AUTOMATA THEORY

Automation - An abstract computing device → used for identifying pattern

A device which ~~do not have~~ need not even be a physical . Anything that is conceptual

↳ Abstract device

We will learn the grammers of the language

4 types of grammar:

1) Type 0 grammar or Recursively Enumerable grammar
2) Type 1 grammar or Content sensitive grammar
3) Type 2 grammar or Content Free Grammar
4) Type 3 grammar or Regular grammar

Who is the pioneer of Automata Theory?

↳ Alan Turing .

Turing Test → the concept of AI has came from it.

→ Acceptor of Type 0 grammar is Turing Machine

The classification of grammar is Noam Chomsky's Hierarchy

Chomsky's classification of grammar

An alphabet is a set of sequences .

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

Words: Words are sentences of strings of alphabets .

$$\Sigma = \{ 0, 1, 00, 01, 10, 11, 100, 110 \ldots \}$$

* $\begin{cases} S \longrightarrow 0A \text{ (i)} \rightarrow \text{Products and Rules of grammar} \\ A \rightarrow 0 \mid 1 \quad \text{(ii)} \end{cases}$

$S \rightarrow 0A$

↑

String (set of alphabets)

$S \rightarrow 0A$

$\rightarrow 00 \mid 01$

A → non-terminal symbol of grammar (string can't end with this non terminal symbol

Non-terminal symbol are S and A. (Capital)
Terminal symbol are 0 and 1.

$S \to 0A$. → means produces
$A \to 0 | 1 | \epsilon$.

Q) $S \to 0A$, $A \to 1A$, $A \to 0B$, $B \to 1B$, $B \to 0F$, $F \to \epsilon$

We have to start from S.

$S \to 0A | 1A | 0B | 1B$
$A \to 0A | 1A | \epsilon$
$B \to 0B | 1B | \epsilon$

$S \to 1A \to 1\epsilon \to 1$

$S \to 0A \to 0\epsilon \to 0$

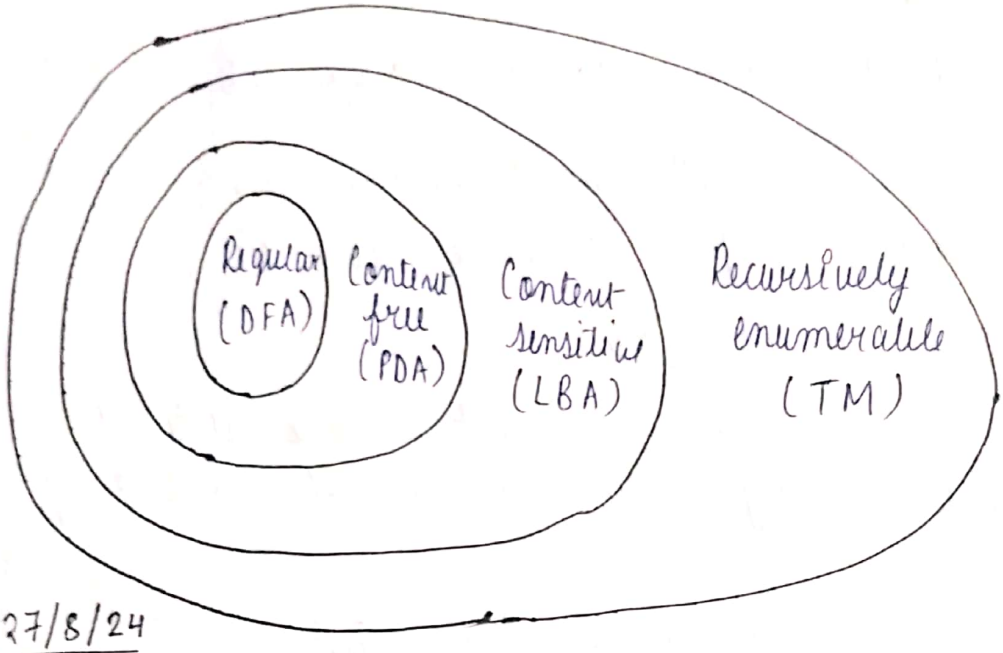
$S \to 0A \to 00A \to 00\epsilon \to 00$ (A → 0A, A → ε)

Language: A language is a collection of sentences of finite length, all constructed from a finite set of alphabets.

Grammar: A grammar can be regarded as a device that enumerate sentences of a language. nothing less nothing more.

It is a finite list of rules defining a language.

Grammar Acceptors
- * Context sensitive → Linear bounded automata
- * Context free → Push down automata
- * Regular → Just Automata



27/8/24

S is the start symbol, $S \in N$

a, b ... (small letters) → Terminal symbols.

$S \to AB$ (i)
$\to aB$ (ii)
$\to ab$ (iii)

Q) $\{\{\{S, A\}, \{a, b\}, S, \{S \to aAb, aA \to aaAbb, A \to \epsilon\}\}$

$S \to aAb$.
$S \to aaAbb$. [$aA \to aaAb$]
$S \to aabb$ [$A \to \epsilon$]

$S \to aAb$
$\to ab$ [$A \to \epsilon$]

$S \to aAb$
$\to aaAbb$. ($aA \to aaAb$)
$\to aaaAbbb$ ($A \to \epsilon$)
$\to aaabbb$.

$\{ab, aabb, aaabbb, \ldots a^n b^n\}$ $n \geq 1$

9) $L(G_1) = \{a^m b^n \mid m \geq 0, n > 0\}$

$L(G_1) = \{b, ab, abb, aab, bb, aab, abb\}$

Since $L(G_1) = \{a^m b^n \mid m \geq 0 \text{ and } n > 0\}$

the set of strings accepted can be rewritten as -

$L(G_1) = \{b, ab, aab, abb \ldots\}$

Here, the start symbol has to take at least one 'b' preceded by any number of including null.

$S \to aS$, $S \to B$, $B \to b$ and $B \to bB$

$S \to B \to b$ (Accepted)

$S \to B \to bB \to bb$ (Accepted)

$S \to aS \to aB \to ab$ (Accepted)

$S \to aS \to aaS \to aaB \to aab$ (Accepted)

$S \to aS \to aB \to abB \to abb$ (Accepted)

* $L(G_1) = \{a^m b^n \mid m > 0 \text{ and } n \geq 0\}$

$L(G_1) = \{a, aa, ab, aab, abb \ldots\}$

$S \to bS$, $S \to A$, $A \to a$ and $A \to aA$

$S \to A \to a$

$S \to A \to aA \to aa$

$S \to bS \to bA \to ba$

$S \to bS \to bbS \to bbA \to bba$

$S \to bS \to bA \to baA \to baa$

$S \to aA$, $A \to aA$, $A \to B$, $B \to bB$, $B \to \lambda$

28/8/24

Type-3 Grammar Production Rule

28/8/24 → Production must be in the form
$X \to a$ or $X \to aY$ where $X, Y \in N$ (Non-Terminal) and $a \in T$ (Terminal)

Example:

$X \to \epsilon$
$X \to a \mid aY$
$Y \to b$

→ TYPE-3 GRAMMAR

TYPE-2 GRAMMAR

$A \to Y$ → Production must be in the form

where $A \in N$ (Non-terminal)
$Y \in (T \cup N)^*$ (String of terminals and non-terminals)

Example:

$S \to Xa$
$X \to a$
$X \to aX$
$X \to abC$
$X \to \epsilon$

$*$ → repeat 0
$+$ → repeat 1
$a+$
$\{a, aa, aaa \ldots a^n\}$
$\{\epsilon, a, aa, aaa, aaaa \ldots\}$

TYPE-1 GRAMMAR

$\alpha A \beta \to \alpha \gamma \beta$

where $A \in N$ (Non-terminal)
and $\alpha, \beta, \gamma \in (T \cup N)^*$ (String of terminals and non-terminals)
α, β may be empty, but γ must be non-empty.

Ex:
$AB \to AbBC$
$A \to bcA$
$B \to b$

TYPE-0 GRAMMAR

$\alpha \to \beta$ → The productions can be in the form of

α is a string of terminals and non-terminals with at least one non-terminal and α can't be null. β is a string of terminals and non-terminals.

Example

$S \to ACaB$
$Bc \to acB$
$cB \to DB$
$aD \to Db$

Recursive Reverse of a string

The recursive nature of a string is defined by the recursive rules :-

Q) $a^R = a$

$(wa)^R = aw^R$

for all $a \in \Sigma$, $w \in \Sigma^*$, using this prove that,

$(uv)^R = v^R u^R$

→ $a^R = a$

$(wa)^R = aw^R \quad \forall\ a \in \Sigma,\ w \in \Sigma^*$

Suppose, u and v are not both empty strings

Let $u = wa$

$v = xb$

LHS

$(uv)^R$

$= (waxb)^R$

$= (xb)^R (wa)^R$ [using recursive method]

$= b^R x^R a^R w^R$

RHS

$v^R u^R$

$= (xb)^R (wa)^R$

$= b^R x^R a^R w^R$

Proved

Q) $A - (B \cup C) = (A - B) \cap (A - C)$

Q) $A - B = A \subseteq B'$

Language Concatenation

→ $\{a, ab\}\{a, ab\} \rightarrow \{aa, aab, aba, abab\}$

→ $\{a, ab\}\{bb, b\} \rightarrow \{abb, ab, abbb\}$

→ $\{a, aa\}\{a, aa\} = \{aa, aaa, aaaa\}$

* $|L_1 L_2| = m + m$, when $|L_1| = m$ and $|L_2| = n$.

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

$\{\}$ → empty language

$\{\epsilon\}$ → trivial language

$L^* = L^0 \cup L^1 \cup L^2 \cup L^3 \cup \ldots$

Q) $A-(B \cup C) = (A-B) \cap (A-C)$

LHS = $A-(B \cup C)$

$= A \cap (B \cup C)'$ [since $(A-B) = A \cap B'$]

$= A \cap (B' \cap C')$ $\because [(B \cup C)' = (B' \cap C')$ → De-morgan's law

$= (A \cap B') \cap (A \cap C')$

$= (A-B) \cap (A-C)$

$=$ RHS

Q) Let, x be an arbitrary element in $A-B$

So, $x \in A$ and $x \notin B$

Since, $x \notin B$, we know that $x \in B'$

Therefore, every element in $A-B$ is also in B'

So, $A-B \subseteq B'$ - - (i)

Now, let's consider an arbitrary element x in A

We want to show that if $x \notin B$, then $x \in B'$

if $x \notin B$, then $x \in A-B$

We already showed that, $A-B \subseteq B'$, so $x \subseteq B'$

Therefore, every element in A that is not in B also in B'

So, $A \subseteq B'$ - - (ii)

From, $A-B \subseteq B'$ - - (i) and $A \subseteq B'$ - (ii)

We can say that.

$A-B = A \subseteq B'$

Proved

AUTOMATION OF AUTOMATA

$\delta(q_0, a) = q_1$

$q_0 \xrightarrow{a} q_1$, can any

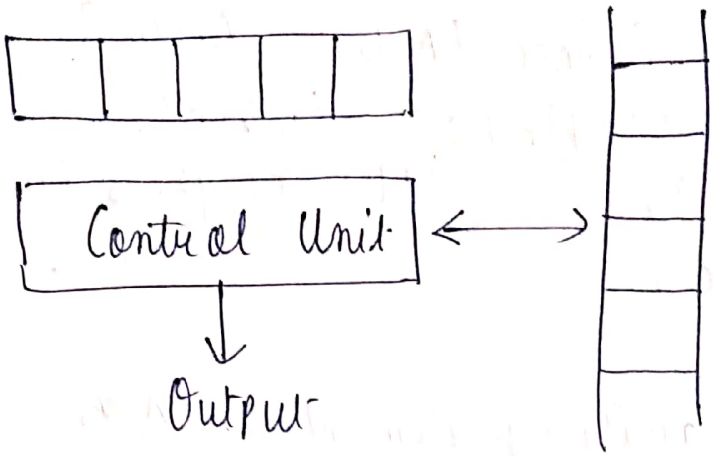


Fig: Automation

A ~~deter~~ Deterministic Finite Automata (DFA) is a 5-tuple $M = (Q, \Sigma, \delta, q_0, F)$

where [tuple → a data structure containing multiple parts]

Q = Finite ~~states~~ sets of internal states (set of all states)

Σ = Finite set of symbols called "Input Alphabet".

$\delta : Q \times \Sigma \to Q$ = Transition Function

$q_0 \in Q$ = Initial State

$F \subseteq Q$ = Set of Final States

An automation is an abstract model of a digital computer. An automation has a mechanism to read input which has a string of input alphabets. The input is actually written on an input file which can be read by automation but can't change it.

The input file is divided into sets, each of which can hold one symbol. The automation has a temporary storage device which has unlimited no of storage cells, the contents of which can be altered by the automation. Automation also has a control unit which is said to be one of finite no. of sets

and the automation can change the state in a defined way

TYPES OF AUTOMATA

1) Deterministic Finite Automation (DFA) (go to a particular state)

* 2) Non-Deterministic Finite Automation (NFA)

(In NFA ~~it to~~ from q_0 we can go to multiple no. of states)

A deterministic automation is one in which each move is (transition from one state to another) is uniquely determined by the current configuration.

