

A ~~CNF~~ CFG is in CNF if production rule are of the following form:

$$A \rightarrow a$$

$$A \rightarrow BC$$

where A, B, C are variable and 'a' is terminal

Procedure for converting into CNF

Step 1: If START symbol 'S' occurs on some right side, take a new start symbol 'S'' and a new production

$$S' \rightarrow S$$

Step 2: Remove null production

Step 3: Remove unit production

Step 4: Remove string of terminals on right hand side of production if it exceeds as follows:

Suppose we have production

$$S \longrightarrow a_1 a_2 a_3$$

Then introduce non-terminal C_{a_i} as,

$C_{a_1} \rightarrow a_1$, $C_{a_2} \rightarrow a_2$, $C_{a_3} \rightarrow a_3$

Step 5: To restrict no. of variables on RHS, introduced new variable and seperate them as follows:

Suppose we have production with 'n' non-terminal as shown below with S non-terminal

$Y \rightarrow X_1 X_2 X_3 X_4 X_5$

Add n-2 new production rule using n-2 new non-terminals and ~~ver~~ modify the production as below:

$Y \rightarrow X_1 R_1$
$R_1 \rightarrow X_2 R_2$ where R_1, R_2, R_3 are new
$R_2 \rightarrow X_3 R_3$ non terminal
$R_3 \rightarrow X_4 X_5$

Step 6: If the right side any production is in the form $A \rightarrow aB$, where a is terminal then the production is replaced by $A \rightarrow XB$, $X \rightarrow a$

Q. Convert the following CFG into CNF

P: $S \rightarrow ASA \mid aB$
$A \rightarrow B \mid S$
$B \rightarrow b \mid \varepsilon$

Sol^n

1. Since S appears in RHS, introduce new start symbol S' and add $S' \rightarrow S$ to production rule.

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

2. Remove null production : $B \rightarrow \varepsilon$

After removing : $B \rightarrow \varepsilon$

P: $S' \rightarrow S$
$S \rightarrow ASA \mid aB \mid a$
$A \rightarrow \varepsilon \mid S \mid B$
$B \rightarrow b$

Again removing $A \rightarrow \varepsilon$ the we get

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

3. Remove unit production:

$S' \rightarrow S$, $S \rightarrow S$, $A \rightarrow S$, $A \rightarrow B$

After removing $S \rightarrow S$

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

After removing : $S' \rightarrow S$

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

After removing : $A \rightarrow S$

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

After removing : $A \to B$

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

④ Now check for more than two variable on RHS.
$S' \to ASA$, $S \to ASA$, $A \to ASA$
Removing these
$S' \to ASA$
$\to AX \quad [X \to SA]$
Also
$S' \to ASA$
$\to AX \quad [X \to SA]$
$A \to ASA$
$\to AX \quad [X \to SA]$

Now we get
P : $S' \to AX \mid aB \mid a \mid SA \mid AS$
$S \to AX \mid aB \mid a \mid SA \mid AS$
$A \to AX \mid aB \mid a \mid SA \mid AS \mid b$
$B \to b$
$X \to AS$

⑤ Change production $S' \to aB$, $S \to aB$, $A \to aB$ as
$S' \to aB$
$\to YB$ where $Y \to a$
also,
$S \to YB$ and $A \to YB$

Assignment 2/21

Finally, we get
P : $S' \to AX \mid YB \mid a \mid SA \mid AS$
$S \to AX \mid YB \mid a \mid SA \mid AS$
$A \to AX \mid YB \mid a \mid SA \mid AS \mid b$
$B \to b$
$X \to AS$
$Y \to a$

Assignment

Q. Convert following CFG into CNF :
$S \to bA \mid aB$
$A \to bAA \mid as \mid a$
$B \to aBB \mid bS \mid b$

1. Here no S is in RHS, so no need to introduce start symbol
2. Here no empty and no unit production.
3. Now check for more than two Variable on RHS
$A \to bAA$, $B \to aBB$
Removing this,
$A \to bAA$
$\to bX \quad [X \to AA]$
also
$B \to aBB$
$\to aY \quad [Y \to BB]$
we get
$S \to bA \mid aB$
$A \to bX \mid as \mid a$
$B \to aY \mid bS \mid b$
$X \to AA$
$Y \to BB$

Convert into CNF

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

1. Since no S is in RHS so no need to introduce start symbol

Wrong

2. Removing null production : $A \to \varepsilon$
After removing $A \to \varepsilon$
P : $S \to AB \mid aB \mid B$
$A \to aab$
$B \to bbA \mid bb$

3. Here no unit production in RHS
4. Now check for more than two variable on RHS
Here more than two variable is not present in RHS
5. Change production $S \to aB$, $A \to aab$ $B \to bbA$ $B \to bb$ as

$S \to aB$
$\to XB$ $[X \to a]$

$A \to aab$
$\to XXbY$ $[X \to a]$ $[bY \to b]$

$B \to bbA$
$\to YYA$ $[Y \to bb]$
$B \to XY$

Ⓐ Change production : $S \to bA / S \to aB$, $A \to bX$, $A \to aS$
$B \to bS$ $B \to aY$ as

$S \to bA$
$\to PA$ $[P \to b]$

$S \to aB$
$\to QB$ $[Q \to a]$

Similarly
$A \to PX$, $A \to QS$, $B \to PS$, $B \to QY$

Finally Production :

$S \to PA \mid QB$
$A \to PX \mid QS \mid a$
$B \to PS \mid QY \mid b$
$X \to AA$
$Y \to BB$

$P \to b$
$Q \to a$

~~Finally~~ we get:

$S \to XB \mid AB \mid B$
$A \to XXbY$
$B \to YYA \mid Y$
$X \to a$
$Y \to b$

wrong

6. Again check for more than two variable on RHS $A \to XXY$ $B \to YYA$

$A \to XXY$
$\to X_1 Y \quad [X_1 \to XX]$

$B \to YYA$
$\to Y_1 A \quad [Y_1 \to YY]$

Now, finally we get,

P: $S \to XB \mid AB$
$A \to X_1 Y$
$B \to Y_1 A \mid Y \quad Y_1 \to YY$
$Y \to a \quad X_1 \to XX$
$Y \to b$

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

1. Since no S is in RHS. So no need to introduce start symbol

2. Removing null production: $A \to \varepsilon$
After removing: $A \to \varepsilon$
$S \to AB \mid aB \mid B$
$A \to aab$
$B \to bbA \mid bb$

3. Removing unit Production $S \to B$
P: $S \to AB \mid aB \mid bbA \mid bb$
$A \to aab$
$B \to bbA \mid bb$

4. Now there more than two variable is not present

5. Change the production $S \to aB$, $S \to bbA$, $S \to bb$ $A \to aab$, $B \to bbA$, $B \to bb$

$S \to aB$
$\to XB \quad [X \to a]$

$S \to bb$
$\to YY$

$A \to aab$
$\to XXY$

$S \to bbA$
$\to YYA \quad [Y \to b]$

$B \to YYA$

$B \to bb$
$\to YY$

P & S → AB | XB | YYA | YY X → a
A → XXY Y → b
B → YYA | YY

(6) Again eha check more than two variable on RHS

S → YYA, A → XXY, B → YYA as

S → YYA
→ Y1A [Y1 → YY]

A → XXY
→ X1Y [X1 → XX]

B → YYA
→ Y1A [Y1 → YY]

Now,

P & S → AB | XB | Y1A | YY
A → X1Y
B → Y1A | YY
X → a
Y → b
Y1 → YY
X1 → XX

Greibach Normal Form (GNF)

A CFG is in GNF if the production rule are of the following form:

A → b
A → bC1C2 ... Cn

where A, C1, C2, ... Cn are non-terminal and b is terminal

Steps to convert into (GNF)

Step1: Check CFG have any null or unit production if any them, then remove it
Step2: Check if CFG is in CNF, if not convert it
Step3: Change the non-terminal symbol into Aij is ascending order of i
Step4: Alter the rule so that non terminal are in ascending order such that if the production is of the form Ai → AjX, then i<j and should never be i≥j
Step5: Remove left recursion, if there are any

Eg: S → CA | BB
B → b | SB
C → a
A → a

Soln (1) No null or unit production
(2) Already in CNF

[Note: Left recursion : make problem]

③ Here we replace non-terminals with A_i,

S with A_1
C with A_2
A with A_3
B with A_4

we get

$A_1 \rightarrow A_2A_3 \mid A_4A_4$
$A_4 \rightarrow b \mid A_1A_4$
$A_2 \rightarrow b$
$A_3 \rightarrow a$

we have

$A_4 \rightarrow b \mid A_1A_4$ [$i > j$]
$A_4 \rightarrow b \mid A_2A_3A_4 \mid A_4A_4A_4$
[$A_1 \rightarrow A_2A_3 \mid A_4A_4$]
$A_4 \rightarrow b \mid bA_3A_4 \mid A_4A_4A_4$ [$A_2 \rightarrow b$]
$A_4 \rightarrow b \mid bA_3A_4 \mid A_4A_4A_4$

Here we get left recursion [$A_2 \rightarrow b$]

To remove left recursion introduce a new non-terminal as

$A_4 \rightarrow b \mid bA_3A_4 \mid A_4A_4A_4$
$Z \rightarrow A_4A_4Z \mid A_4A_4$

Now

$A_4 \rightarrow b \mid bA_3A_4 \mid bZ \mid bA_3A_4Z$

The grammar is

$A_1 \rightarrow A_2A_3 \mid A_4A_4$
$A_4 \rightarrow b \mid bA_3A_4 \mid bZ \mid bA_3A_4Z$
$A_2 \rightarrow b$
$A_3 \rightarrow a$
$Z \rightarrow A_4A_4Z \mid A_4A_4$

Now convert it into GNF as

$A_1 \rightarrow bA_3 \mid bA_4 \mid bA_3A_4A_4 \mid bZA_4 \mid bA_3A_4ZA_4$
$A_4 \rightarrow b \mid bA_3A_4 \mid bZ \mid bA_3A_4Z$
$A_2 \rightarrow b$
$A_3 \rightarrow a$
$Z \rightarrow bA_4Z \mid bA_3A_4A_4Z \mid bZA_4Z \mid bA_3A_4ZA_4Z$

This is the required GNF

Pushdown Automata (PDA)

↳ a PDA is a machine that is designed to implement context-free language
↳ CFL is accepted by PDA

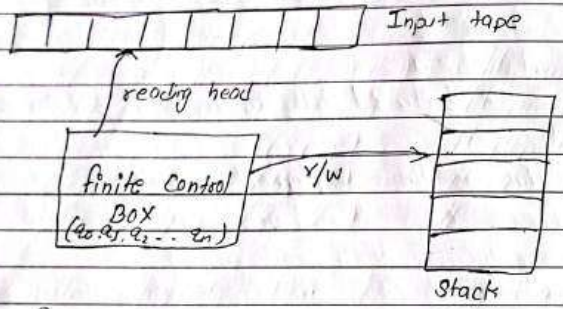


fig. pushdown Automata

↳ Similar to FA, but it has memory called pushdown Automata.
↳ element can be read-write on stack

Formally,
A PDA consist of seven tuples defined as
$M = \{Q, \Sigma, \Gamma, \delta, q_0, z_0, F\}$
where,
Q = set of finite state
Σ = i/p symbol or alphabet (that can be null)
(Note) Γ = symbol in stack
δ = transition function, $\delta(q, a, X)$
where, q is state in Q
a is i/p symbol
X is member of Γ (i.e. element in stack)

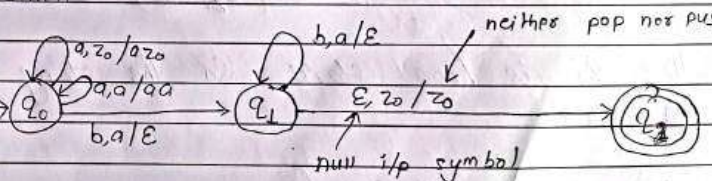
q_0 = initial state
z_0 = top of stack (member of Γ)
F = set of final states

* Instantaneous Description (ID)
Suppose, $A = \{Q, \Sigma, \Gamma, \delta, q_0, z_0, F\}$ is a PDA and instantaneous description is (q, x, α)

Eg:
$(q_0, a_1 a_2 \ldots a_n, z_0 z_1 \ldots z_n)$ is ID. This describe when PDA is in current state q_0 and input symbol to be processed is $a_1 a_2 \ldots a_n$ and PDS has $z_0 z_1 \ldots z_n$ where z_0 is top element, z_1 is the second top element and z_n is the lowest element

Q. Design a PDA for $L = \{a^n b^n \mid n \geq 1\}$

Soln Concept:
→ push every 'a' in the stack
→ Then, reading each 'b', remove each 'a' from the stack.



Thus PDA is
$A = \{Q, \Sigma, \Gamma, \delta, q_0, z_0, F\}$
where
$Q = \{q_0, q_1, q_f\}$

$\Sigma = \{a, b\}$
$\Gamma = \{a, z_0\}$
$q_0 = \{q_0\}$
z_0 = top of stack
$F = \{q_2\}$
δ can be written as

$\delta(q_0, a, z_0) = (q_0, a z_0)$ // push 'a'
$\delta(q_0, a, a) = (q_0, aa)$ // push 'a'
$\delta(q_0, b, a) = (q_1, \varepsilon)$ // pop 'a'
$\delta(q_1, b, a) = (q_1, \varepsilon)$ // pop a
$\delta(q_1, \varepsilon, z_0) = (q_2, z_0)$

Now we will simulate PDA for string "aaabbb" as

$(q_0, aaabbb, z_0) \vdash (q_0, aabbb, a z_0)$
$\vdash (q_0, abbb, aa z_0)$
$\vdash (q_0, bbb, aaa z_0)$
$\vdash (q_1, bb, aa z_0)$
$\vdash (q_1, b, a z_0)$
$\vdash (q_1, \varepsilon, z_0)$
$\vdash (q_2, z_0)$

Here q_2 is final state. So the string is accepted.

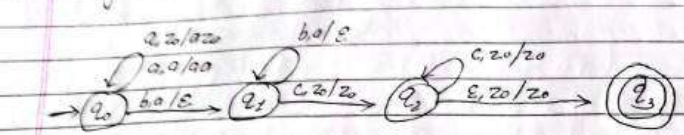
Assignment:> Design PDA for

① $L = \{a^n b^{2n} \mid n \geq 1\}$
② $L = \{a^n b^n c^m \mid n, m \geq 1\}$

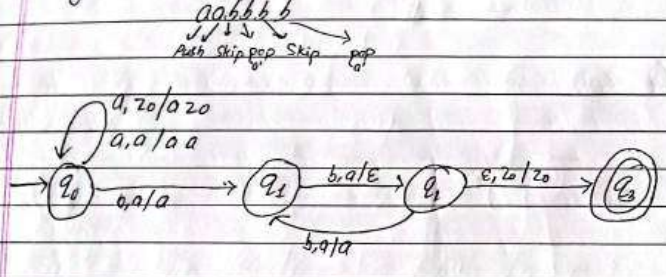
2/27

Assignment

Q Design PDA for $L = \{a^n b^n c^m / n,m \geq 1\}$



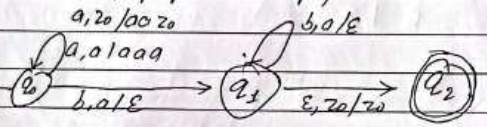
Q Design PDA for $L = \{a^n b^{2n} / n \geq 1\}$



or

For each a push two a

For each b, pop each pop

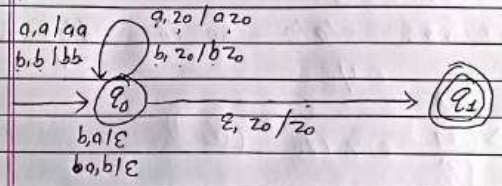


Q. Design PDA to accept $L = \{w / w \in (a+b)^* \text{ and } n_a(w) = n_b(w)\}$ where equal no of a's and b's

→ Initially when stack is empty, whatever we read push in stack

→ if we read 'a' and top of stack is 'b', pop b

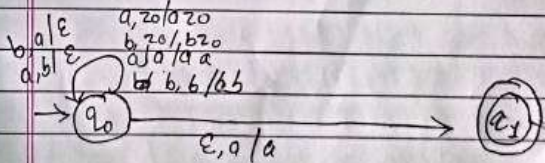
→ if we read 'b' and top of stack is 'a', pop a



Q. Design PDA to accept $L = \{w / w \in (a+b)^* \text{ and } n_a(w) > n_b(w)\}$ where a is greater than b

Concept

Initially whatever the symbol push into stack.
If we read a and top of stack b, pop b
If we read b and top of stack a, pop a
finally if we read ε, then stack should contain 'a' on top of stack.



2/28

Suppose string 'abbaa'

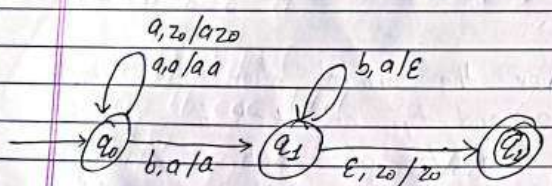
$(q_0, abbaa, z_0) \vdash (q_0, bbaa, az_0)$
$\vdash (q_0, baa, bz_0)$
$\vdash (q_0, aa, bbz_0)$
$\vdash (q_0, a, z_0)$
$\vdash (q_0, \varepsilon, az_0)$
$\vdash (q_1, \varepsilon, z_0)$

Here q_1 is the final state. So the string is accepted.

Q. Design a PDA for $L = \{a^n b b^n \mid n \geq 0\}$

Concept

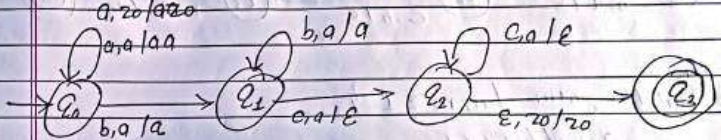
Skip first 'b'



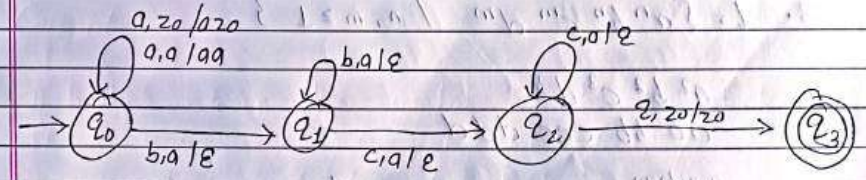
Q. Design PDA for $L = \{a^n b^m c^n \mid n, m \geq 1\}$

Concept

Skip 'b'



Q. Design PDA for $L = \{a^{m+n} b^m c^n \mid n, m \geq 1\}$



Assignment

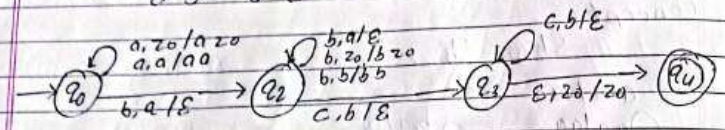
1. $L = \{a^n b^{m+n} c^m \mid n, m \geq 1\}$
2. $L = \{a^n b^m c^{n+m} \mid n, m \geq 1\}$
3. $L = \{a^n b^n c^m d^m \mid n, m \geq 1\}$
4. $L = \{a^n b^m c^m d^n \mid n, m \geq 1\}$
5. $L = \{a^i b^j c^k \mid i = j + k\}$

Assignment

1. $L = \{a^n b^{m+n} c^m \mid n, m \geq 1\}$

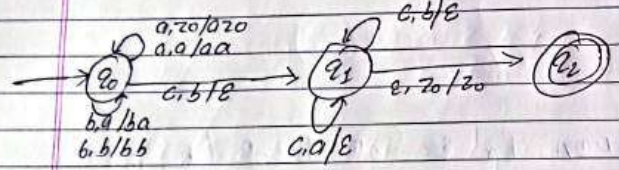
Let $a^n b^m \cdot b^n c^m$

$a^n b^n b^m c^m$



2. $L = \{a^n b^m c^{n+m} \mid n, m \geq 1\}$

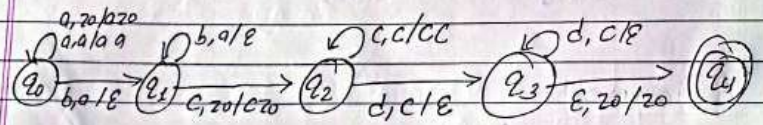
aa bb cccc



3. $L = \{a^n b^n c^m d^m \mid n, m \geq 1\}$

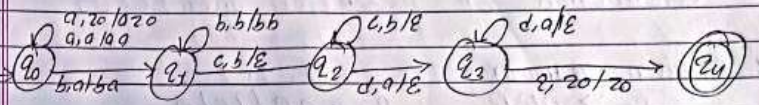
$a^2 b^2 c^3 d^3$

aa bb ccc ddd



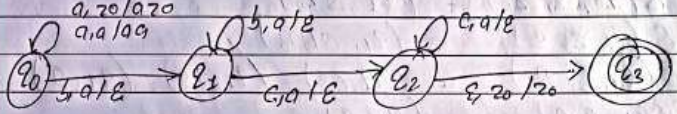
Assignment

4. $L = \{a^n b^m c^m d^n \mid n, m \geq 1\}$ aa bbb cc d



5. $L = \{a^i b^j c^k \mid i = j+k\}$

aaaa bb cc



Equivalance of CFG and PDA

Construction of PDA for given CFG

Eg: Given grammar
$S \to OBB$, $B \to OS/1S/0$
Find PDA.

we defined ~~as~~ PDA as

$P = \{Q, \Sigma, \Gamma, \delta, q_0, z_0, F\}$

Now define transition function δ as

$\delta(q_0, \varepsilon, z_0) = (q_1, Sz_0)$
$\delta(q_1, \varepsilon, S) = (q_1, OBB)$
$\delta(q_1, \varepsilon, B) = (q_1, OS), (q_1, 1S), (q_1, 0)$
$\delta(q_1, 0, 0) = (q_1, \varepsilon)$
$\delta(q_1, 1, 1) = (q_1, \varepsilon)$
$\delta(q_1, \varepsilon, z_0) = (q_f, z_0)$

Suppose for "01000"

$(q_0, \varepsilon 01100, z_0) \vdash (q_1, \varepsilon 01000, Sz_0)$
$\vdash (q_1, 01000, OBBz_0)$
$\vdash (q_1, 1000, BBz_0)$
$\vdash (q_1, 1000, 1SBz_0)$
$\vdash (q_1, \varepsilon 000, SBz_0)$
$\vdash (q_1, 000, OBBBz_0)$
$\vdash (q_1, \varepsilon 00, BBBz_0)$
$\vdash (q_1, 00, OBBz_0)$
$\vdash (q_1, \varepsilon 0, BBz_0)$
$\vdash (q_1, 0, Bz_0)$
$\vdash (q_1, \varepsilon, z_0)$
$\vdash (q_1, z_0)$

Q. Construct PDA for $S \to aSa/bSb/c$

Sol" We define PDA as:
$P = \{Q, \Sigma, \Gamma, \delta, q_0, z_0, F\}$
Now, define transition function as.