If the internal state input and the content of the storage are known it is possible to predict the future behaviour of automation. This is said to be deterministic automata otherwise called non-deterministic automata

An automation whose output response is YES or NO is known as <u>acceptor</u>.

* Design a DFA, M which accepts the language.

$L(M) = \{ w \in (a,b)^* \}$

w doesn't contain three consecutive b's.

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

where,

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

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

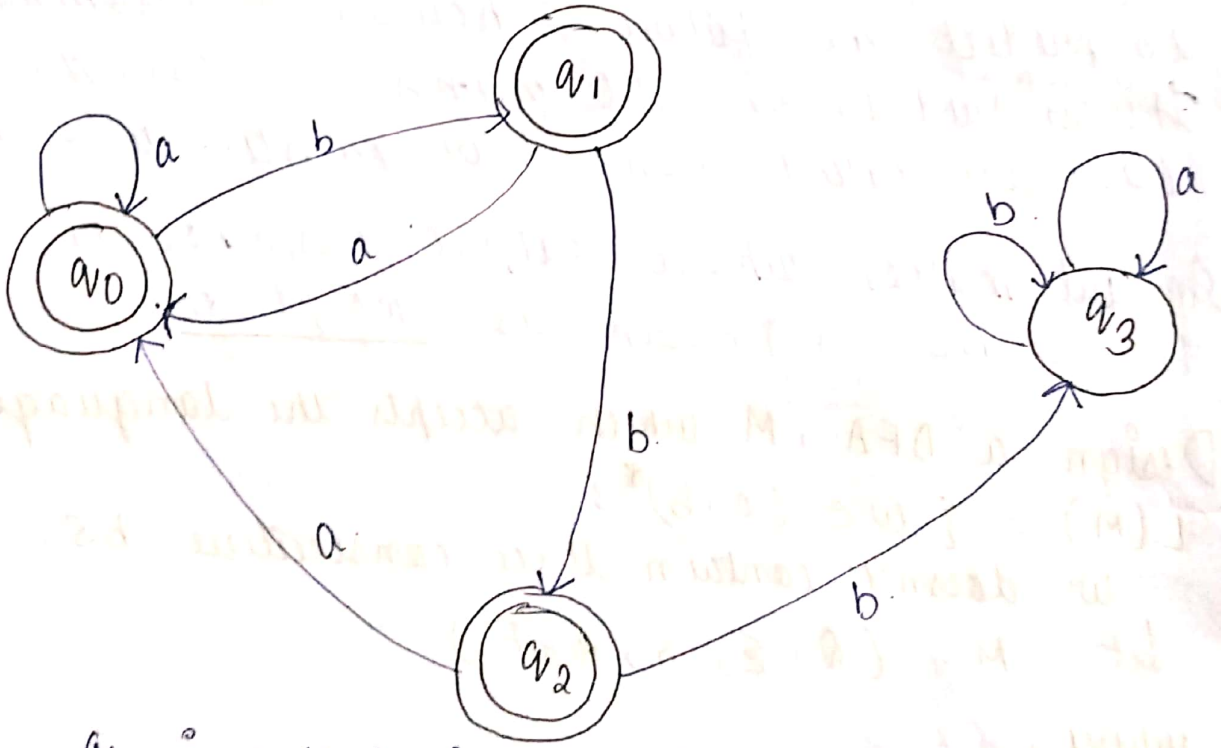
q_0 = Initial state

$F = \{ q_0, q_1, q_2 \}$ are initial states

and δ is defined as follows:

<u>Transition table of DFA:</u>

Initial State (q)	Symbol (σ)	Final State $\delta(q, \sigma)$
q_0	a	q_0
q_0	b	q_1
q_1	a	q_0
q_1	b	q_2
q_2	a	q_0
q_2	b	q_3
q_3	a	q_3
q_3	b	q_3



q_3 is called ~~Death~~ Dead state or Trap State (Because from q_3 we cannot move to any other state)

a, ab, ba, abb, abbb

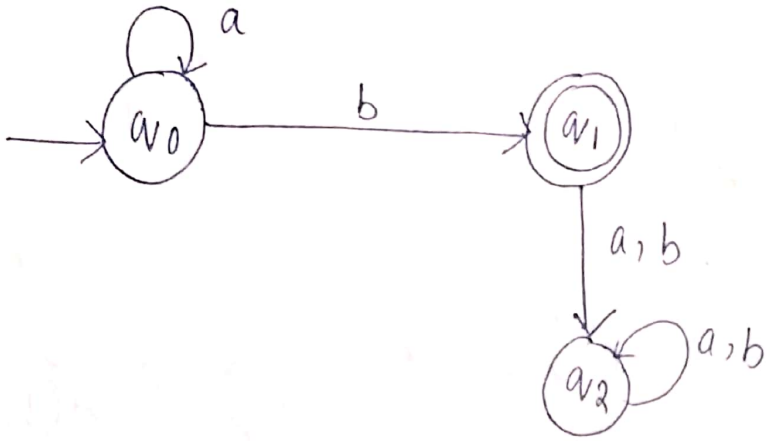
↳ is not accepted (because when we move to q_3 state we cannot go back to any other state)

Date → 3/9/24

Q) $L = \{a^n b : n \geq 0\}$

Draw the DFA

⇒



Q)

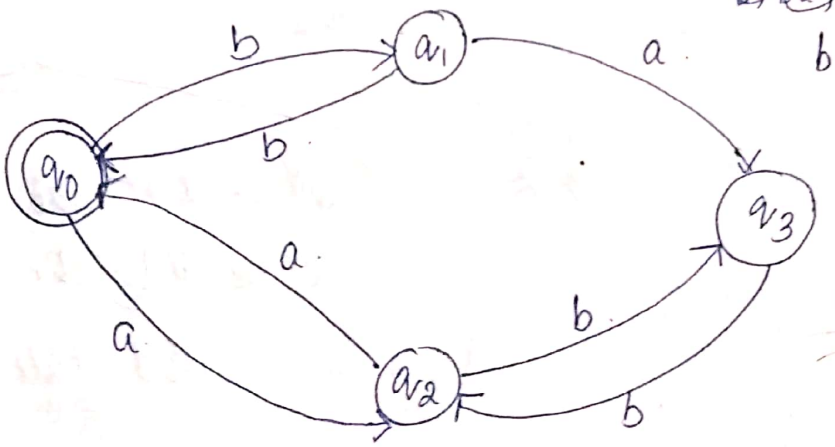
Initial state (q)	Symbol (σ)	Final state $\delta(q, \sigma)$
q_0	a	q_2
q_2	b	q_3
q_1	a	q_3
q_0	b	q_1
q_3	b	q_2
q_2	a	q_0
q_1	b	q_0

q_0 is starting state as well as final state.

Draw the DFA.

~~b, ba, aba,~~

bb, aa, ba, baba, aba

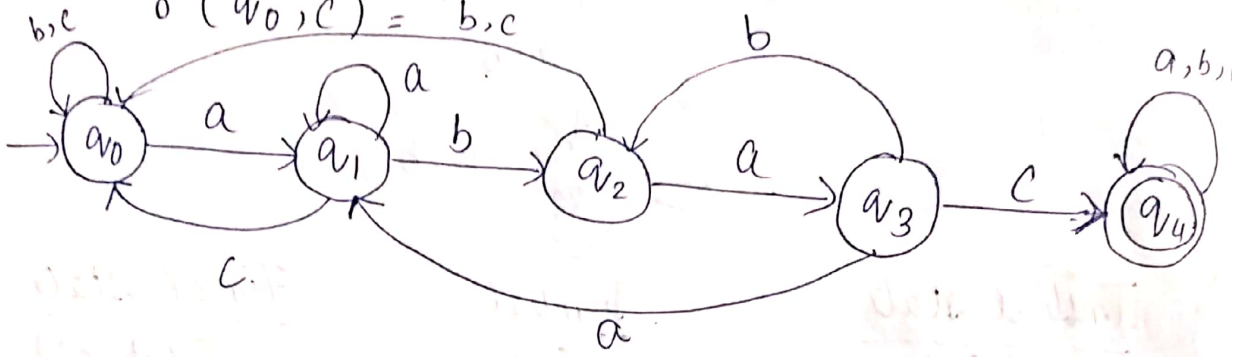


Q) $L(M) = \{w \mid w \in \{a,b,c\}^*$, w contains the pattern abac. q_4 is final state.

⇒ $\delta(q_0, a) = q_1$

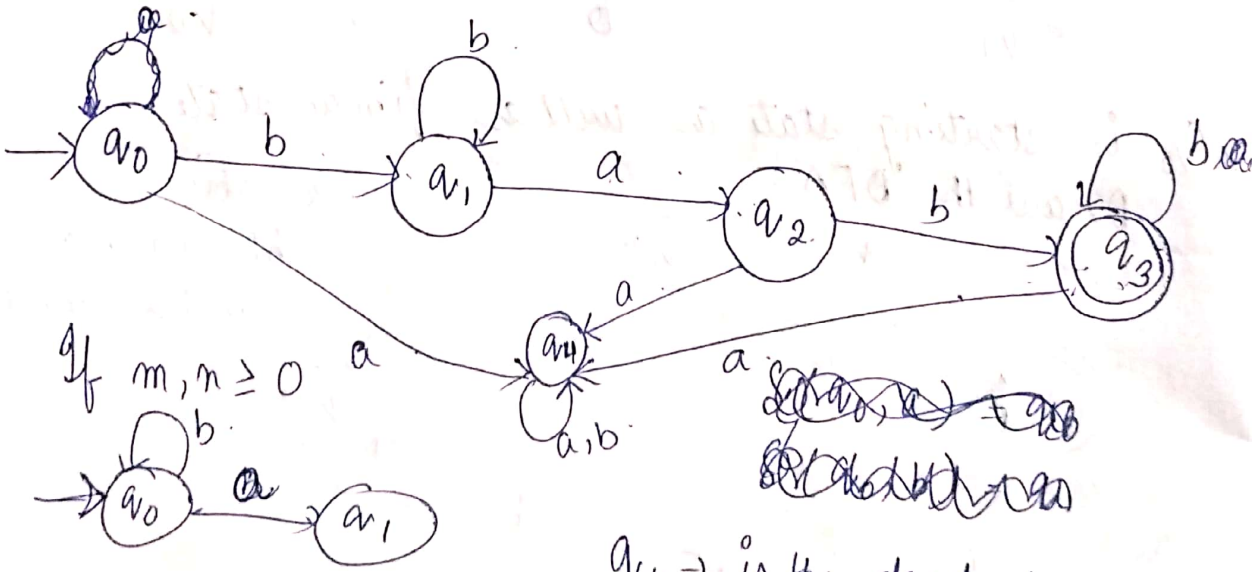
$\delta(q_0, b) = q_0$

$\delta(q_0, c) = q_0$



Q) $L(M) = \{w \mid w \in \{a,b,c\}^*$, w contains the pattern $b^m a b^n$ $\therefore m>0$, $n>0$.

⇒ bab, bbab, babb ...



If $m, n \geq 0$

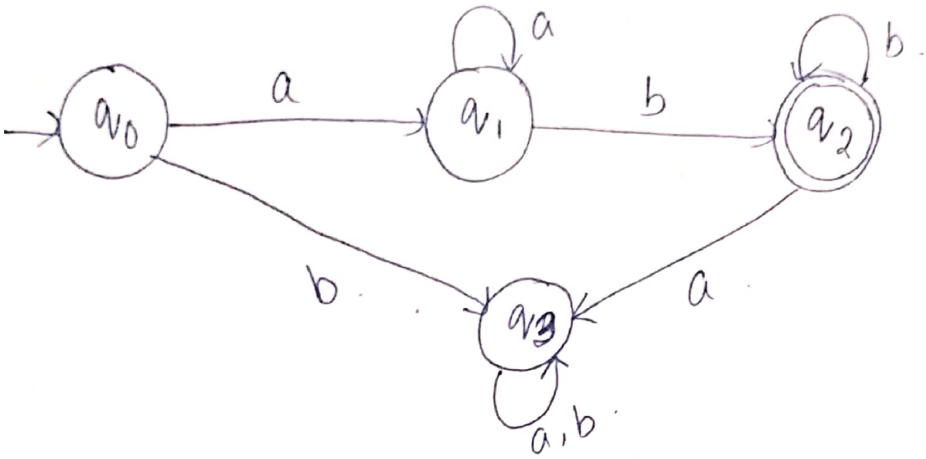
q_4 → is the dead state.

$\delta(q_0, a)$
$\delta(q_2, a)$
$\delta(q_3, a)$

} Are not present in the figure.

* $L = \{ a^m b^n, \text{ where } m, n > 0 \}$

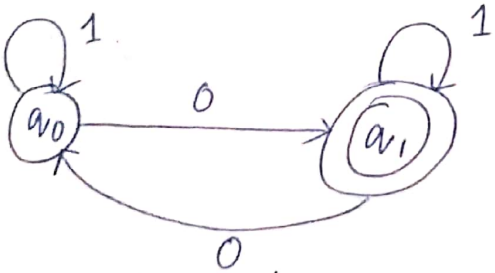
ab, abb, aab



* Draw a DFA with alphabets 0 and 1 which will recognize
  i) Strings of odd no of zeroes
  ii) Strings of odd no. of ones
  iii) Strings of even no of ones
  iv) Strings of even no of zeroes.