$\delta(q_0, \varepsilon, z_0) = (q_1, Sz_0)$
$\delta(q_1, \varepsilon, S) = (q_1, aSa), (q_1, bSb), (q_1, c)$
$\delta(q_1, a, a) = (q_1, \varepsilon)$
$\delta(q_1, b, b) = (q_1, \varepsilon)$
$\delta(q_1, c, c) = (q_1, \varepsilon)$
$\delta(q_1, \varepsilon, z_0) = (q_f, z_0)$

Now we simulate PDA for 'abbcbba'
$(q_0, \varepsilon abbcbba, z_0) \vdash (q_1, \varepsilon abbcbba, Sz_0)$
$\vdash (q_1, abbcbba, aSaz_0)$
$\vdash (q_1, bbcbba, Saz_0)$
$\vdash (q_1, bbcbba, bSbaz_0)$
$\vdash (q_1, bcbba, Sbaz_0)$
$\vdash (q_1, bcbba, bSbbaz_0)$
$\vdash (q_1, cbba, Sbbaz_0)$

$\vdash (q_0, cbba, cbba\, z_0)$
$\vdash (q_1, bba, bba\, z_0)$
$\vdash (q_1, ba, ba\, z_0)$
$\vdash (q_1, a, a\, z_0)$
$\vdash (q_1, \varepsilon, z_0)$
$\vdash (q_f, z_0)$

Construct of CFG from PDA

PDA = $\{Q, \Sigma, \Gamma, \delta, q_0, z_0, F\}$
CFG = $\{V, \Sigma, P, S\}$

So, $T = \{\varepsilon\}$, $S = \{s\}$

Suppose, $Q = \{q_0, q_1\}$, Γ = a of a PDA
Now, variable 'v' can be defined as
$[Q \Gamma Q]$

So, $V = \{S, [q_0 a q_1], [q_0 a q_0], [q_1 a q_0], [q_1 a q_1]\}$

Suppose we have transition function δ as

1. $\delta(q_0, a, z_0) = (q_0, \varepsilon)$ ⇒ POP
$[q_0\ z_0\ q_0] \rightarrow a$

2. $\delta(q_0, a, z_0) = (q_0, a\, z)$ ⇒ push
$[q_0 z_0] \rightarrow a [q_0 a] [z_0]$

3. $\delta(q_0, b, a) = (q_1, a)$ ⇒ skip
$[q_0 a] \rightarrow b [q_1 a]$

4. $\delta(q_0, a, z_0) = (q, a x x)$
$[q_0 z_0] \rightarrow a [q a] [x] [x]$

Q. Construct CFG from PDA

Given PDA $A = \{\{q_0, q_1\}, \{a, b\}, (a, z_0), \delta, q_0, z_0, \phi\}$, δ is given by
$\delta(q_0, a, z_0) = (q_0, a, z_0)$
$\delta(q_0, a, a) = (q_0, aa)$
$\delta(q_0, b, a) = (q_1, \varepsilon)$
$\delta(q_1, b, a) = (q_1, \varepsilon)$
$\delta(q_1, \varepsilon, z_0) = (q_1, \varepsilon)$

Sol^n (1) For $\delta(q_0, a, z_0) = (q_0, a, z_0)$
P_1: $[q_0 z_0 q_0] \rightarrow a [q_0 a q_0] [q_0 z_0 q_0]$
P_2: $[q_0 z_0 q_0] \rightarrow a [q_0 a q_1] [q_1 z_0 q_0]$
P_3: $[q_0 z_0 q_1] \rightarrow a [q_0 a q_0] [q_0 z_0 q_1]$
P_4: $[q_0 z_0 q_1] \rightarrow a [q_0 a q_1] [q_1 z_0 q_1]$

(2) For $\delta(q_0, a, a) = (q_0, aa)$
P_5: $[q_0 a q_0] \rightarrow a [q_0 a q_0] [q_0 a q_0]$
P_6: $[q_0 a q_0] \rightarrow a [q_0 a q_1] [q_1 a q_0]$
P_7: $[q_0 a q_1] \rightarrow a [q_0 a q_1] [q_0 a q_1]$
P_8: $[q_0 a q_1] \rightarrow a [q_0 a q_1] [q_1 a q_1]$

③ $\delta(q_0, b, a) = (q_1, \varepsilon)$
P9: $[q_0 a q_1] \to b$

④ $\delta(q_1, b, a) = (q_1, \varepsilon)$
P10: $[q_1 a q_2] \to b$

⑤ $\delta(q_1, \varepsilon, z_0) = (q_1, \varepsilon)$
P11: $[q_1 z_0 q_1] \to \varepsilon$

⑥ $S \to [q_0 z_0 q_0] / [q_0 z_0 q_1]$
Now we reduce useless symbol (non-reachable and non-generating symbol)

1. Here we remove P_2 and P_6 because $[q_1 z_0 q_0]$ and $[q_1 a q_0]$ are non-generating symbol.
2. Remove P_5 because generating itself
3. Remove P_1, P_3, P_7 because $[q_0 a q_0]$ does not generate any ~~smb~~ symbol
4. Remove $[q_0 z_0 q_0]$ from S because it is non generating symbol

So, we have CFG,
$G = \{V, \Sigma, P, S\}$
where
$V = \{S, [q_0 z_0 q_1], [q_0 a q_1], [q_1 z_0 q_1], [q_1 a q_1]\}$
$T = \Sigma = \{a, b\}$
$q = \{S\}$
P is given as
$S \to [q_0 z_0 q_1]$

P4: $[q_0 z_0 z_1] \to a [q_0 a q_1] [q_1 z_0 q_1]$
P8: $[q_0 a q_1] \to a [q_0 a q_1] [q_1 a q_1]$
P9: $[q_0 a q_1] \to b$
P10: $[q_1 a q_1] \to b$
P11: $[q_1 z_0 q_1] \to \varepsilon$

Closure properties of CFL

① Closed under union

Let L_1 and L_2 be two CFL generated by CFG $G_1 = \{V_1, \Sigma_1, R_1, S_1\}$ and $G_2 = \{V_2, \Sigma_2, R_2, S_2\}$

Now we construct a new language $L(G)$ using grammar $G = \{V, \Sigma, R, S\}$ such that it can accept $L(G_1) \cup L(G_2)$ where

$V = V_1 \cup V_2 \cup \{S\}$
$\Sigma = \Sigma_1 \cup \Sigma_2$
$R = R_1 \cup R_2 \cup \{S \to S_1 | S_2\}$
S = start state

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 we have $S \to S_1 | S_2$, So S will lead the string w

Hence $L(G) = L(G_1) \cup L(G_2)$

proved

(ii) Closed under concatenation

Let L_1 and L_2 two context free language (CFL) generated by $G_1 = \{V_1, \Sigma_1, R_1, S_1\}$ and $G_2 = \{V_2, \Sigma_2, R_2, S_2\}$ respectively.

Now, we construct a new language $L(G_1)$ using the grammar $G_1 = \{V, \Sigma, R, S\}$ such that it can accept $L(G_1) \cdot L(G_2)$ where

$V = V_1 \cup V_2 \cup \{S\}$

$\Sigma = \Sigma_1 \cup \Sigma_2$

$R = R_1 \cup R_2 \cup \{S \to S_1 S_2\}$

$S =$ start Symbol

Now let we choose a string $w_1 \in \Sigma_1$ and $w_2 \in \Sigma_2$ we know that $S_1 \xrightarrow{*} w_1$ and $S_2 \xrightarrow{*} w_2$ but in the above grammar G_1, S derives $S \to S_1 S_2$.

So, S will lead the concatination of the string w_1 and w_2 i.e. $w_1 \cdot w_2$ and the language will be $L_1 \cdot L_2$. Hence $L_1 \cdot L_2$ is also CFL.

(iii) closed under Kleene star

Let L_1 be the CFL generated by CFG $G_1 = \{V_1, \Sigma_1, R_1, S_1\}$. Now we construct a new language $L(G)$ using the grammar $G = \{V, \Sigma, R, S\}$ such that it can accept kleene star of the language L_1 i.e. L_1^*

where

$V = V_1 \cup \{S\}$

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

$R = R_1 \cup \{S \to \wedge, S \to S_1 S\}$

$S =$ Start Symbol

Here R follows all the properties of CFG as R_1 is the production of given CFG and $S \to \wedge$, $S \to S_1 S$ also fulfill the requirement so we say that G is a CFG that can generate CFL L_1^*

* Context free language are not closed under Intersection

We know that $L_1 = \{a^n b^n c^i \mid n \geq 1, i \geq 0\}$ and $L_2 = \{a^i b^n c^n \mid n \geq 1, i \geq 0\}$ are CFL.

Now L_1 intersection L_2 i.e. $L_1 \cap L_2 = \{a^n b^n c^n \mid n \geq 1\}$

So, $L = L_1 \cap L_2$

Here, $L = \{a^n b^n c^n\}$ is not CFL

Hence $L_1 \cap L_2$ is not context free

* Context free language are not closed under complement

Suppose L_1 and L_2 are CFL

Assume $\overline{L_1}$ and $\overline{L_2}$ are CFL.

Now using Demorgan's Law

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

We know that CFL is closed under union. and by

our assumption we have CFL closed under complement. But CFL are not closed under intersection.

Hence By contradiction CFL, is not closed under complement.

Decision property of CFL (Algorithm)

1. Deciding whether a CFL is finite

Construct a non-redundant CFG, G generating L-{ε}. We draw a directed graph whose vertices are variable in G. If $A \to BC$, is a production then there are directed edges from A to B and A to C. L is finited if and only if the directed graph has no cycle.

2. Deciding whether a CFL is empty

Given any CFL, L, there is a CFG, G to generate it. We can determine using the construction describe in the context of ~~elimation~~ elimination of useless symbols, whether the staat symbol is useless, if so, then $L(G) = \phi$; otherwise not.

2. Deciding wheather a string 'w' can be generated by some CFG (any string is member of CFL)

To determine wheather a particular string 'w' is in language of G, we will inspect all derivation trees of height and not at most length of string w.