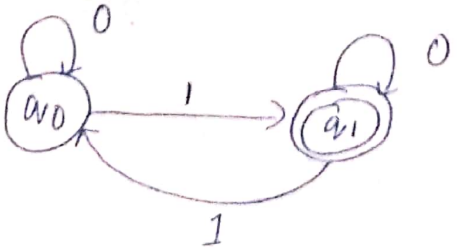
Modify this DFA which can recognize odd no. of zeros and ~~odd~~ even no. of ones
~~Od~~ even no. of ~~ones~~ zeros and odd no. of ~~ones~~ ones

i) Strings of odd no. of zeroes

→ 

{0, 01, 0100, 101, 1010101}

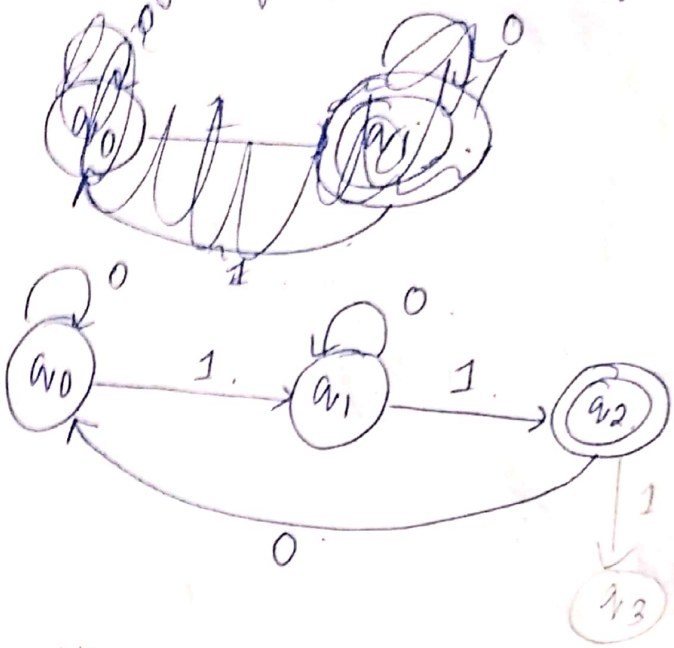
ii) Strings of odd no. of ones



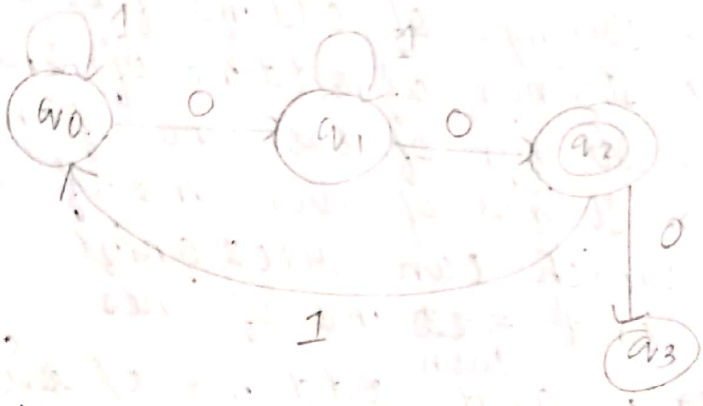
{1, 01, 010, 0101, ...}

iii) String of even no of ones

{11, 011, 0101
101 - - -}



iv) String of even no of zeros



v) String of odd no of zeros and even no of ones

* Construct a DFA that accepts:

① Odd no. of zeros and even no. of ones

② Odd no. of zeros and even no. of ones

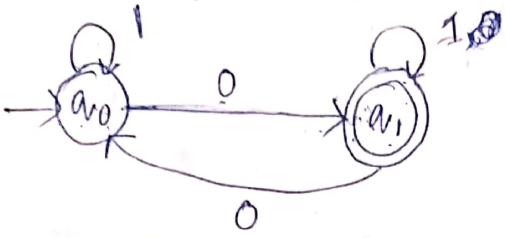
③ Either odd no. of zeros or even no. of ones but not the both together.

0, 1

① L(M) = { 101, 101

Odd no. of zeroes.

L(M) = { 0, 01, 0100, 101, 1010101 }



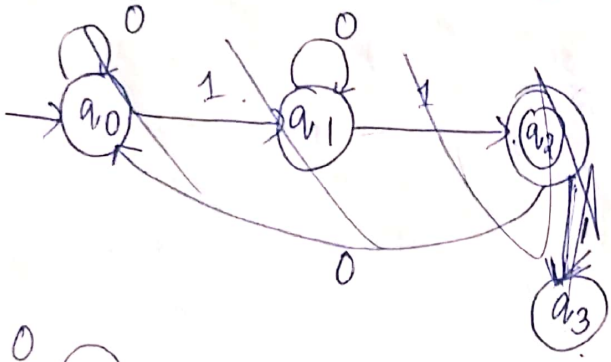
{ 0, 000001

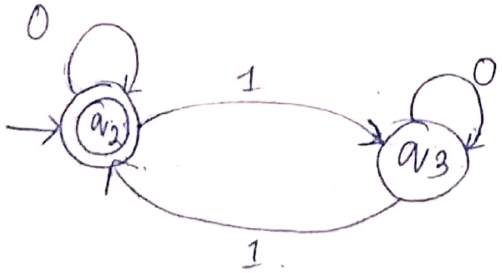
10D

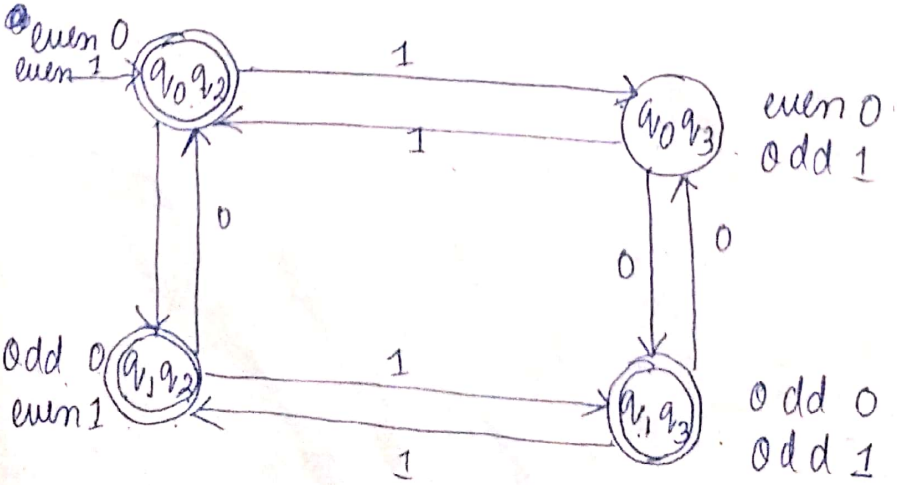
101010

Even no. of ones.

{ 11, 011, , 101 - - - }

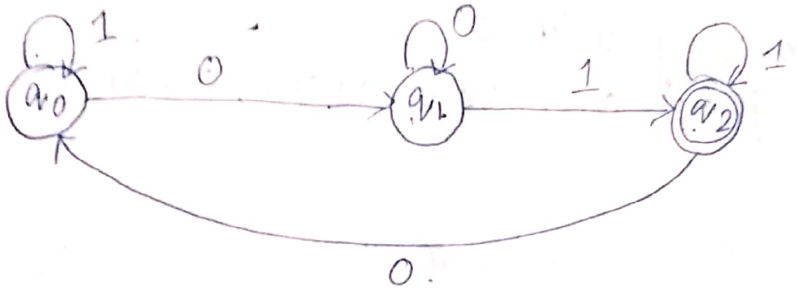




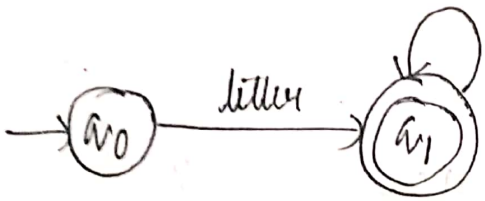


Q) Draw a Design a DFA over the alphabets Σ which accepts the set of strings either start with 01 or end with 01.

⇒ $L(M) = \{01, 1001, 101, 010, 011, 0110 \cdots\}$



Q)



a
a 1
1 a

DFA for identifier.

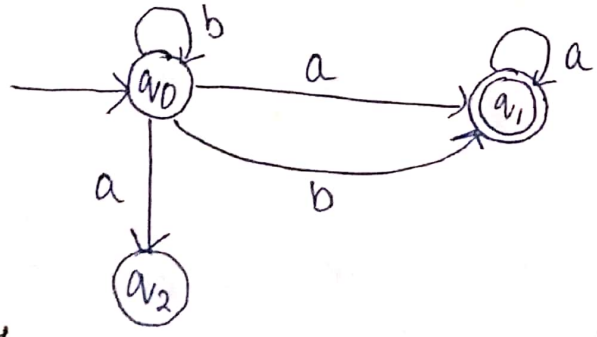
Q) Draw a single automata which can ~~define~~ recognize all the different numbers of any format.

Date: 18/09/24

NFA (Non-Deterministic finite Automata)

- NFA stands for non-deterministic finite autom
It is easy to construct an NFA, than DFA for a given regular language.
- The finite automata are called NFA when there exist many paths for specific input from the current state to the ~~no~~ next state.
- Every NFA is not DFA, but each NFA can be translated into DFA.
- NFA is defined in the same way as DFA but with the following two exceptions, it contains multiple next states, and it contains ε transitio

In the following image, we can see that from state q_0 for input a, there are two next states q_1 and q_2, similarly from q_0 for input b, the next states are q_0 and q_1. Thus it is not fixed or determined but with a particular input where to go next. Hence, this FA is called non-deterministic finite automata.



Formal definition of NFA

NFA also has five states same as DFA, but with different transition function, as shown follows:

$\delta : Q \times \Sigma \rightarrow 2^Q$ [In case of DFA, it is Q]

where,

Q = finite set of states

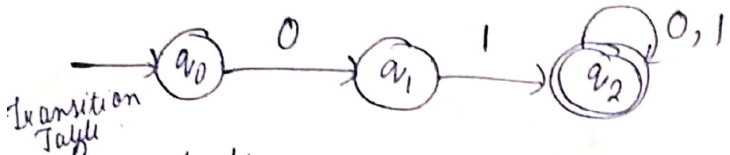
Σ = finite set of the input symbol

q_0 = Initial state

F = Final state

δ = Transition function

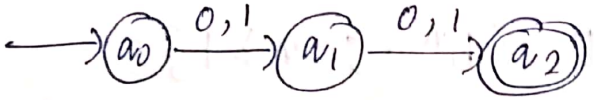
Ex: NFA with $\Sigma = \{0,1\}$ accepts strings with 0,1



Transition Table

Present State	Next state for i/p 0	Next state of i/p 1
$\rightarrow q_0$	q_1	ε
q_1	ε	q_2
$+ q_2$	q_2	q_2

Ex: NFA with $\Sigma = \{0,1\}$ and accept all string of length atleast 2.



P. State	Next state for i/p 0	Next state for i/p 1
$\rightarrow q_0$	q_1	q_1
q_1	q_2	q_2
$+ q_2$	ε	ε

Q)1 Obtain an NFA for a language consisting of all strings over 0,1 containing a 1 from the third position in the end.

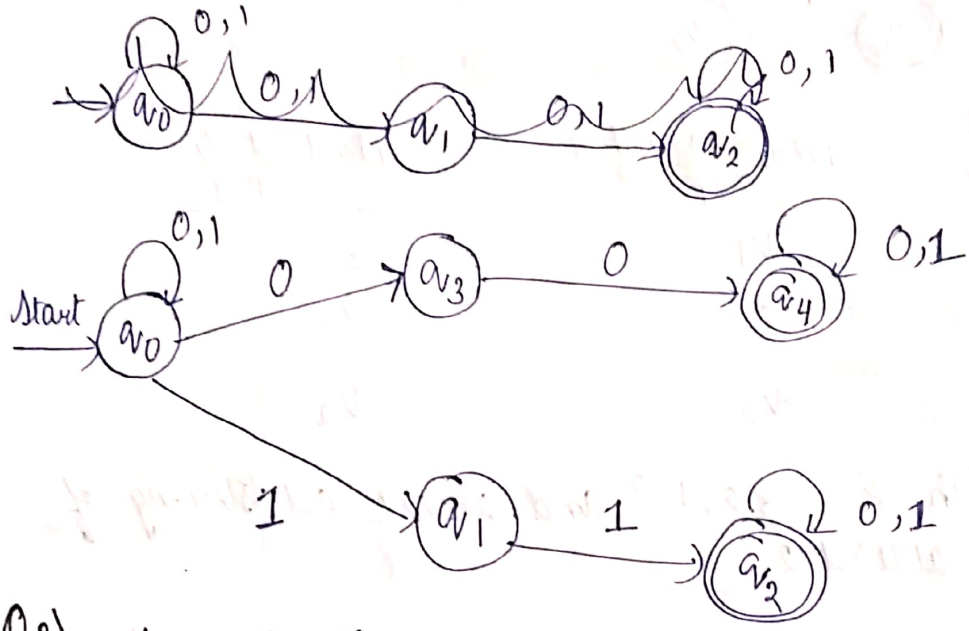
Q)2 Determine an NFA accepting all strings over 0,1, which end with 1 but does not contain substring 00.