If string w is found in any of the derivation treen, then $w \in L$

Pumping Lemma for CFL

Statement: Let L be a context free language and 'n' be the length of string or pumping length such that

1. Every $z \in L$ and $z \geq n$ can be written as $uvwxy$, for some u, v, w, x and y.
2. $|vx| \geq 1$.
3. $|vwx| \leq n$ (i.e if u and y are n, then $|vwx| = n$)
4. $uv^k wx^k y \in L$ for all $k \geq 0$ (i.e. generate infinite numbes of string by setting any value for k)

Proof:

To prove the theorem, we consider a CFG whose production are given as

$S \rightarrow AB \mid a$

$A \rightarrow aB \mid a$

$B \rightarrow bA \mid b$

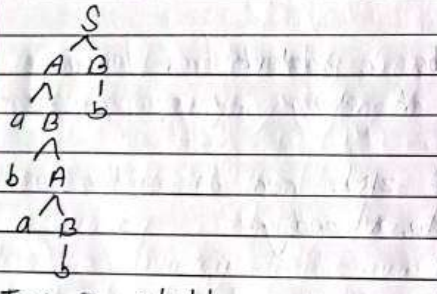
Note: should be derived from grammar

Now let any string $z = ababb$ such that $z \in L$

Thus we decompose z as

$u = a$, $v = ba$, $w = b$, $x = \wedge$, $y = b$

Now, draw a parse tree for given string



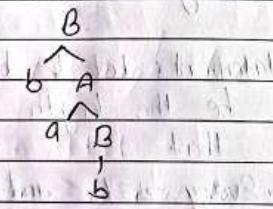
$T_0 : z = ababb$

$T_1 : z_1 = b$

As z and z_1 are derivation of T and a proper subtree T_1 of T, so we can write

$z = uvwxy$

$\therefore z = u z_1 y$



$T_1 : z_1 = bab$

As z_1 and z_2 are derivation of T_1 and proper subtree T_2 of T_1 we can write $z_1 = vz_2x$ (i.e. vwx)

Also, we have $|vwx| \geq |w|$

So, we have $|vx| \geq 1$

Thus we have $z = uvwxy$ with $|vwx| \leq n$ and $|vx| \geq 1$ where n is length of string z.

As T is a S-tree (starting symbol) and T_1 and T_2 are B-Tree (i.e. starting from symbol 'B').

we get

$S \xrightarrow{*} uBy$

$B \xrightarrow{*} vBx$

$B \rightarrow w$

Now, $S \rightarrow uBy$

$\rightarrow uvBxy \quad [B \rightarrow vBx]$

$\rightarrow uvwxy \quad [B \rightarrow w]$

Thus $S \rightarrow uv^1wx^1y$, where $k = 1$

Similarly,

$S \rightarrow uBy$

$\rightarrow uvBxy \quad [B \rightarrow vBx]$

$\rightarrow uvBxxy \quad [B \rightarrow vBx]$

$\rightarrow uv^2Bx^2y \quad [B \rightarrow w]$

and so on.

Thus $uv^k wx^k y \in L$ for $k \geq 1$

Hence proved.

* Procedure to prove given language is not CFL.

Step 1: Assume L is context free. let n be pumping length.

Step 2: Choose $z \in L$ so that $|z| \geq n$ write z = uvwxy using pumping lemma

Step 3: Find a suitable k show that $uv^k wx^k y \notin L$ This is a contradiction so, L is not CFL.

Q. Show that $L = \{a^n b^n c^n / n \geq 1\}$ is not CFL.

Sol^n

Step 1: Assume L is context free Language Let n be pumping length.

Step 2: let $z = a^n b^n c^n$ Then $|z| = 3n \geq n$. Write $z = uvwxy$, where $|vx| \geq 1$

Step 3: Here $z = uvwxy$. So, $uvwxy = a^n b^n c^n$
As $1 \leq |vx| \leq n$, v or x cannot contain all three symbol

Suppose n=4, so $z = a^4 b^4 c^4$

Case I:
v and x, each contain only one type of symbol

a | aa | abbbbc | cc | c
u | v | w | x | y

We have,
$uv^k wx^k y$

For k=2
$uv^2 wx^2 y$
= aaaaa abbbb cccccc
= $a^6 b^4 c^6$

Here no. of a's b's and c's are not equal
So, $uv^k wx^k y \notin L$ for k=2

Case II:
Either v or x can have more than one symbol

aa | aabb | bbc | ccc
u | v | w | x y

We have $uv^k wx^k y$
So for k=2
$uv^2 wx^2 y = uvvwxxy$
= aaaabbaabbbbcccccc

Since, it is not in pattern. So,
$uv^k wx^k y \notin L$ for k=2

Here in both cases we get contradiction. So given Language $L = \{a^n b^n c^n / n \geq 1\}$ is not C

Assignment

Q. Show that $L = \{ww \mid w \in (0,1)^*\}$ is not CFL

→

1. Assume L is CFL, n is pumping length
2. let $z = 0^n 1^n 0^n 1^n$ then $|z| = 4n \geq n$
   were $z = uvwxy$ where $|vx| \geq 1$
3. Here $z = uvwxy$
   so, $uvwxy = 0^n 1^n 0^n 1^n$
   as, $1 \leq |vx| \leq n$, v or x cannot contain all 4 symbol
   Suppose $n = 4$
   Case 1: v and x contain only one type of symbol
   So $z = \underbrace{00}_{u}\ \underbrace{00}_{v}\ \underbrace{11}_{w}\ \underbrace{11}_{x}\ \underbrace{0000}_{y}\ 1111$
   we have $uv^k wx^k y$
   For $k=2$, uv^2wx^2y
   $= 00\ 0000\ 11\ 11\ 0000\ 1111$
   $= 0^6 1^4 0^6 1^4$

Here no. of 1's and 0's are not equal so, $uv^k wx^k y$ for $k \neq 1 \notin L$

Case 2: Either v or x can have 2 symbol
So $\underbrace{00}_{u}\ \underbrace{0011}_{v}\ \underbrace{1100}_{w}\ \underbrace{00}_{x}\ \underbrace{1111}_{y}$

for $k=2$
i.e $z = 00\ 0011\ 0011\ 1100\ 0000\ 1111$
$= 0^4 1^2 0^2 1^4 0^6 1^4$
Here not in pattern so, it $\notin L$

The both two condition get contradiction. So given L is not in CFL.

Q. Prove that $L = \{a^n b^n c^n \mid n \geq 0$ if not CFL

1. Consider L is CFL also let $z = a^n b^n c^n$, $z \in L$

According to pumping lemma of CFL
z decompose into u, v, w, x and y as follows:
$u = a^r$
$v = a^s$ $(s \geq 0)$
$w = a^{p-(r+s)}$
$x = b^t$ $(t > 0)$
$y = b^{(p-t)} c^p$

Now using pumping lemma, $z = uv^k wx^k y$, $k \geq 0$
We have.

let $k = 2$
so, $z = uv^2 wx^2 y$
$= a^r (a^s)^2 a^{p-(r+s)} (b^t)^2 b^{(p-t)} c^p$
$= a^r a^{2s} a^{p-r+s)} b^{2t} b^{(p-t)} c^p$
$= a^{(p+s)} b^{(p+t)} c^p$
$\neq a^p b^p c^p$

Hence our assumption $z \in L$, contradicts with our result

so L is not CFL.

Turing Machine

Turing Machine
→ mathematical method of general purpose computer
→ TM can solve any problem that can be solved by any computer machine
→ developed / designed by 'Alan Turing'

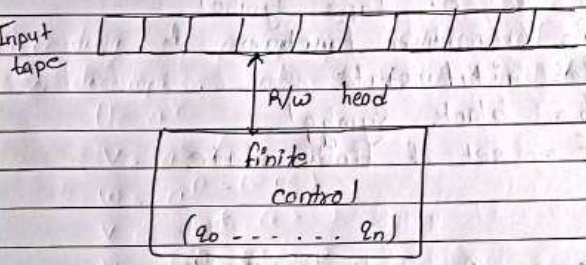


fig: Turing Machine (TM)

→ TM has 1x tape which is divided into number of cell. Each cell can store only one number of System.
→ In one move, the ~~TM~~ machine read the present symbol under the R/w head ~~and~~ on the tape and the present state of finite control to determine
1) A new symbol to be written on the tape under the R/w head
2) A motion of R/w head along, the head either the head move once at the left ~~and~~ or once at right
3) The next step of automata
4) Weather to halt or not.

Mathematically, TM has 7 tuples:

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

where,

Q = set of finite states
Σ = i/p symbol
Γ = set of tape symbol
δ = transition function
q_0 = inital state
b = blank Symbol
F = set of final state

Instantaneous Description (ID)

An ID of Turing Machine M is a string $\alpha\beta\gamma$, here β is present state of M, the entire input string is splite as $\alpha\gamma$. The first of symbol of γ is the current symbol under the R/W head and γ as all the symbol of input string and the input string α is the substring of input string formed by all the symbol to the left of present state.

L = Left
R = Right

Example:

	Tape Symbol (blank symbol)		
Present state	b	0	1
$\rightarrow q_1$	$1Lq_2$	$0Rq_1$	–
q_2	bRq_3	$0Lq_2$	$1Lq_2$
q_3	–	bRq_4	bRq_5
q_4	$0Rq_5$	$0Rq_4$	$1Rq_4$
(q_5)	$0Lq_2$	–	–

Consider above TM, draw computation of input string "00b"

$\Rightarrow (q_1, 00b) \vdash 0q_1 0b \vdash 00q_1 b \vdash 0q_2 01$

$\vdash q_2 001 \vdash q_2 b001 \vdash bq_3 001$

$\vdash bbq_4 01 \vdash bb0q_4 1 \vdash bb01q_4 b$

$\vdash bb010q_5 b \vdash bb01q_2 00$

$\vdash bb0q_2 100 \vdash bbq_2 0100$

$\vdash bbq_2 b0100 \vdash bbbq_3 0100$

$\vdash bbbq_4 100 \vdash bbb1q_4 00 \vdash bbb10q_4 0$

$\vdash bbb100q_4 b \vdash bbb1000q_5 b$

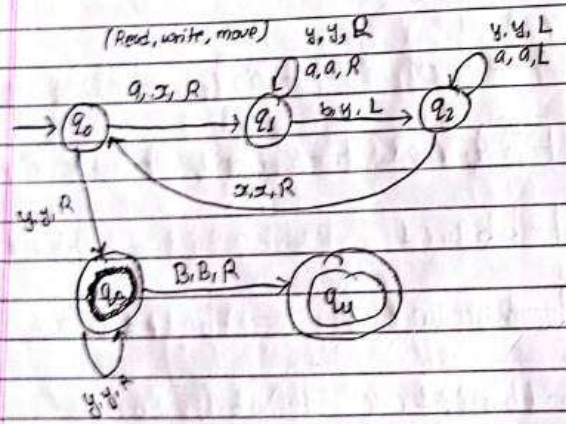
$\vdash bbb1000q_2 00 \vdash bbb10q_2 000 \vdash bbbq_2 10000$

$\vdash bbb q_2 10000 \vdash bb q_2 b 10000 \vdash bbb q_3 10000$
$\vdash bbbb q_5 0000$

Since q_5 is final step. So accepted.

Design a TM

Q. Design a TM that accepts $L = \{a^n b^n \mid n \geq 1\}$



Hence, required TM, M is

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

where

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

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

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

$q_0 = \{q_0\}$

$(q_0, a) \longrightarrow (q_1, x, R)$ — change state; write; move

$b = \{B\}$

$F = \{q_4\}$

and δ is

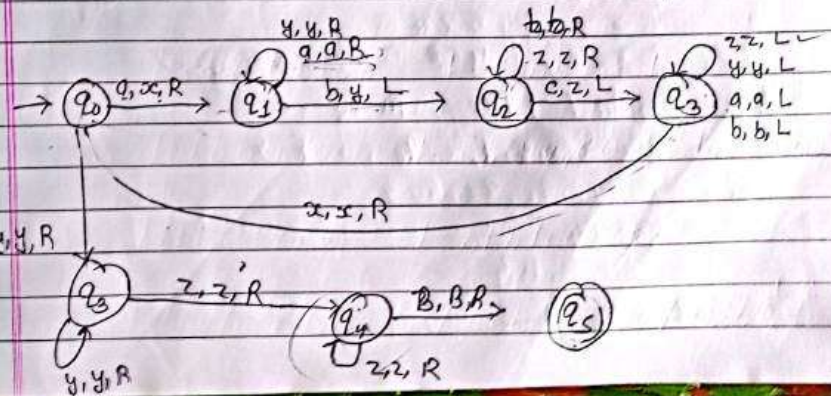
State \ Symbol	Tape Symbol a	b	B	x	y
$\rightarrow q_0$	xRq_1	–	–	–	yRq_3
q_1	aRq_1	yLq_2	–	–	yRq_1
q_2	aLq_2	–	–	xRq_0	yLq_2
q_3	–	–	BRq_3	–	yRq_3
q_4^*	–	–	–	–	–

Now, we simulate TM for string 'aabb':

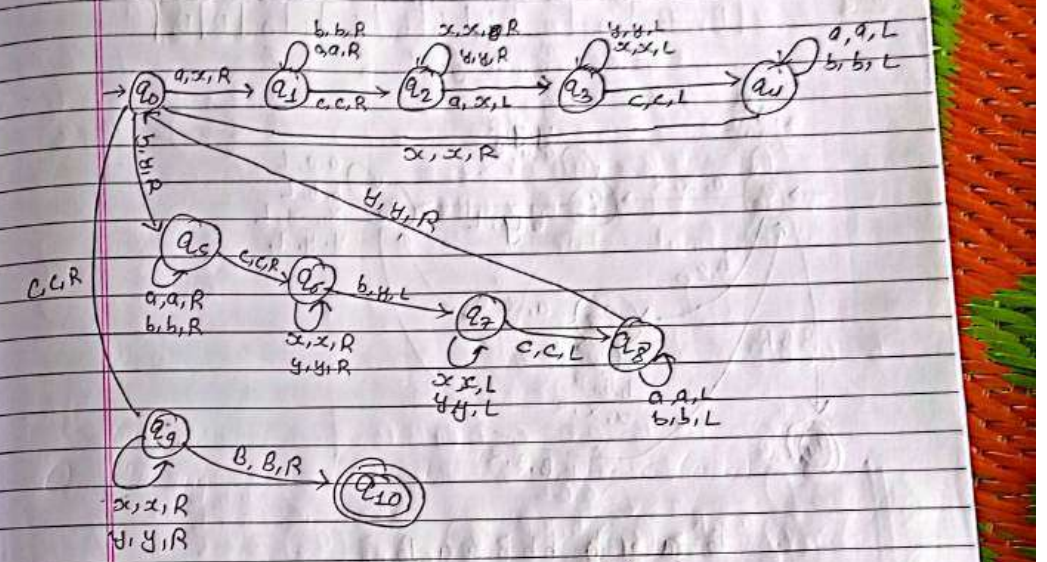
$(q_0, aabb) \vdash xq_1abb \vdash xaq_1bb \vdash xq_2ayb$
$\vdash q_2xayb \vdash xq_0ayb \vdash xxq_1yb$
$\vdash xxyq_1b \vdash xxq_2yy \vdash xq_2xyy$
$\vdash xxq_0yy \vdash xxyq_3y \vdash xxyyq_3B$
$\vdash xxyyBq_4B$

Q. Design a TM that accept $L = \{a^n b^n c^n \mid n \geq 1\}$

BBaabbccBB



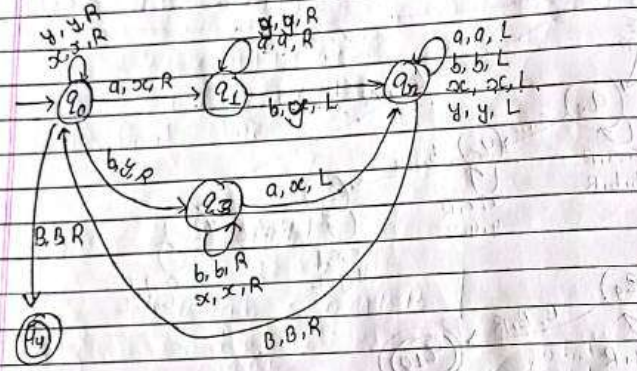
Q. Design TM for $L = \{wcw \mid w \in (a,b)^*\}$



Assignment
3/9

Q. TM for equal number of a's & b's

String: B B a a b b b b a a B B



B a a b b b b a a B
B x a y b b b a a B
B x x y y b b a a B
B x x y y y b x a B
B x x y y y b x a B

Here Required TM, M is

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

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

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

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

$q_0 = q_0$

Assignment
3/9

$b = \{B\}$

$F = \{q_4\}$

δ is

Symbol	a	b	B	x	y
→ q_0	xRq_1	yRq_3	BRq_4	xRq_0	yRq_0
q_1	aRq_1	yLq_2	–	–	yRq_1
q_2	aLq_2	bLq_2	BRq_0	xLq_2	yLq_2
q_3	xLq_2	bRq_3	–	xRq_3	–
q_4	–	–	–	–	–

String 'aabbbbaa'

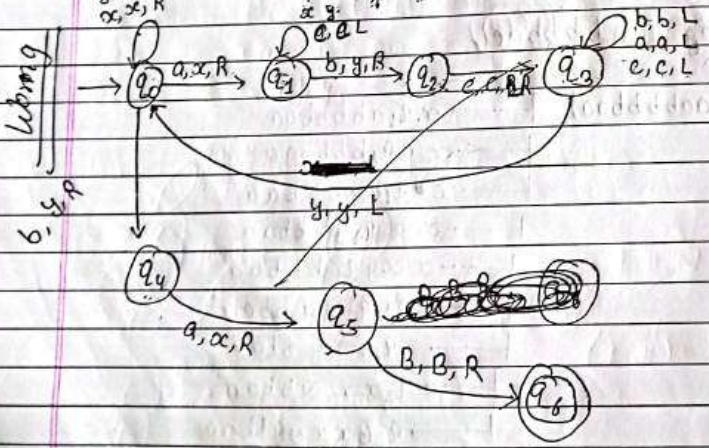
$(q_0, aabbbbaa) \vdash x q_1 abbbbaa$
$\vdash x a q_1 bbbbaa$
$\vdash x x q_2 a y bbbaa$
$\vdash x x q_1 y bbbaa$
$\vdash x x y q_1 bbbaa$
$\vdash x x q_2 y y bbaa$
$\vdash x q_2 x y y bbaa$
$\vdash q_2 x x y y bbaa$
$\vdash q_2 B x x y y bbaa$
$\vdash B x q_0 x y y bbaa$
$\vdash B x x q_0 y y bbaa$
$\vdash B x x y q_0 y bbaa$
$\vdash B x x y y q_0 bbaa$
$\vdash B x x y y y q_3 baa$
$\vdash B x x y y y b q_3 aa$

$\vdash Bxxyy\,q_2\,bxa \vdash Bxxy\,q_2\,y\,bxa$
$\vdash Bxx\,q_2\,yy\,bxa \vdash Bx\,q_2\,x\,yy\,bxa$
. . . . . .
$\vdash Bxxyyy\,q_3\,x\,xB$
$\vdash Bq_0\,xxyyyxxB$
. . . .
$\vdash Bxxyyyxxq_0B$
$\vdash BxxyyyxxBq_4$
is the final state...