Q)3 Find an NFA with 4 states for $L = \{a^n : n \geq 0\} \cup \{b^n a : n \geq 1\}$

Q) Determine a NFA with no more than 5 states which accepts the language $L = \{ab\,ab^n : n \geq 0\} \cup \{ab\,a^n : n \geq 0\}$

Q) Design an NFA for the alphabets 0, 1 that has atleast two consecutive zeros or 1's.

→ $L = \{00, 11, 100, 001, 110, 011 \ldots\}$



Q) $L = \{a^n : n \geq 0\} \cup \{b^n a : n \geq 1\}$

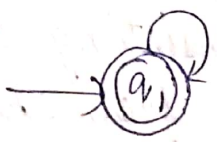
$L_1 = \{a^n : n \geq 0\}$

$= \{\lambda, a, aa, aaa \ldots\}$

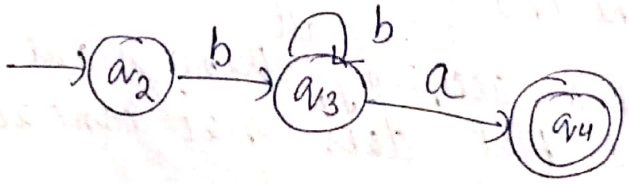
$L_2 = \{b^n a : n \geq 1\}$

$= \{ba, bba, bba \ldots\}$

L_1



L_2,

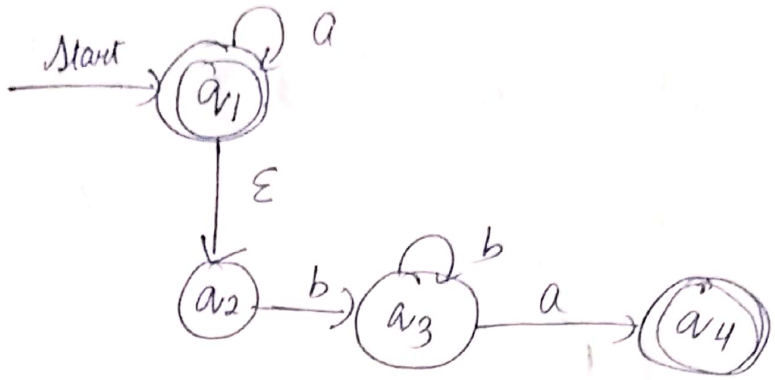


or

NFA supports ε transition

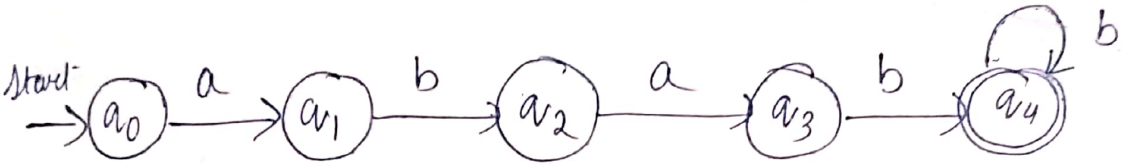
DFA does not support ε transition.

L3)



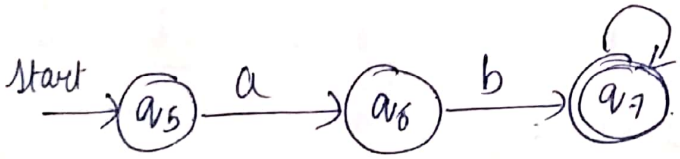
(4) $L_1 = \{ a b a b^n : n > 0 \}$

$= \{ abab, ababb, ababbb \ldots \}$

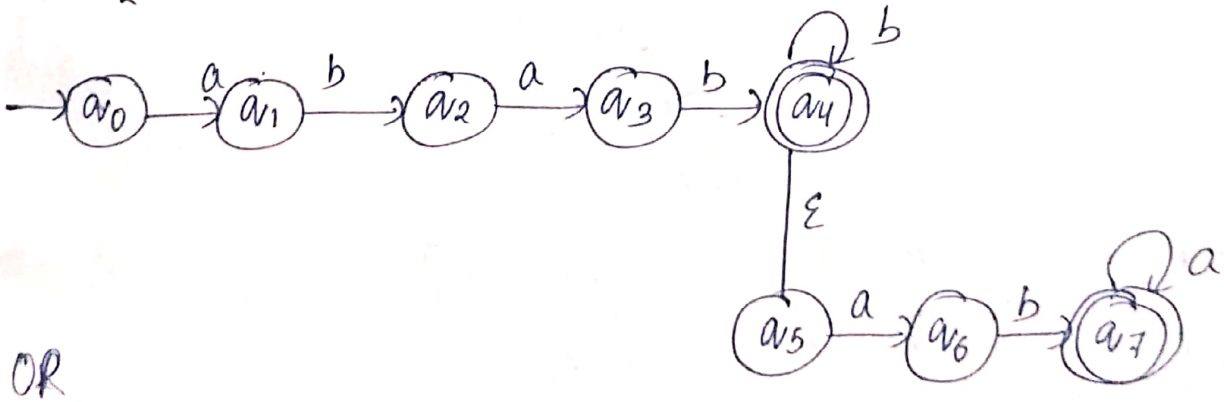


$L_2 = \{ aba^n : n \geq 0 \}$

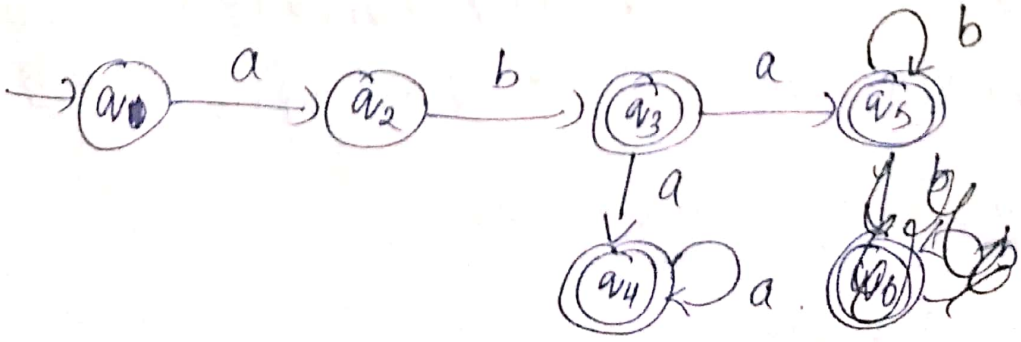
$= \{ ab, aba, abaa \ldots \}$



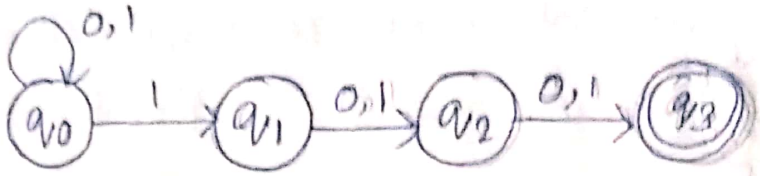
$L_1 \cup L_2$



OR

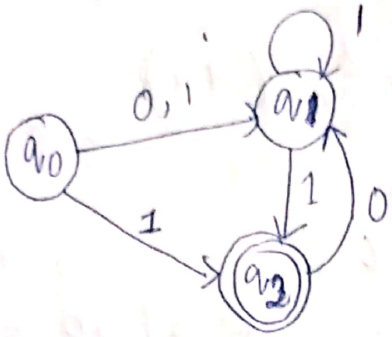


Q1)

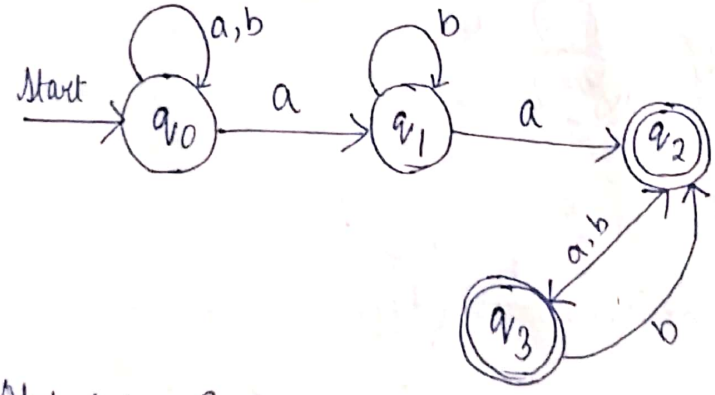


{101, 010

Q2) $L = \{1, 01, 101, 1011, \ldots\}$



Q) Convert NFA to DFA



Step 1: $\delta(q_0, a) = \{q_0, q_1\}$

Step 2: $\delta(q_0, b) = \{q_0\}$

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

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

$\delta([q_0, q_1], b) = \delta(q_0, b) \cup \delta(q_1, b)$

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

$= \{q_0, q_1\}$

Step 3: $\delta([q_0, q_1, q_2], a)$

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

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

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

$\delta([q_0, q_1, q_2], b)$

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

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

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

Step 4: $\delta([q_0, q_1, q_2, q_3], a)$

$= \delta(q_0, a) \cup \delta(q_1, a) \cup \delta(q_2, a) \cup \delta(q_3, a)$

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

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

$\delta([q_0, q_1, q_2, q_3], b)$

$= \delta(q_0, b) \cup \delta(q_1, b) \cup \delta(q_2, b) \cup \delta(q_3, b)$

$= \{q_0\} \cup \{q_1\} \cup \{q_3\} \cup \{q_2\}$

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

AND

$\delta([q_0, q_1, q_3], a)$

$= \delta(q_0, a) \cup \delta(q_1, a) \cup \delta(q_3, a)$

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

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

$\delta([q_0, q_1, q_3], b)$

$= \delta(q_0, b) \cup \delta(q_1, b) \cup \delta(q_3, b)$

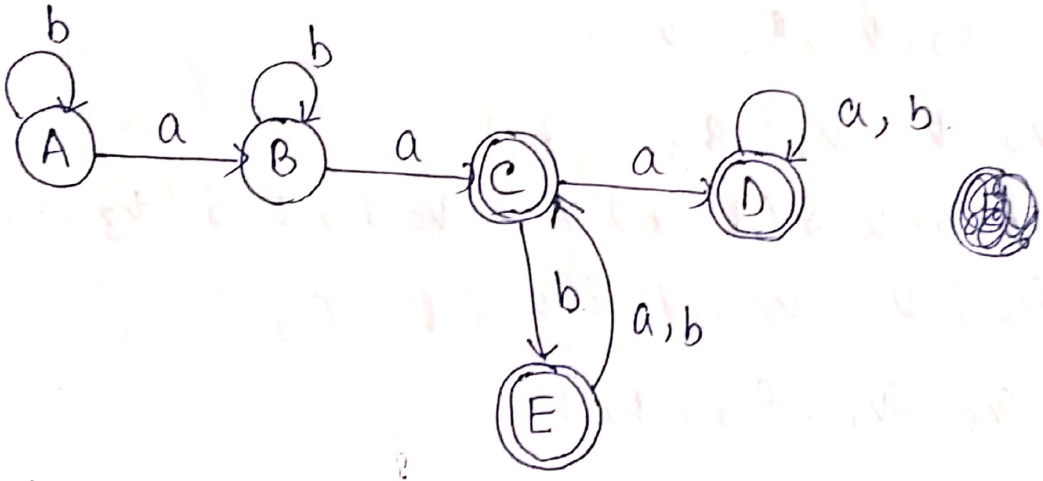
$= \{q_0\} \cup \{q_1\} \cup \{q_2\}$

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

States we have got:

1) $\{q_0\} \rightarrow A$
2) $\{q_0, q_1\} \rightarrow B$
3) $\{q_0, q_1, q_2\} \rightarrow C$
4) $\{q_0, q_1, q_2, q_3\} \rightarrow D$
5) $\{q_0, q_1, q_3\} \rightarrow E$

$\delta(A,a) = B$	$\delta(B,a) = C$	$\delta(C,a) =$
$\delta(A,b) = A$	$\delta(B,b) = B$	$\delta(C,b) =$
$\delta(D,a) = D$	$\delta(E,a) = C$	
$\delta(D,b) = D$	$\delta(E,b) = C$	

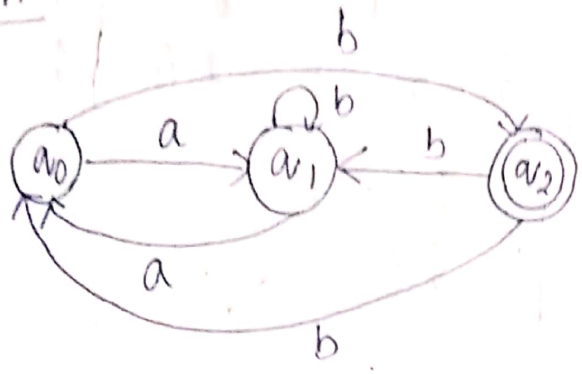


Q) Transition table for NFA.

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

q_2 is the final state

NFA



$\delta(a_0, a) = \{a_0, a_1\}$

$\delta(a_0, b) = \{a_2\}$

$\delta(a_1, a) = \{a_0\}$

$\delta(a_1, b) = \{a_1\}$

$\delta(a_2, a) = \{\in\}$

$\delta(a_2, b) = \{a_0, a_1\}$

$\delta([a_0, a_1], a)$

$= \delta(a_0, a) \cup \delta(a_1, a)$

$= \{a_1\} \cup \{a_0\}$

$= \{a_0, a_1\}$

$\delta([a_0, a_1], b)$

$= \delta(a_0, b) \cup \delta(a_1, b)$

$= \{a_2\} \cup \{a_1\}$

$= \{a_1, a_2\}$

$\delta([a_1, a_2], a)$

$= \delta(a_1, a) \cup \delta(a_2, a)$

$= \{a_0\} \cup \{\in\}$

$= \{a_0\}$

$\delta([a_1, a_2], b)$

$= \delta(a_1, b) \cup \delta(a_2, b)$

$= \{a_1\} \cup \{a_0, a_1\}$

$= \{a_0, a_1\}$

$A \rightarrow \{a_2\}$

$B \rightarrow \{a_0\}$

$C \rightarrow \{a_0, a_1\}$

$D \rightarrow \{a_1, a_2\}$

$D \rightarrow \{\in\}$

$E \rightarrow \{\phi\}$

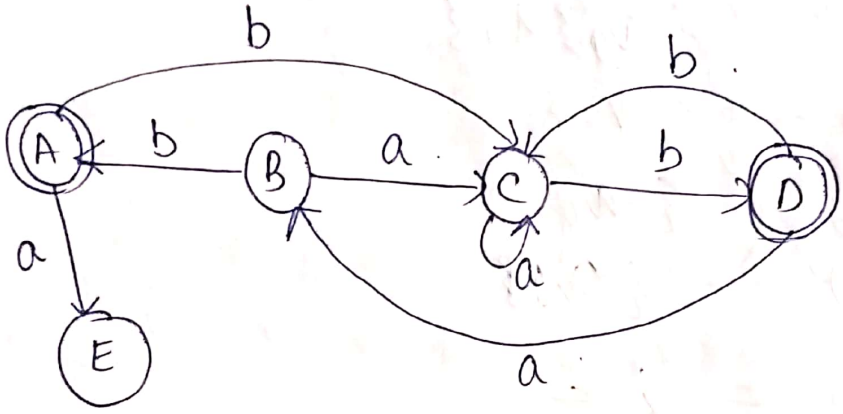
$\delta(A,a) = E$
$\delta(A,b) = B\ C$

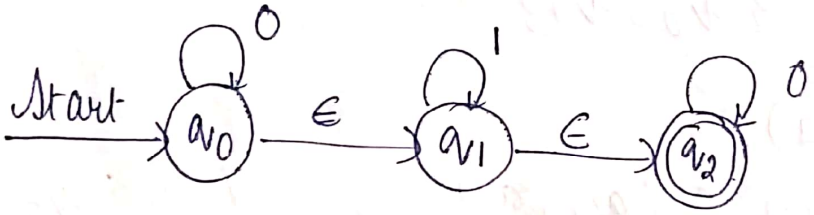
$\delta(B,a) = B\ C$
$\delta(B,b) = C\ A$

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

$\delta(D,a) = B$
$\delta(D,b) = C$

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





⇒ ϵ-closure $\{q_0\} = \{q_0, q_1, q_2\}$
ϵ-closure $\{q_1\} = \{q_1, q_2\}$
*** ϵ-closure $\{q_2\} = \{q_2\}$

* Consider the NFA with ϵ-Transition

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

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

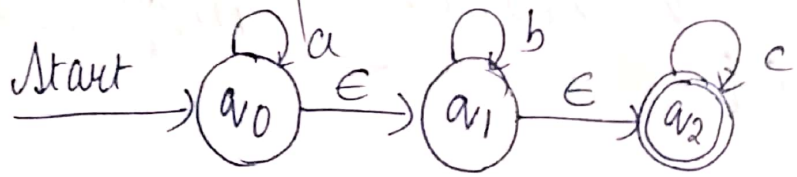
$\Sigma = \{a, b, c\}$ and ϵ moves

Initial state $= \{q_0\}$

$F = \{q_2\}$

Transition Table

δ / Σ	a	b	c	ϵ
q_0	q_0	ϕ	ϕ	q_1
q_1	ϕ	q_1	ϕ	q_2
q_2	ϕ	ϕ	q_2	ϕ



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

ϵ- closure $\{q_1\}$ = $\{q_1, q_2\} \rightarrow Q_Y$

ϵ- closure $\{q_2\}$ = $\{q_2\} \rightarrow Q_Z$

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

Initial state = $\{q_0, q_1, q_2\}$

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

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

= ϵ- closure $\{(q_0, q_1, q_2), a\}$

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

= " $\{q_0\} \cup \phi \cup \phi$

= $\{q_0\}$ ϵ- closure $\{q_0\}$

= $\{q_0, q_1, q_2\} \rightarrow Q_X$

$\delta' \{(q_0, q_1, q_2), b\}$

= ϵ- closure $\{(q_0, q_1, q_2), b\}$

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

= " $\{\phi \cup q_1 \cup \phi$

= ϵ- closure $\{q_1\}$

= $\{q_1\}$ $\{q_1, q_2\} \rightarrow Q_Y$

$\delta'\{(q_0, q_1, q_2), c\}$

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

$= \epsilon$ - closure $\{q_2\}$

$= \{q_2\} \rightarrow Q_Z$

$\delta'(\{(q_1, q_2), a\}$

$= \epsilon$ - closure $\{(q_1, q_2), a\}$

$= \epsilon$ - closure $\{\epsilon\epsilon\} \phi$ → No need to consider ϕ

$\delta'\{(q_1, q_2), b\}$

$= \epsilon$ - closure $\{(q_1, q_2), b\}$

$= \epsilon$ - closure $\{q_1\}$

$= \{q_1, q_2\} \rightarrow Q_Y$

$\delta'\{(q_1, q_2), c\}$

$= \epsilon$ - closure $\{(q_1, q_2), c\}$

$= \epsilon$ - closure $\{q_2\}$

$= \{q_2\} \rightarrow Q_Z$

$\delta'(\{q_2, a\}$

$= \epsilon$ - closure $\{q_2, a\}$

$= \epsilon$ - closure ϕ

$= \phi$

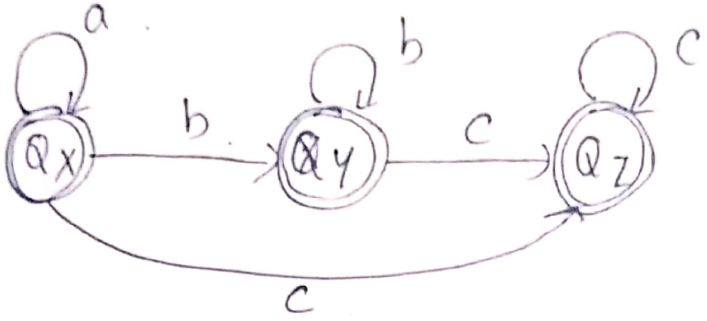
$\delta'\{q_2, b\} \rightarrow \phi$

$\delta'\{q_2, c\} \rightarrow \epsilon$ - closure $\{q_2, c\}$

$= \epsilon$ - closure $\{q_2\}$

$= \{q_2\} \rightarrow Q_Z$

$\delta(Q_X, a) \to Q_X$	$\delta(Q_Y, a) \to \phi$	$\delta(Q_Z, a) \to \phi$
$\delta(Q_X, b) \to Q_Y$	$\delta(Q_Y, b) \to Q_Y$	$\delta(Q_Z, b) \to \phi$
$\delta(Q_X, c) \to Q_Z$	$\delta(Q_Y, c) \to Q_Z$	$\delta(Q_Z, c) \to Q_Z$



***

Q ϵ-closure $\{q_0\} = \{q_0, q_1, q_2\} \to q_X$

ϵ-closure $\{q_1\} = \{q_1, q_2\} \to q_Y$

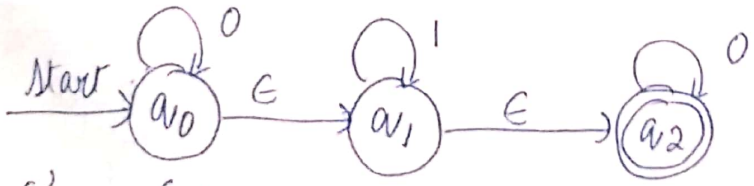
ϵ-closure $\{q_2\} = \{q_2\} \to q_2$

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

Initial state $= \{q_0, q_1, q_2\}$

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

$\delta' \{(q_0, q_x, q_z), +\}$



$\delta' = \{(q_0, q_1, q_2), 0\}$

$= \epsilon$-closure $\{(q_0, 0) \cup (q_1, 0) \cup (q_2, 0)\}$

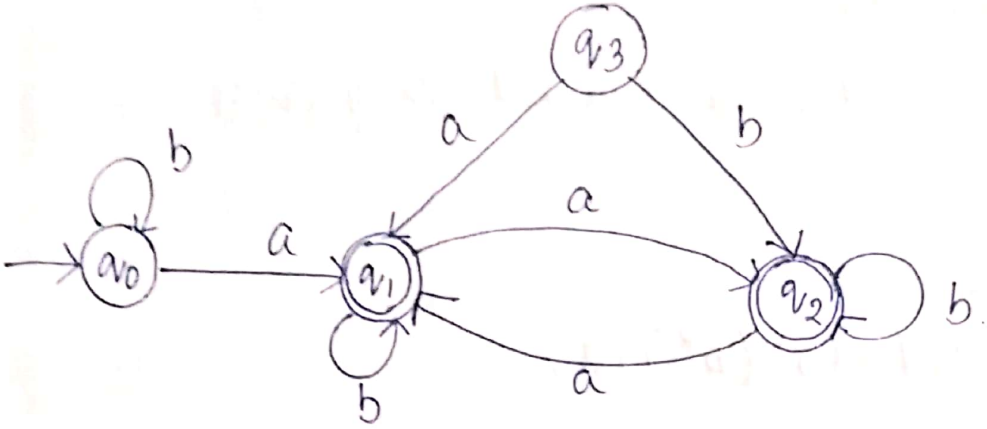
$= \epsilon$-closure $\{q_0\} \cup \{\phi\} \cup \{q_2\}$

$= \epsilon$-closure $\{q_0, q_2\}$

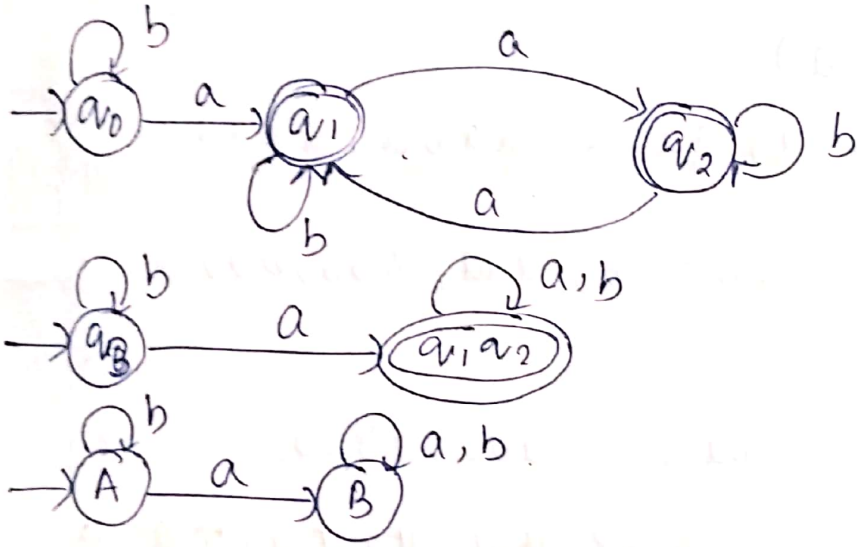
$\delta' = \{(q_0 q_1 q_2),$

$= \epsilon$-closure $\{q_0\} \cup \epsilon$-closure $\{q_2\}$

Minimization



Removing the inaccessible state q_3.



① Remove unreachable states

② Classes

- Accepting final
- Non-accepting (Non final)

Date: 22/10/24

Regular Expressions

It is a way to describe formal languages

$\epsilon \rightarrow \{\epsilon\}$

$\phi \rightarrow \{\}$

$a \rightarrow \{a\}$

Example: • $(((a+b)(b^*))a)$

→ {aa, ba, abba, aba, abba, bbba, abbba, bbbba, ...}

• $((a((a+b)^*))a)$

→ {aa, aaa, aba, aaaa, aaba, abaa, abba ...}

• $((a^*)(b^*))$

→ {ϵ, a, b, aa, ab, bb, aaa, abb, bbb ...}

• $((ab)^*)$

→ {ϵ, ab, abab, ababab, abababab, ...}

Q) All strings over {a, b} that start with an a.