b. $wcw^R \vdash w \in (a,b)^*$

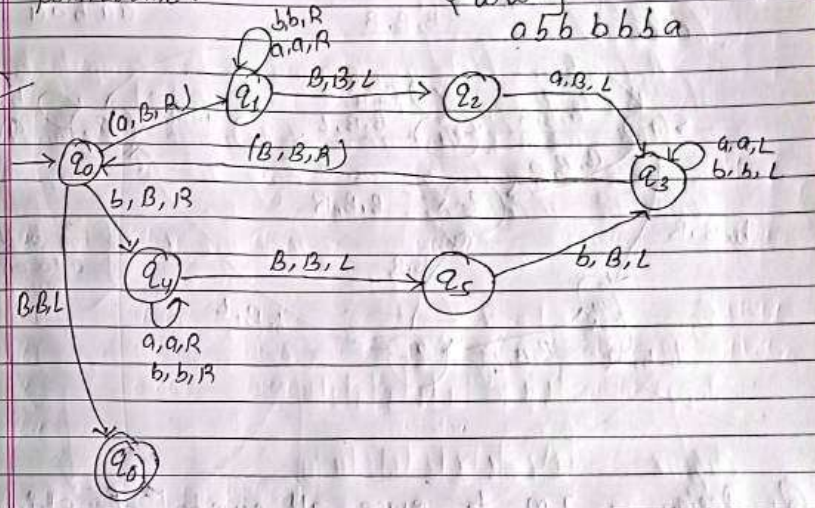
$w = ab$

B B a b c b a B B

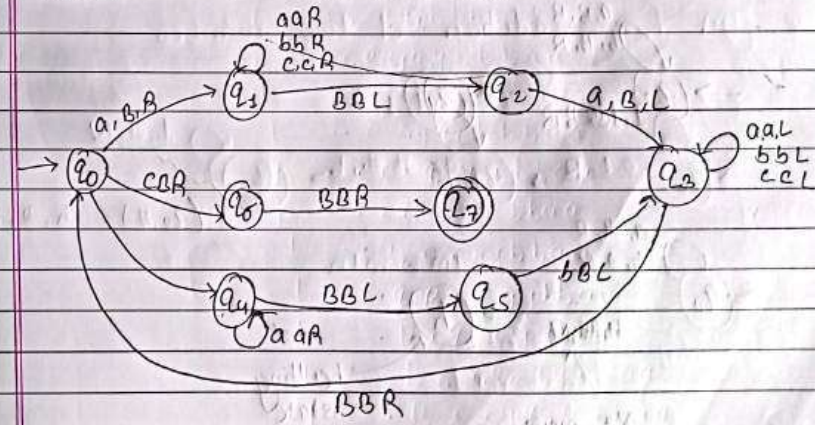


Q. Design TM to check string $w \in (a,b)^*$ is palindrome. $\{ww^R\}$

a b b b b a

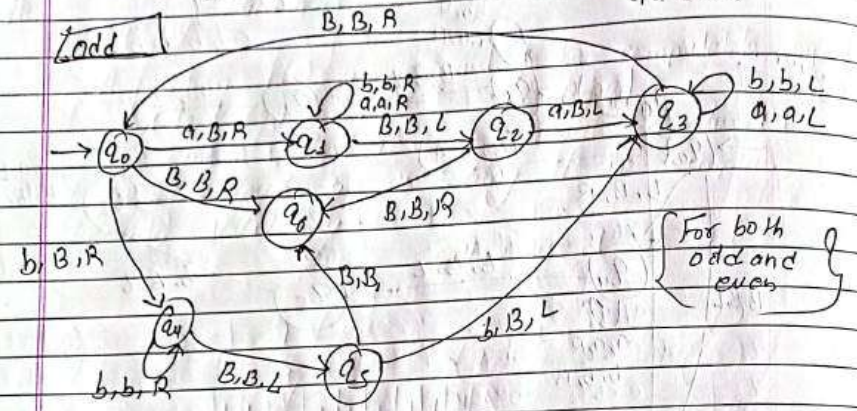


(b) $wcw^R \vdash w \in (a,b)^*$



for checking string $w \in (a,b)^*$ is palindrome

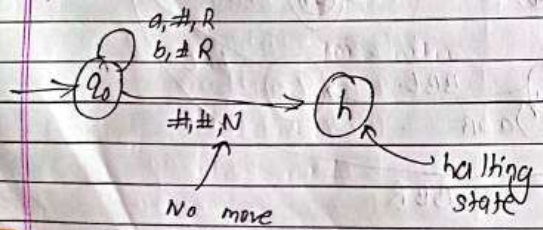
aabaa



Q. Design a TM to erase all symbol of string over $\Sigma = \{a,b\}$.

Or

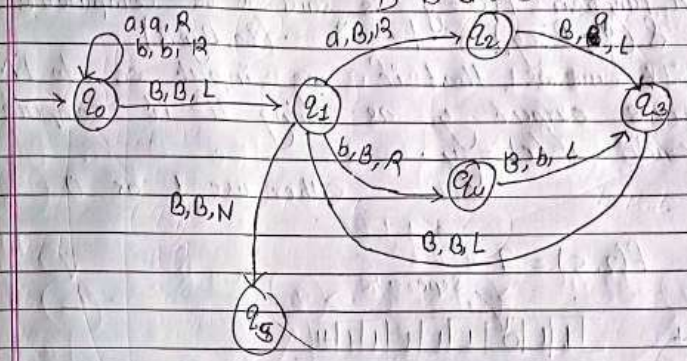
$(q_0, \#) = (h, \#, N)$



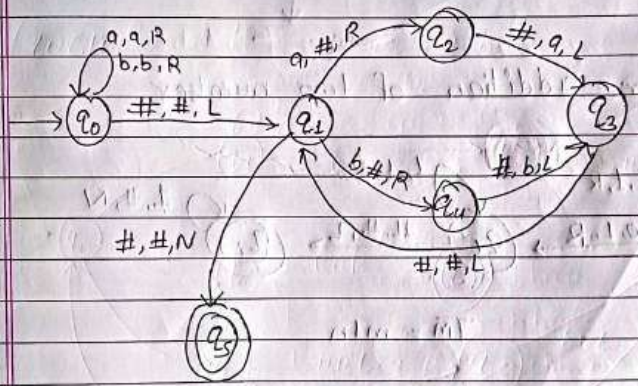
Q. Design a TM as a right shift machine that transforms #w# to #w#w# with $\Sigma = \{a,b\}$

B a bb a B

B B B b a B



#	a	b	a	b	b	a	#



Computing Function

A function $f(x) = y$ is said to be computable by TM if $\delta(q_0, Bxy) \xrightarrow{*} (q_0, ByB)$.

It means that if we input x to TM, it gives output y as string if it compute function $f(x) = y$.

either use 1 or 0.

Eg: $f(2,5) = ?$

Input: 1 1 0 1 1 1 1 1 # (0 separate)

Output: 1 1 0 1 1 1 1 1 # #

Imp

* TM for addition of two number

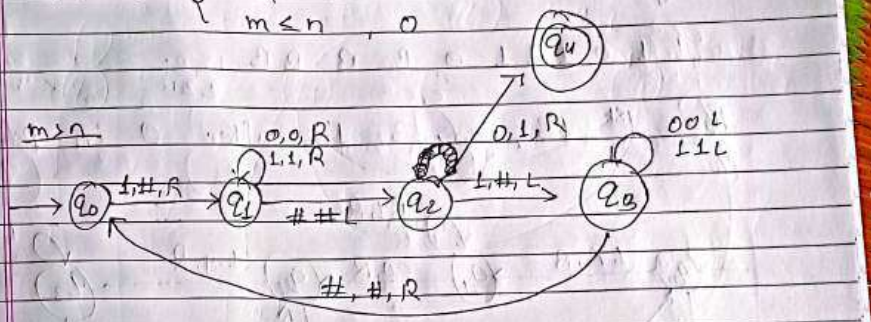
TM = m+n

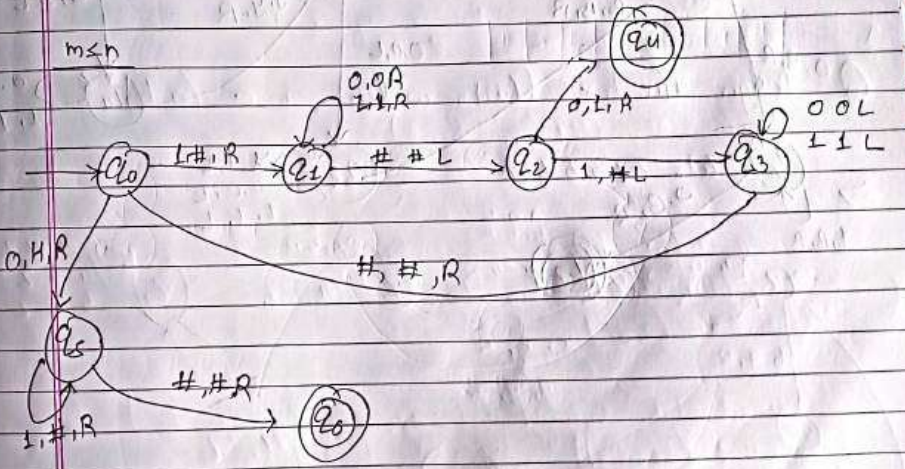
Similarly, $f(n) = n+2$

$f(n) = n+1$

② TM for subtraction of two number

$$f(m,n) = \begin{cases} m-n & m \geq n \\ 0 & m < n \end{cases}$$





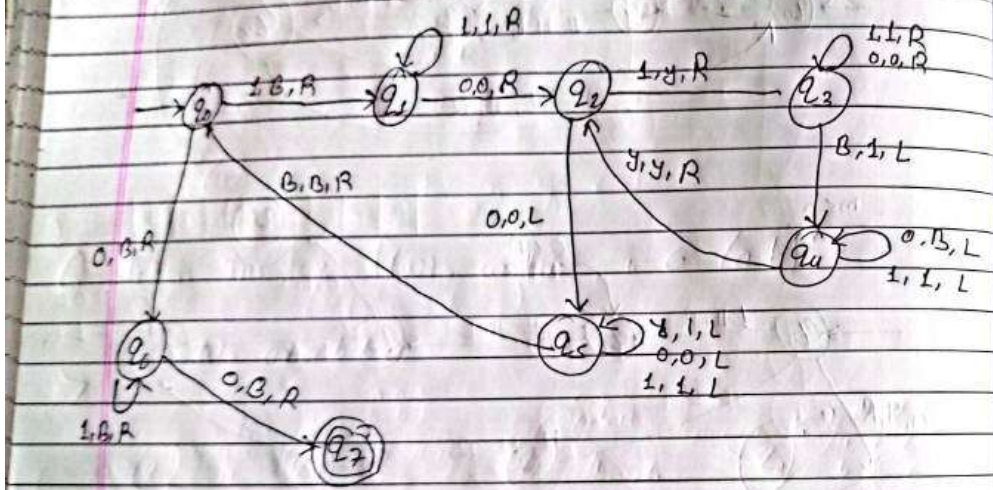
* Turing Machine for multiplication

$f(x,y) = x \times y$

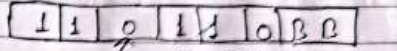
Eg: 2×3

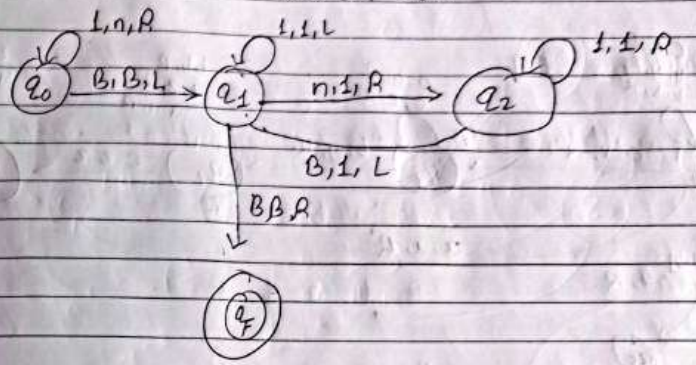
1 1 0 1 1 1 0 B B B B ...

For 1st 1's, Copy 3 1's after 0 in position

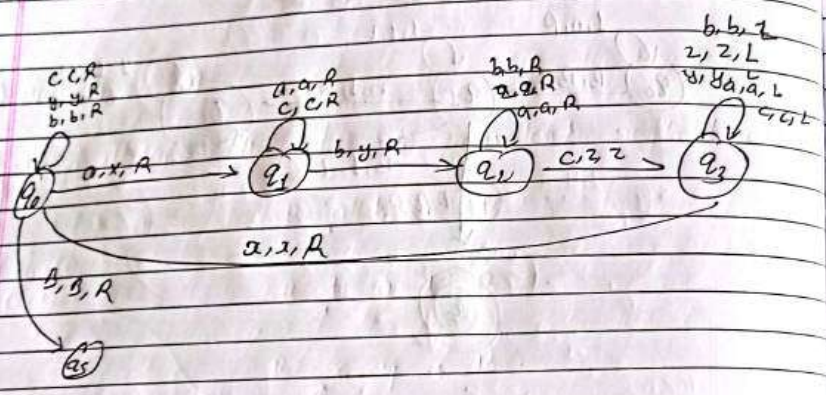


* TM for $f(n) = 2n$





Q Design a TM for equal no. of a's, b's and c's.

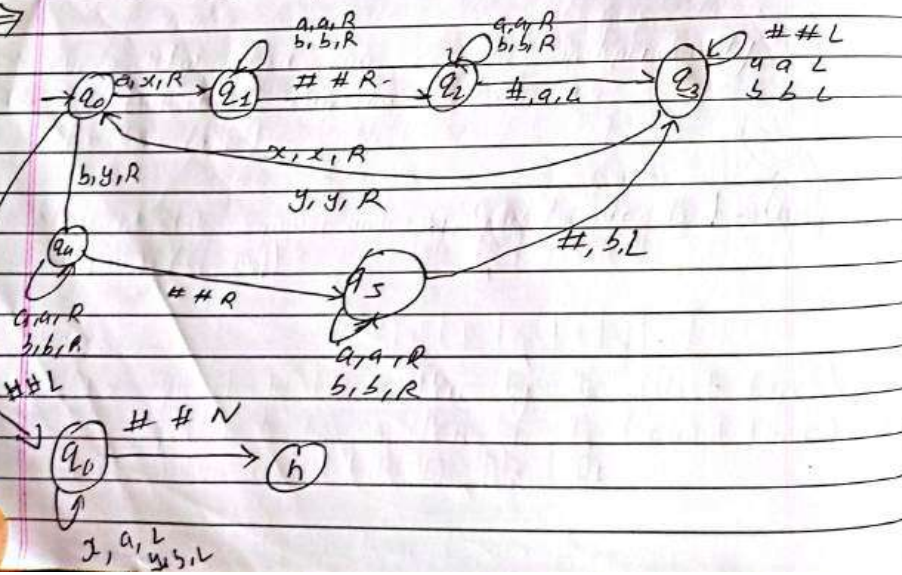


Ex:

Q. Design TM for $f(w) = w\#w$, $w \in (a,b)^*$

$w = ababba$

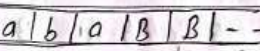
$\Rightarrow$



Extension (Variation) of Turing Machine

1) Infinite on two side of input tapes
2) Multi head on single tape
3) Multi tape TM
4) Non-deterministic TM

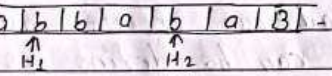
1) Infinite on two side of input tape



→ Left and Right side have infinite blank symbol.
→ In standard TM, blank symbol is define in TD
→ In this TM, we do not include blank symbol in TD
→ Eg: $a^n b^n$

2) Multi head on Single tape:

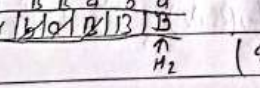
Suppose two head are define in a single tape, H_1, H_2. These two heads can read/write/move independently.