→ ~~((a+b)(a*))a~~ $a(a+b)^*$

Q) All strings over {a, b} that are even in length

→ $((a+b)(a+b))^*$

Q) All strings over {0, 1} that have an even no of 1's

→ $0^*(10^*10^*)^*$

Q) All strings over a, b that start and end with same letter

a, b

→ $a(a+b)^*a + b(a+b)^*b + a + b$

Q) All strings over {0,1} with no occurrences of 00

→ 1*(011*)*(0+1*)

Q) All strings over {0,1} with exactly one occurrence of 00

→ 1*(011*)00(11*0)*1*

Q) All strings over {0,1} that contain 101

→ (0+1)*101(0+1)*

Q) All strings over {0,1} that do not contain 01

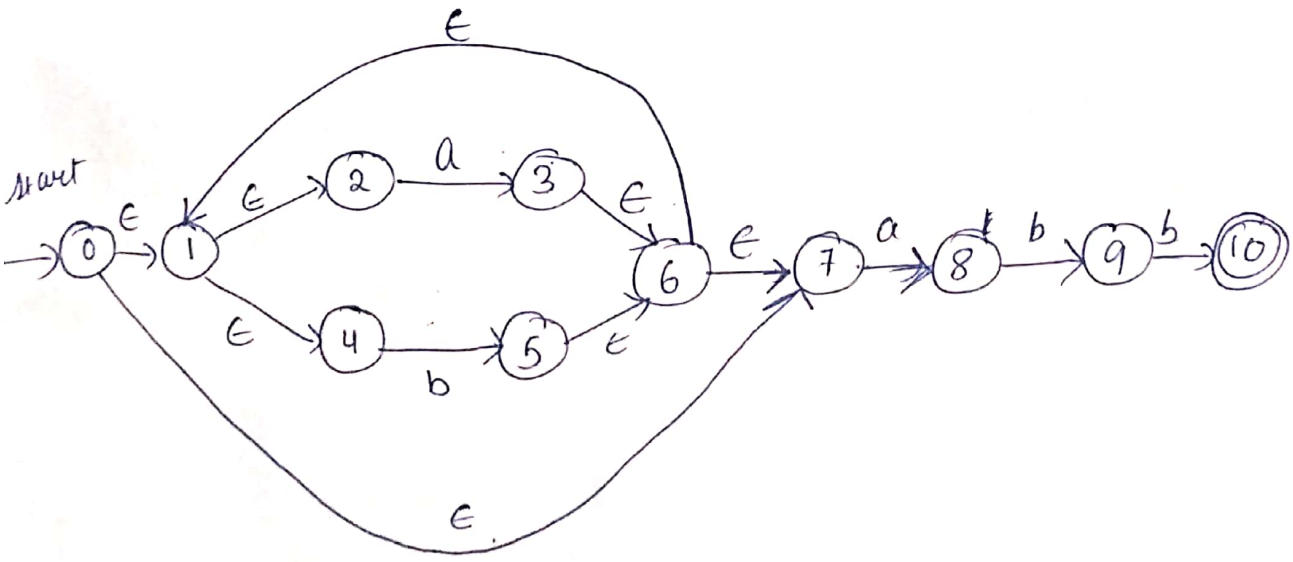
→ 1*0*

Regular Expression to NFA to DFA:

Can be done through Thompson's construction

Step 1: Conversion of regular expression to NFA.

Step 2: Conversion of NFA to DFA.



ϵ-closure(0) =

A = {0,1,2,4,7}

move(A,a) = {3,8}

move(A,b) = {5}

B = ε - closure {move (A, a)} = {1, 2, 3, 4, 6, 7, 8}

C = ε - closure {move (A, b)} = {1, 2, 4, 5, 6, 7}

move (B, a) = {3, 8} = {1, 2, 3, 4, 6, 7, 8}

move (B, b) = {5, 9} = {1, 2, 4, 5, 6, 7, 9}

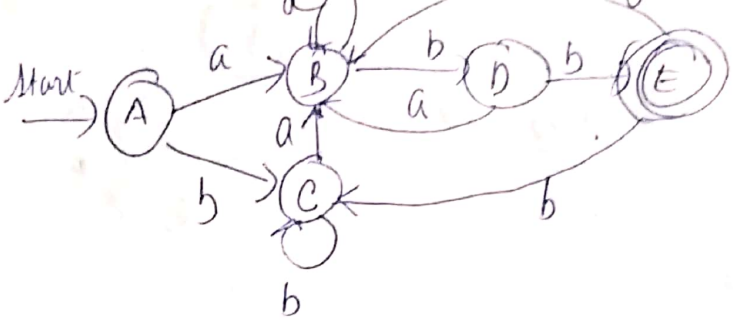
↳ D

move (C, a) = {8, 8}

move (C, b) = {5}

ε - closure {move (C, a)} = {

3, 6, 1



	Space	Time
NFA →	$O(\|r\|)$	$O(\|r\| \times \|w\|)$
DFA →	$O(2^{\|r\|})$	$O(\|w\|)$

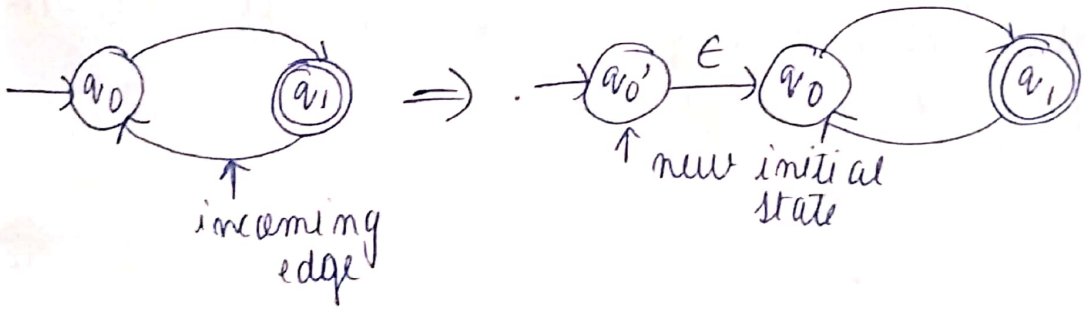
DFA to Regular Expression

1) Arden's Method
2) State Elimination Method

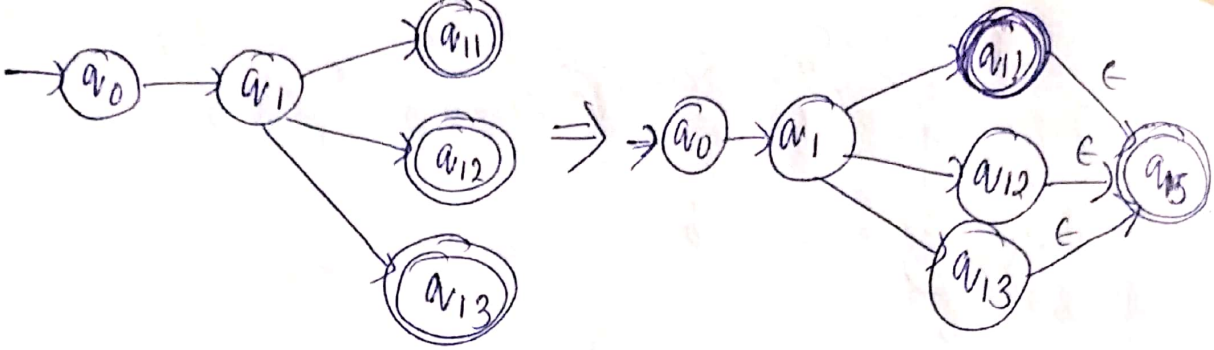
State Elimination Method

In this method we use the following steps to convert the DFA to corresponding regular expression

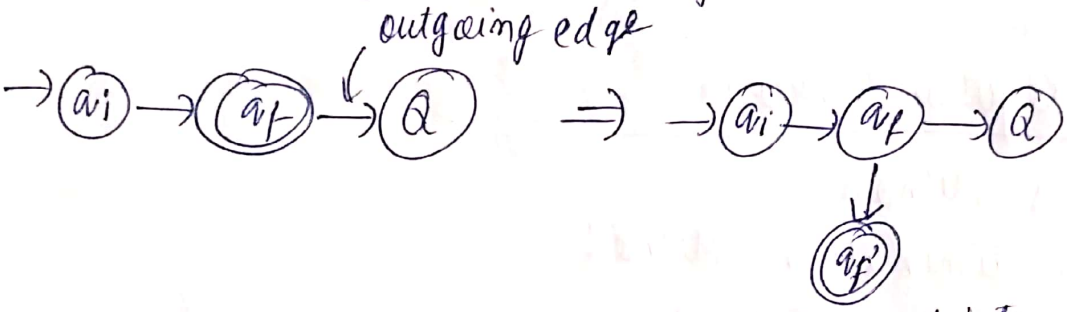
Step 1: If there exist any incoming edge ~~state to the~~ to the initial state then create a new initial state having no incoming edge.



Step 2: If there exists multiple final states in the DFA then convert all the final states into non final states and create a new single final state.



Step 3: If there exists any outgoing edge from the final state then create a new final state having no outgoing edge from it.



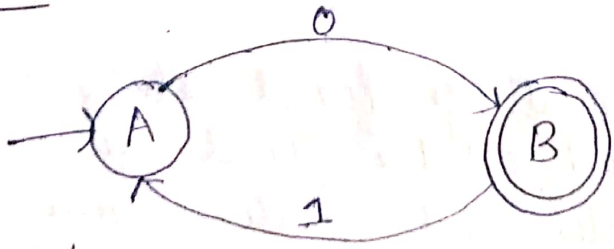
Step 4: (i) Eliminate all the intermediate states one by one.

(ii) These states may be eliminated in any order.

Only an initial state going to the final state will be left and the cost of this transition is the required regular expression.

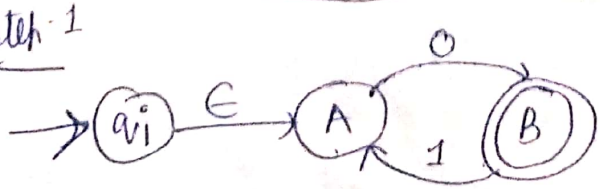
* This State Elimination Method is applicable to any type of finite automata.

* DFA



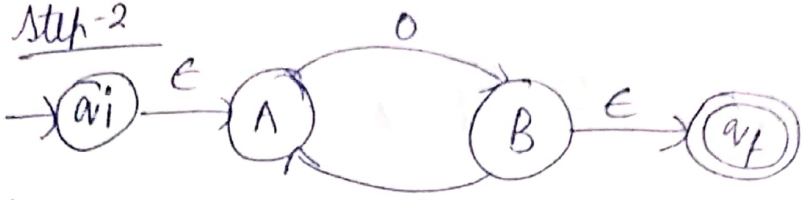
Step 1

$\Rightarrow$

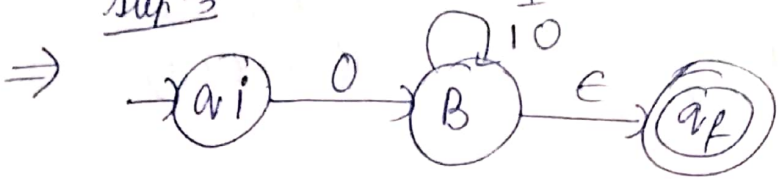


$\epsilon . 0 = 0$

Step-2

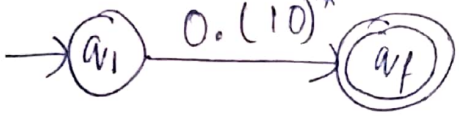


Step-3



$AB = BA$

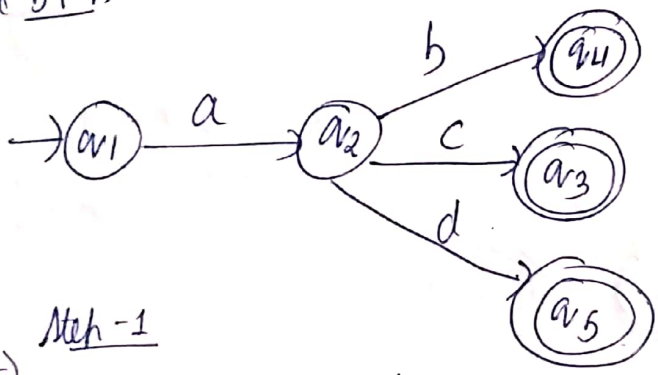
Step-4



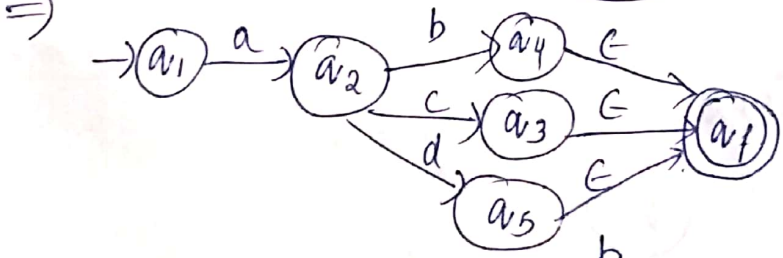
Regular Expression: $0(10)^*$

By eliminating B first regular expression will be $(10)^* 0$

* DFA



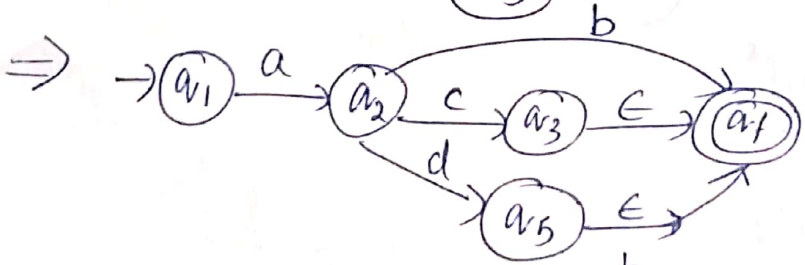
Step-1

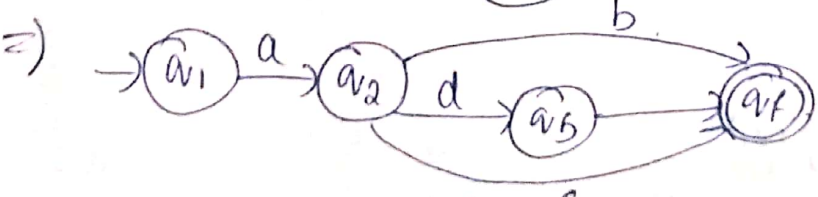


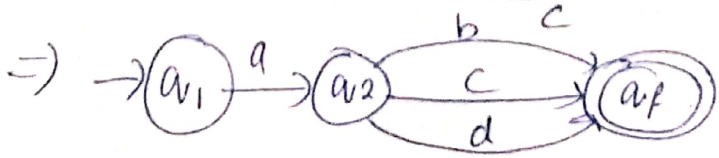
$b \cdot \epsilon = b$

$c \cdot \epsilon = c$

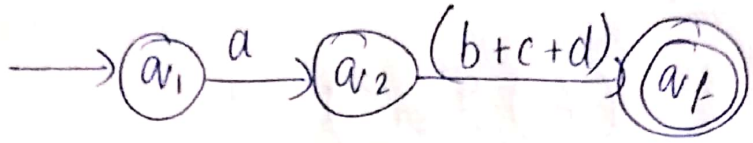
$d \cdot \epsilon = d$



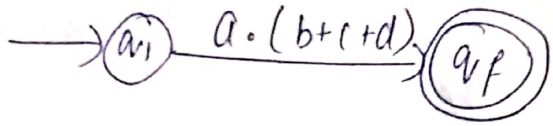




=)



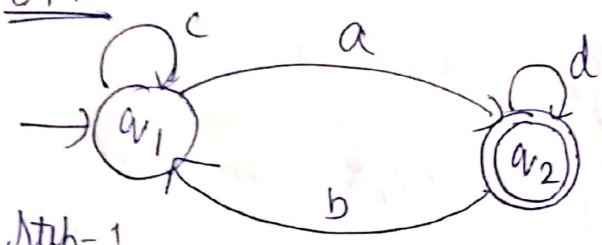
=)



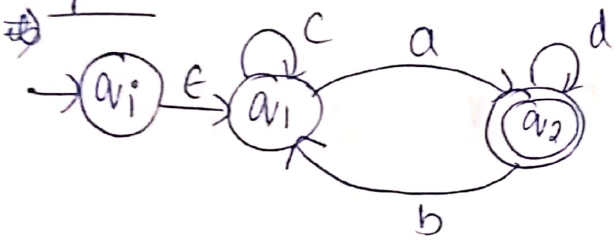
b or c or d
γ(b+c+d)

Regular Expression: $a.(b+c+d)$

DFA

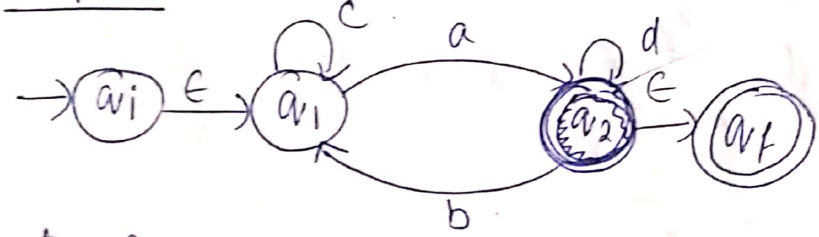


Step-1

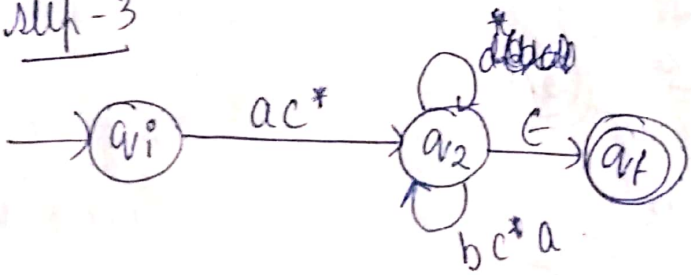


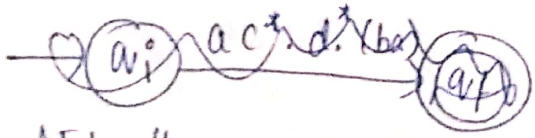
Step-2

[q_2 is not a final state in step-2

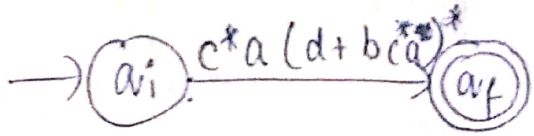


step-3



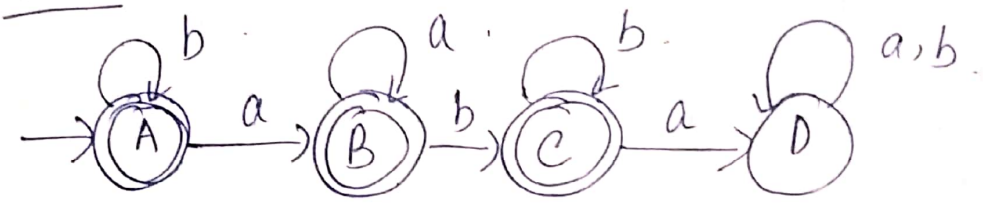


Step-4

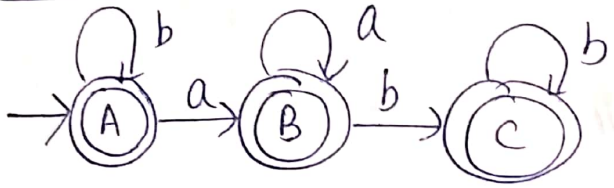


Regular Expression: $c^*a(a + bc^*a)^*$

DFA

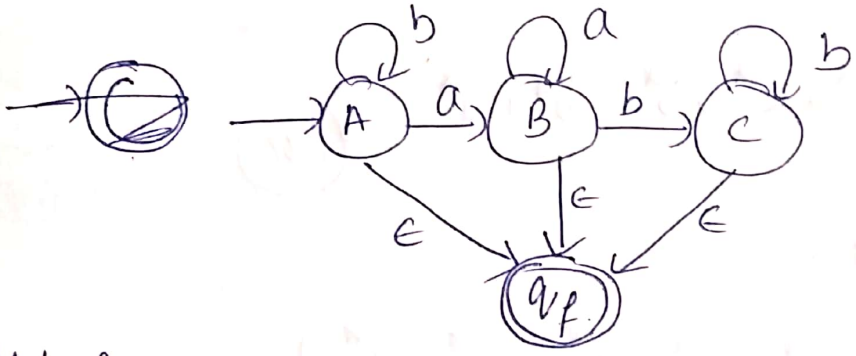


Step-1

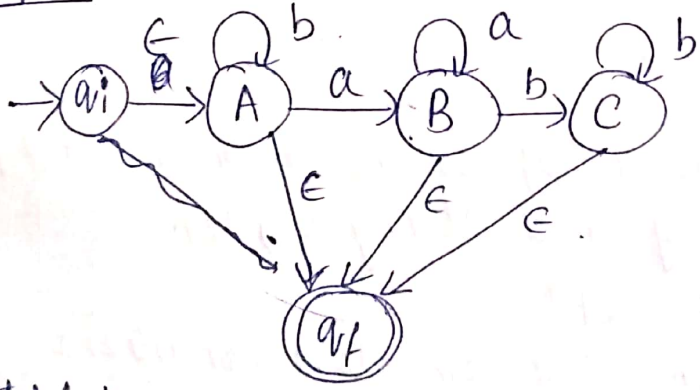


[Eliminating the trap state]

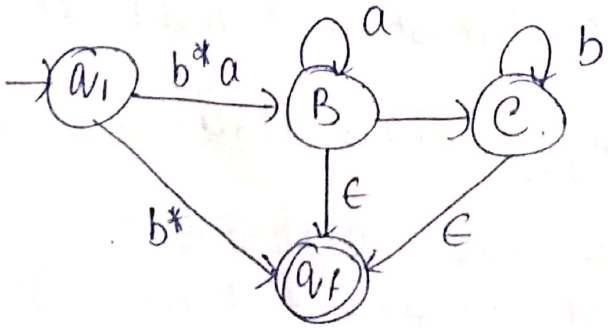
Step-2



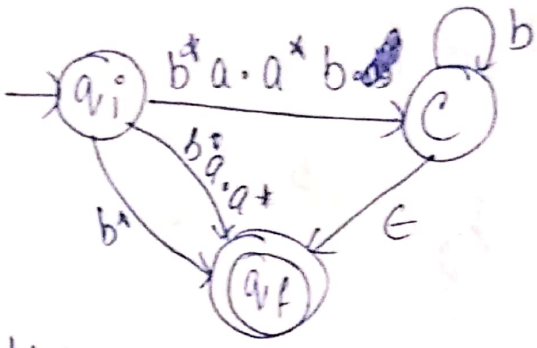
Step-3



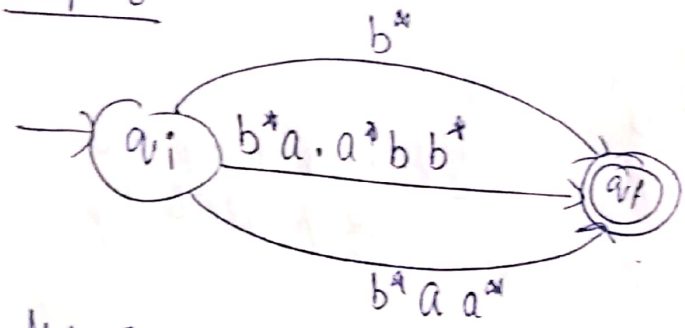
Step-4



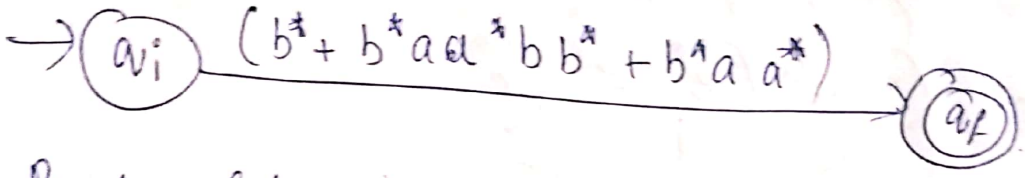
Step - 5



Step - 6



Step - 7



Regular Expression:

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

Arden's Theorem

If P, Q are two regular expressions over Σ, and P does not contain any null string ϵ, then

$$R = Q + RP \Rightarrow R = Qp^*$$

In order to use Arden's Theorem two conditions needs to be satisfied:

i) The transition diagram must not have any ϵ Transition.

ii) There must be a single initial state.