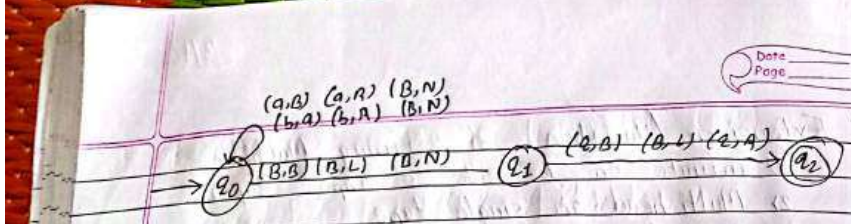
The transition can be defined as δ(current state, Symbol Under H_1, Symbol under H_2) = (new state, (S_1, m_1) (S_2, m_2))

Here, S_1 ⇒ symbol to be written under H_1
S_2 ⇒ Symbol to be written under H_2
m_1 = move i.e. (L, R, N) for H_1
m_2 = move i.e. (L, R, N) for H_2

Eg:



$(q_1, a, B) = (q_2 (B, L), (q_1$



3) Multitape TM

→ every multitape TM has an equivalent single tape TM

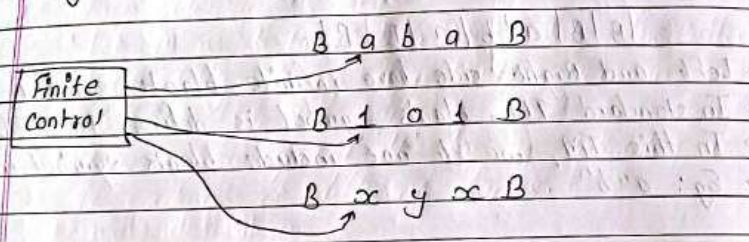


fig: Multitape TM
(k tape, k=3)

→ every R/w head can operate independently
→ The transition can be define as
(for two tape)

δ(current state, Symbol under I_1, Symbol under I_2) = (new state, (S_1, M_1) (S_2, M_2))

Here,
S_1 = Symbol to be written by I_1 (first tape)
S_2 = Symbol to be written by I_2 (second tape)
M_1 = move i.e. (L, R, N) for I_1

M_2 = move i.e. (L, R, N) for I_2

* Multitape TM

Theorem: For any k-tape (multitape) TM, there is an equivalent single tape TM

Let $k \geq 1$ be any k-tape TM. The key idea to convert multitape TM is to show the simulation of multitape TM 'M' in single tape TM 'S'.

- Let M' as k-tape, then S simulate the effect of k-tape by storing their information on single tape.
- It uses new symbol '#' as a delimiter to seperate the contains of different tapes.
- In Addition, to the contain of those tapes, S must be track of the location of the head by writing a tape symbol with a dot (•). To make the place where the head on that tape would be.

Following figure illustrate how 1 tape can be used to represent 3-tape.

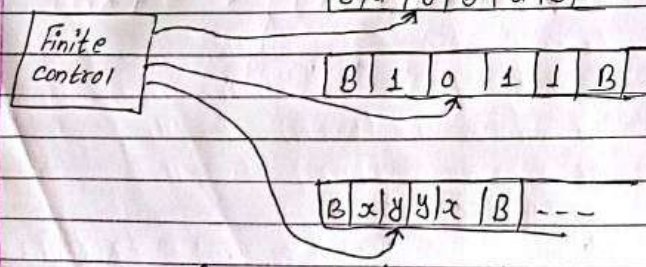


fig: K-tape (where K=3)

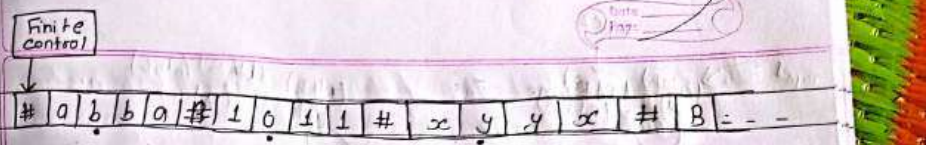


fig: representing 3 tapes with one tapes

To simulate a transcition from state q, we must scan our tape to see which symbol 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.

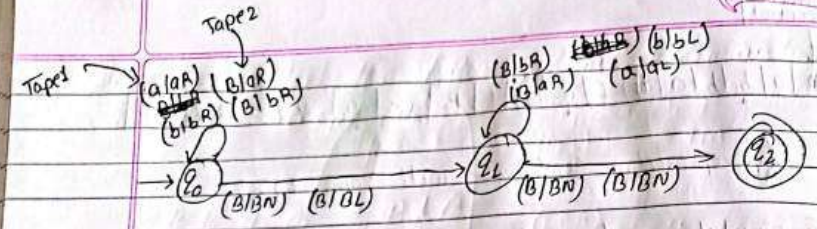
Whenever one tape, whenever one head moves of the right ends; we must shift our tape so we can insert a blank symbol.

Construct a 2 tape TM to convert an input W into ww^R, $w \in (a,b)^*$.

Initially tape 1 contains W and tape 2 is blank.
Let W be 'abb'

Tape 1: B a b b B - -
H_1 (H = Head)

Tape 2: B B B B B B - - -
H_2



* Non-Deterministic TM

A TM is non-deterministic if there are multiple possibilities for the current input and present state.

Formal definition

The non-deterministics is combined of 6 Tuples $(Q, \Sigma, \Gamma, \delta, q_0, F)$ where

$Q \rightarrow$ set of state
$\Sigma \rightarrow$ set of i/p state
$\Gamma \rightarrow$ set of tape symbol
$q_0 \rightarrow$ initial state
$F \rightarrow$ set of final state

δ is

$$Q \times \Gamma \longrightarrow P(\Gamma \times R, L, N) \times Q$$

(Power set)

In this transition function the left side of transition is enclosed in power set, which means that a state can move to more than one state on getting a particular input

All ~~possibles~~ possible transition sequence of non-deterministic TM on a given input string can be represented by computation tree.

For eg:

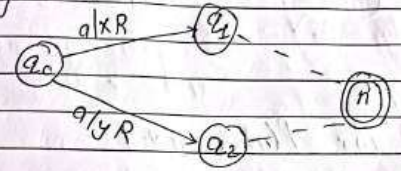
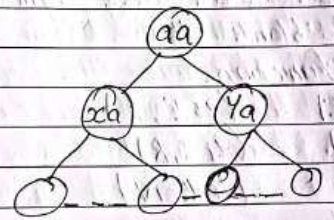


fig: non deterministic TM

We draw computation ~~for~~ tree for string 'aa'



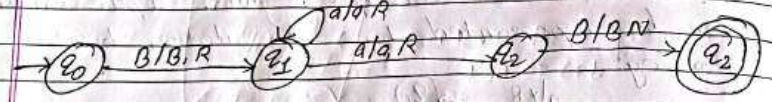
The outcome of NTM is

1) If any branch accept the input string, then NTM is ~~or~~ also accept
2) If all branches halt or reject the input, then NTM also rejected

3/18

Eg: Consider the following non-deterministic TM that accept a^+.

B B a a a B B



Design a TM for $f(n) = n+1$

$w \rightarrow$ i/p

To represent n we use symbol I.

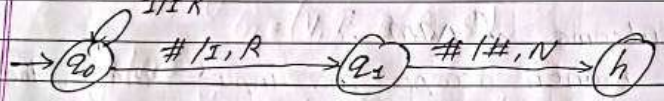
for blank #

Transition function.

$\delta(q_0, I) = (q_0, I, R)$

$\delta(q_0, \#) = (q_1, I, R)$

$\delta(q_1, \#) = (h, \# N)$



Design a TM for $f(n) = n+2$

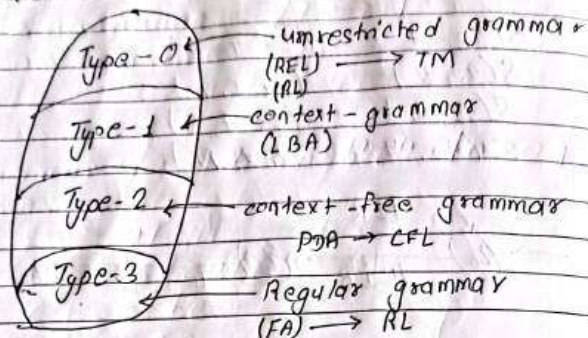
$\delta(q_0, I) = (q_0, I, R)$

$\delta(q_0, \#) = (q_1, I, R)$

$\delta(q_1, \#) = (q_2, I, R)$

$\delta(q_2, \#) = (h, \# N)$

Unrestricted Grammar



↳ phase structure grammar

Unrestricted Grammar is the most general in the Chomsky Hierarchy. It is called unrestricted bcz no other restriction is made on this ~~ae~~ except each of their LHS being non-empty. The left Hand Side of the rule can contain terminal and non terminal but the condition is at least one of them must be non-terminal.

A TM can simulate ~~no~~ unrestricted grammars and unrestricted grammar can simulate TM. Formally,

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

where

V = set of non-terminal symbol
T = set of terminal symbol
S = Starting Symbol, $S \in V$
P = Set of rule or production

Each production is of the form, $\alpha \to \beta$

where α is $(V \cup T)^* V (V \cup T)^*$
β is $(V \cup T)^*$

The language generated by a unrestricted grammar is called Recursively Enumerable Language (REL).
The REL is accepted by TM.

Eg: Given grammar for $L = \{a^{2^n} \mid n \geq 0\}$

Production s:

$S \to TaU$
$U \to \varepsilon \mid AU$
$aA \to Aaa$
$TA \to T$
$T \to \varepsilon$

Derive a^8.

$S \longrightarrow TaU$
$\longrightarrow TaAU \quad [U \to AU]$
$\longrightarrow TaAAU \quad [U \to AU]$
$\longrightarrow TaAAAU \quad [U \to AU]$
$\longrightarrow TaAAAC \quad [U \to \varepsilon]$
$\longrightarrow TAaaAA \quad [aA \to Aaa]$
$\longrightarrow TAaAaaA \quad [aA \to Aaa]$
$\longrightarrow TAAaaaaA$
$\longrightarrow TAaaaAaa \quad [aA \to Aaa]$
$\longrightarrow TaaAaaaa \quad [aA \to Aaa, TA \to T]$
$\longrightarrow TaAaaaaaa$
$\longrightarrow TAaaaaaaaa$

→ Taaaaaaaa [TA → T]
→ εaaaaaaaa [T → ε]
→ aaaaaaaa

Q. Given grammar for $L = \{a^n b^n c^n \mid n \geq 0\}$
production:

$S \rightarrow UT$
$U \rightarrow \varepsilon / aUbC$
$Cb \rightarrow bC$
$CT \rightarrow TC$
$T \rightarrow \varepsilon$

Derive $a^3 b^3 c^3$.

$S \rightarrow UT$
→ aUbCT [U → aUbC]
→ aaUbCbCT [U → aUbC]
→ aaaUbCbCbCT [U → aUbC]
→ aaaεbCbCbCT [U → ε] [Cb → bC]
→ aaabbCbCCT [Cb → bC]
→ aaabbbCCCT [T → ε]
→ aaabbbccc

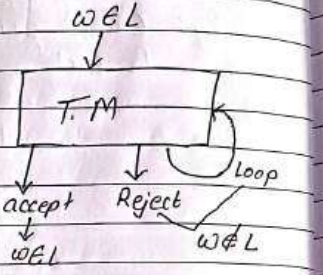
03/20

Recursively Enumerable language (REL)

A language 'L' is said to be recursively enumerable language (REL) if there exists a TM which will accept and halt for all the input string which are in language 'L'. But may or may not halt for input string which are not in L.

It means TM can loop forever for the string which are not in 'L'.

It is also known as Tuning Reconizable language or Partially decidable

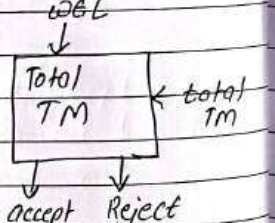


Eg: All regular language, context-free languages. Recursive language are REL

Recursive language

A language 'L' is said to be recursive if there exists a tuning machine which will accept all the string in 'L' and reject all the string not in 'L'.

The tuning machine will halt every time and give an answer (accepted or rejected) for each and every input.



Eg: $a^n b^n$, $a^n b^n c^n$ language are recursive

Language $L = a^n b^n c^n \mid n \geq 1$ is recursive because we can construct TM which will move to final state if string is of the form $a^n b^n c^n$ else move to nonfinal state. So tuning machine will always hat it in this case.

5

Undecidability

3/24

Decidable language (yes/no TM answer with)

↳ if it is recursive language (TM stop)

Partially decidable language

↳ if it is recursive enumerable language.

Undecidable language

↳ no design or made Turing Machine
↳ (Answer is not decidable by yes or no)
↳ It can be partially decidable but not decidable

Decidable problem

↳ solution

Yes No

↳ it has algorithm for given i/p

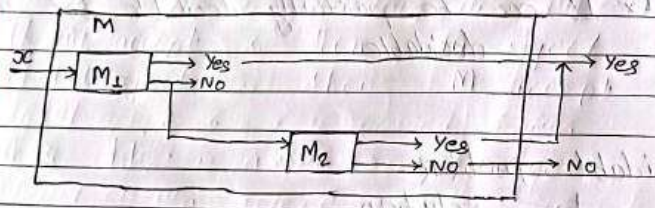
Decidable language or Problem:

1. Eg: Equivalance of two regular language,
2. Finiteness of regular language,
3. Emptiness of regular language

Example of undecidable problem language:
① Ambiguity of CFL (No, TM)
② Everything or Completness of CFG (No TM)
$\{\Sigma^*\}$ ← infinite combination

Imp
Properties of Recursive language

① Union:
If L1 and L2 are too recursive language, ~~there~~ their Union i.e. L1 ∪ L2 will also be recursive.

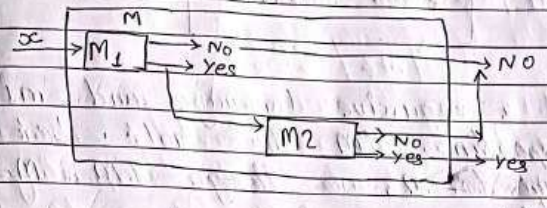


Let L2 and L2 be too recursive language and m1, m2 be be their crossponding TM that halt. let x belongs to L1∪L2 ($x \in L_1 \cup L_2$) be a string to M. So we design a TM, M = M1∪M2 such that:

1) If M1 accepts x then M accept x.
2) If M1 reject x then x is passed to M2.
3) If M2 accept x then M accept x.
4) If M2 reject x then M reject x.

Thus L1∪L2 is a recursive languages.

② Intersection
If L1 and L2 are too recursive language then their intersection i.e. L1∩L2 will also be recursive.



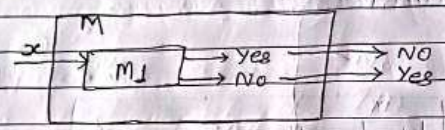
let L1 and L2 be too recursive language and M1 and M2 be their corresponding TM that halt. Let x ∈ L1∩L2 be a string to M.
so, we design a TM, M = M1∩M2 such that

1) If M1 reject x then M also reject x
2) If M1 accept x then x is passed to M2
3) If M2 accept x then M also accept x.
4) If M2 reject x then M also reject x.

Thus L1∩L2 is a recursive language.

③ Complement : $\overline{L_1}$

If L_1 is a recursive languang, then its complement i.e. $\overline{L_1}$ will also be recursive.



Let L_1 be a recursive language and m_1 be its corresponding TM that halt. Let α be a string to M, we design TM, $M = \overline{M_1}$ such that:

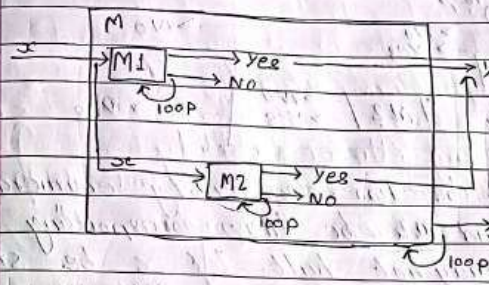
1) If M_1 accept α then M reject α
2) If M_1 reject α then M accept α

Thus $\overline{L_1}$ is also a recursive.

Properties of Recursively Enumerable Language

① Union

If L_1 and L_2 are recursively enumerable then $L_1 \cup L_2$ will also be recursively enumerable.



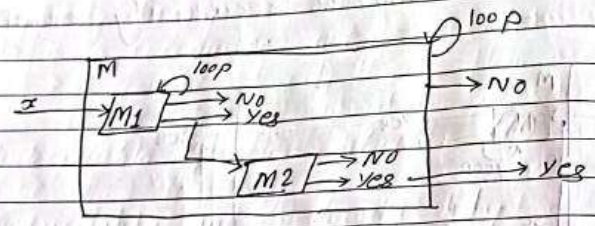
Let L_1 and L_2 be a two recursively enumerable language M_1 and M_2 are their crosponding TM that may or may not halt. Let α be string to M. We design TM, $M = M_1 \cup M_2$ such that:

1) If M_1 accept α, then M also accept α.
2) If M_2 accept α, then M also accept α
3) If M_1 and M_2 rejects or goes infinite loop, then M may reject or goes on infinite loop.

Thus $L_1 \cup L_2$ is also a recursively enumerable language.

② Intersection

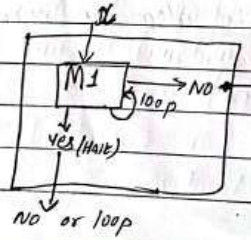
If L_1 and L_2 are two recursively enumerable then their intersection $L_1 \cap L_2$ is also recursively enumberable



Let L_1 and L_2 be two recursively enumerable language. M_1 and M_2 be their corresponding TM that may or may not halt. Let x be string to M. We design TM, $M = M_1 \cap M_2$ such that:

1) If M_1 accep x, x is passed to M_2
2) If M_2 accept x, M accept x.
3) If M_1 or M_2 reject or goes on infinte loop. Then M also may reject or goes on infinite loop

③ Complement: If L_1 is a recursively enumerable language, $\overline{L_1}$ is not recursively enumerable language.



- If M_1 accept x, then and halt then M may goes on loop or reject.
If x does not belong to L_1 then M_1 may goes on loop or reject

So for this, we cannot define accept state in TM M. So $\overline{L_1}$ is not recursively enumerable language.

Imp

Halting Problem

→ undecidable problem (no TM)

3/20

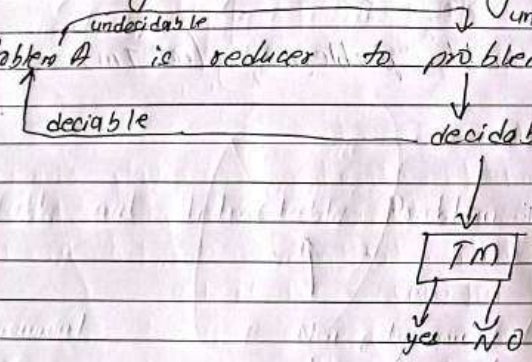
Universal Turing Machine

↳ A TM is said to be universal if it accept
  i) i/p data
  ii) description (algorithm) for computing

↳ is a recognizer not a decider

Undecidable problem about TM

① Reducibility and Undecidability



Recursive Function Theory

↳ develop ed by Alonza.church.
↳ design for natural number 'N'
There are initial function:
  1) 0-place function (z function) : z ()
     → it takes no argument and give output '0' i.e. z() = 0
  2) 1-place function : successor function
     Eg: f(n) = n+1
     f(4) = 5 , f(7) = 8
     f(n) = n²
     f(n!) = n (n-1) !
     f(n) = n+3
     f(xⁿ) = x · xⁿ⁻¹

② Primitive recursive function:
→ A function is primitive recursive function if
1) If is one ~~of~~ basic (initial) three function.
2) it is obtained by performing operation like composition, recursion or minimization on these basic three function

Three function:
① zero function, z() = 1
② Successor function, Eg f(n) = n+1
③ Projector function, $P_K(x_1, x_2, x_3, x_4, \ldots x_n) = x_K$
   Eg: $P_3(2,4,6,8) = 6$
   $P_4(1,3,5,7,9,10) = 7$

6 Computation Complexity Theory 3/30

What do mean by computational or computable?
↳ anything that can be accepted by TM
↳ if given enough time and space

(*) Complexity Theory
→ it consider not only the problem are solvable but also how efficient it can be solved.
→ problem are classified on the basis of difficulty
→ Time complexity and Space complexity are major two factor of its
(time complexity ↑ no of steps; space complexity ↑ memory used)

Tractable and Intractable Problem:

Polynomial time
(*) problem:
recursival language
adding
subtractable

Non-polynomial time
i.e. exponential time
factorial time
(*) problem:
undecidable proble

Imp

Complexity classes:
① P classes problem (easy)
② NP class problem (Hard)

① P Class problem
→ polynomial time decidable problem
→ having algorithm or TM
$O(n^{15}) = n \log^n$
→ Deterministic TM
→ Tractable problem

② NP class problem
→ Nondeterministic ~~polynomial~~ polynomial time decidable problem
→ { non-polynomial
exponential: $O(k^n)$
factorial: $O(n-1)!$
: $\log k^n$ }
→ ~~Decidable~~ Intractable problem
→ not practial applied
→ there DTM that can verified the ~~soly~~ solution in polynomial time.
→ Machine of NP class can be created but not in polynomial time
→ also know as intractable problem.
→ Eg: ① Traveling sales man
② Hamilton ~~problem~~ graph

	Decidable	Verifies
P	✓	✓
NP	X	✓

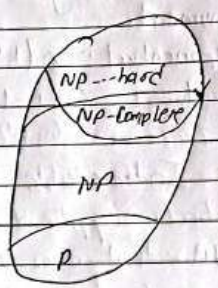
(iii) NP- hard

↳ A = NP

Now proceed according to it

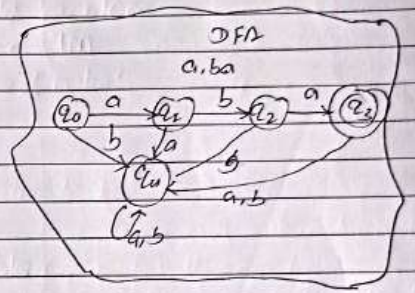
(iv) NP- complete

if problem is
- NP hard
- NP



Q. Design DFA that accept three loop keyword used in C.

- Keyword: while, for do



⇒