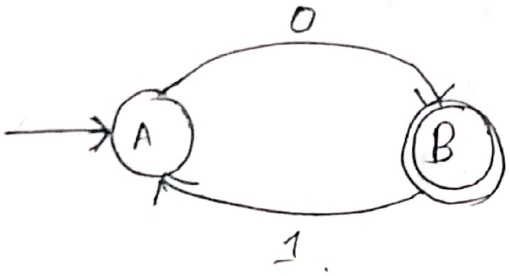
Steps to convert a DFA into corresponding regular expression using Arden's Theorem:

Step 1: Form a equation for each state considering the transitions that comes towards it. If state A has ϵ in the equation of the initial state

Step 2: Write down the final state in the form $Q+RP$ to get the desired regular expression. If there exists multiple final state then write a regular expression for each final state seperately and add all the regular expressions to get the final one

(1)

DFA:



~~$A = \epsilon + B.1$ ... (i)~~

Step 1:

$A = \epsilon + B.1$ ... (i)

$B = A.0$ ... (ii)

Putting value of A in equation (ii)

$B = (\epsilon + B.1).0$

$\Rightarrow B = \epsilon.0 + B.1.0$

$\Rightarrow B = 0 + B(10)$

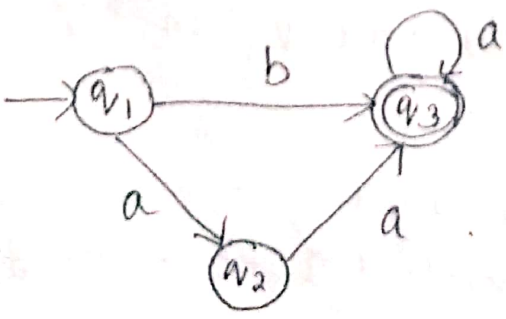
$B = 0.(10)^*$

~~$R = Q1$~~

i.e Regular expression: $0.(10)^*$

(2)

DFA:



Equations

$q_1 = \epsilon$ ... (i)

$q_2 = q_1 \cdot a$ ... (ii)

$q_3 = q_1 \cdot b + q_2 \cdot a + q_3 \cdot a$ ... (iii)

Putting the values of q_1 and q_2 in Eq (iii), we get:

$q_3 = \epsilon \cdot b + q_1 \cdot a \cdot a + q_3 \cdot a$

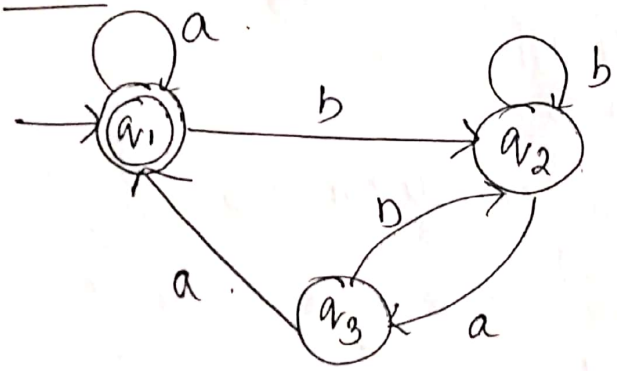
$= \epsilon \cdot b + \epsilon \cdot a \cdot a + q_3 \cdot a$

$= b + a \cdot a + q_3 \cdot a$

$= (b + (aa)) + q_3 \cdot a$

$\therefore q_3 = (b + aa) \cdot a^*$

(3) * DFA:



Equations:

$q_1 = \epsilon + q_1 \cdot a + q_3 \cdot a$

$q_2 = q_1 \cdot b + q_2 \cdot b + q_3 \cdot b$ ... (i)

$q_3 = q_2 \cdot a$ ... (ii)

... (iii)

On putting values of q_2 and q_3 in Eq(i), we get

$q_1 = \epsilon + q_1 \cdot a + q_2 \cdot a \cdot a$

$= \epsilon + q_1 \cdot a + (q_1 \cdot b + q_2 \cdot b + q_3 \cdot b) aa$

$= \epsilon + q_1 a + (q_1 \cdot b + q_2 \cdot b + q_2 \cdot a) aa$

On putting the value of q_3 in Eq(ii), we get:

$$q_2 = q_1 \cdot b + q_2 \cdot b + q_2 \cdot ab$$
$$= q_1 \cdot b + q_2 (b + ab)$$

By applying Arden's Theorem, we get.

$$q_2 = q_1 \cdot b (b+ab)^*$$

On putting the value of q_2 and q_3 in Eq(i), we get.

$$q_1 = \epsilon + q_1 \cdot a + q_2 \cdot aa$$
$$= \epsilon + q_1 \cdot a + (q_1 \cdot b (b+ab)^*) aa$$
$$= \epsilon + q_1 (a + b (b+ab)^* aa)$$

By applying Arden's Theorem, we get:

$$q_1 = \epsilon \cdot (a + b (b+ab)^* aa)^*$$
$$= (a + b (b+ab)^* aa)^*$$

The Pumping Lemma for Regular Languages:

Pumping Lemma is a powerful tool for proving certain language non-regular. It's also useful in the development of algorithm to answer certain questions concerning, finite automata, such as whether the language accepted by a given FA is finite or infinite

Statement: Let L be a regular language then there exist a constant n (which depends on L) such that for every string W in L, such that $|W| \geq n$, we can break W into three substrings, $W = xyz$, such that:

1) $y \neq \epsilon$

2) $|xy| \leq n$

3) For all $i \geq 0$, the string $xy^i z$ is also in L

That is we always find a non-empty string y not too far from the beginning of W can be "pumped": that is repeating y any number of times, keeps the resulting string in the language

Q1) $L = \{a^n b^n : n \geq 0\}$ check the regularity of the language.

⇒ Case 1:

If y is entirely made with a's, then we can go on inserting more no of a's.

x aaaaa z

It is not feasible as there should be same no of a's and b's. This case does not allows

Case 2:

If y is entirely made with b's, then we can go on inserting more no of b's

x bbbbb z

It is no feasible because no of b's will be more than a's but actually it needs to have same no of a's and b's. This case is not feasible

Case 3:

If y is entirely made of ab, we can pump more 'ab' in it

x ababab ... z

~~Because~~ It is not feasible as language suppo only one ab i.e ab, a(ab)b, aa(ab)bb.

Pumping lemma is not applicable for this language; hence it is not a regular language

Q 2) $L = \{ a^m b\, a^n : n \geq 0 \}$ Check the regularity of the language

⇒ Case 1:

If y is entirely made with b's, then we can pump more 'b' in it

u b b b b z

It is not feasible as the string supports only one 'b'.

Case 2:

If y is entirely made with a's, then we can pump more 'a' in it.

u a a a a z

We can consider any side a^n and we see that on both sides the no. of a's should be same. So it is not feasible.

Hence, Pumping lemma is not applicable for this language, it is not a regular language.

Q 3) $L = \{ a^n b^m : n \geq 0, m \geq 0 \}$ Check the regularity of the language.

⇒ Case 1:

When $n > m$

We can pump more a's on the left side

u a a a a z

It is feasible.

Case 2:

When $n < m$

We can pump more b's on the right side

u bbbb- z

It is feasible

Case 3:

When $n = m$

We can pump equal no. of a and b's

u abab. z

It is no feasible in case of $n = m = 0$, there will be no string i.e $y \in = \phi \in$, but as per rule Pumping Lemma is not applicable for this $y \neq \in$ language, hence it is not a regular language.

Q4) Proof that the parameters of the language consists of sets of balanced paranthesis is regular language cannot be

⇒ Case 1:

If y is made of left paranthesis, then we can pump more of 'y's.

u ((((. . . z

It is not feasible

Case 2:

If y is made of right paranthesis, then we can pump more of 'y's

u)))) . . . z

It is not feasible

Case 3:

If y is made of ( ) paranthesis, then we can pump more of 'y's

x ( )( )( ) ... z

It is feasible.

Pumping lemma is not applicable for this language. Hence it is not a regular language.

(5) $L = \{a^n b^n a\, b^{n+1} : n \geq 1\}$ Check the regularity of the language.

aabbabb
aaabbb
abb

⇒ Case 1:

If y is made of a, then we can pump more no of a's

x aaaa z

Either on a should be present or same no of a's and b's must be there. Hence it is not feasible.

Case 2:

If y is made of b, then we can pump more no of b's

x bbbb ... z

There should be same no of b's and a's on left side. Hence it is not feasible.

Case 3:

If y is made of ab, then we can pump more no of 'ab's

x ababab ... z two

There should be only ~~one~~ ab present. Hence it is

not feasible.

Case 4:

If y is made of ba, then we can pump more no. of 'ba's.

u bababa.. z

There can be only one 'ba's in the string. Hence not feasible.

It is not a regular language, as it does not follow pumping lemma.

Q6) $L = \{0^n 1^m 2^n : n, m \geq 0\}$

⇒ Case 1: $(m = 0)$. , $0^n 2^n$.

If y is made of 0's, then we can pump more no. of 0's.

u 0000.. z

There can be same no. of 0 and 2's, so it is no feasible.

If y is made of 2's, then we can pump more no. of 2's.

u 2222.. z

There can be only same no. of 0 and 2's, so it is not feasible.

Case 2: $n = m$

If $n = m = 0$, then $y = \epsilon$, but as per rule $y \neq \epsilon$. Hence not feasible.

case 3: $n = 0$. , 1^m .

If y is made of 1's, then we can pump more no. of 1's.

u 1111 .. z

There can be many no of 1's. It is feasible. It is not a regular language, as it does not follow pumping lemma

Q7) $L = \{ 0^n 1^{2n} : n \geq 1 \}$

⇒ Case 1:

If y is made up of 0's, then we can pump more no of 0's

u 0000 .. z

As, the no of 0's is more than 1's which is not feasible as no of 1's should be double of no of 0's. Hence, it is not feasible

Case 2:

If y is made up of 1's, then we can pump more no of 1's

u 1111 .. z

~~As the no of 0's should be less~~

The no of 1's is much more than the no of zeroes. Technically, the no of zeroes should be half the no of 1's. Hence, not feasible

Case 3:

If y is made up of 01's, then we can pump more no of 01's

u 010101 .. z

There should be double no of 1's than 0's. ~~Hence~~ But there are equal no of 0 and 1's. Hence not feasible

It is not a regular language, as it does not follow pumping lemma.

Q8) $\{0^n : n \text{ is a perfect square}\}$

$\Rightarrow$ Case 1: $n = 0$

If we consider $n = 0$, then $y = \epsilon$ but according to the rule ~~x~~ $y \neq \epsilon$, therefore it is not feasible.

Case 2: $n = 1$

If we consider $n = 1$, then $y \neq \epsilon$, therefore it is feasible.

Q9) $\{0^n : n \text{ is a perfect cube}\}$

Content Free Grammar

V → variable
T → terminal
$V \cap T = \phi$

~~$\alpha \to \beta$~~
~~$\alpha \in (V_t \cup V_n)$~~
~~$\beta \in (V_t \cup V_n)^*$~~

only one variable on left side

$\alpha \to \beta$

$\alpha \in V_n$ → Variable

$\beta \in (V_t \cup V_n)^*$

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

⇒ 1. r. e = a^*

Production rule for the Regular Expression:

1) $S \to aS$ rule 1
2) $S \to \varepsilon$ rule 2

Now if we want to derive a string "aaaaaa" we can start with start symbols.

1. S
2. aS
3. aaS rule 1
4. aaaS rule 1
5. aaaaS rule 1
6. aaaaaS rule 1
7. aaaaaaS rule 1
8. aaaaaaε rule 2
9. aaaaaa

Example: Construct a CFG for the regular expression $(0+1)^*$

⇒ The CFG can be given by:

1. Production rule (P):
2. $S \to 0S \mid 1S$
3. $S \to \varepsilon$

The rules are in combination of 0's and 1's with the start symbol.
Since $(0+1)^*$ indicates $\{\varepsilon, 0, 1, 01, 10, 00, 11 \ldots\}$.

In this set, ε is a string, so in the rule, we can set the rule $S \to \varepsilon$.

Example: Construct a CFG for a language,

$L = \{ wcw^R \mid \text{where } w \in (a,b)^* \}$

w^R = reverse.

$\Rightarrow$ wcw^R

$w \in \{a,b\}^*$

c

aca

abcba

abbcbba

bacab

$S \rightarrow aSa$
$S \rightarrow bSb$
$S \rightarrow c$ } ans.

$S \rightarrow aSa$
$\rightarrow abSba$
$\rightarrow abbSbba$
$\rightarrow abbcbba$

Q) $S \rightarrow bA/aB$
$A \rightarrow bAA/aS/a$
$B \rightarrow aBB/bS/b$

$\Rightarrow$

$S \rightarrow bA$
$\rightarrow bbAA$
$\rightarrow bbAaS$
$\rightarrow bbaaS$
$\rightarrow bbaaaB$
$\rightarrow bbaaaab$

$S \rightarrow aB$
$\rightarrow aaBB$
$\rightarrow aaBbS$
$\rightarrow aabbS$
$\rightarrow aabbaB$
$\rightarrow aabbab$

$S \rightarrow bA$
$\rightarrow ba$

$S \rightarrow bA$
$\rightarrow baS$
$\rightarrow babA$
$\rightarrow baba$

$S \rightarrow aB$
$\rightarrow abS$
$\rightarrow abaB$
$\rightarrow abab$

$a^m b^m$, $m \geq 1$

Equal no of a's and b's.

Only one terminal symbol is there in each production rule i.e called Backus Naur form of CFG.

* S → bA / aB
  A → bAA / aS / a
  B → aBB / bS / a

⇒ S → bA
→ bbAA
→ bbA.aS
→ bbAaabA
→ bbaabA
→ bbaaba

S → aB
→ aaBB
→ aabbS
→ aabbbA
→ aabbba

$a^n b^m$, $m \geq 0$, $n \geq 1$

S → bA.
→ bbAA
→ bbaSa
→ bbaaBa
→ bbaaaa.

S → aB
→ aaBB
→ aaaa
S → aB
→ aa

1) * $L = \{ a^n b^n : n \geq 1 \}$
2) * $L = \{ a^n b^{2n} : n \geq 1 \}$

⇒ 1) { ab, aabb, aaabbb, aaaabbbb, ... }

S → aSb
S → ab
S → ε

⇒ 2) { abb, aabbbb, aaabbbbbb, ... }

S → aSbb
S → abb
S → ε

S → aSC
S → aC
S → ε
C → bb

a a S b b
a a a S b b b

Normal form of Backus Naur Form:
S → aSC.
S → aC
S → ε.
C → b.

According to Backus Naur Form, only one terminal symbol is should be there hence we apply this method.

Derivation

A grammar is given and we have to derive set of strings. Two types of derivation exists:

(i) leftmost derivation
(ii) rightmost derivation

Derivation is a sequence of production. It is used to get the input strings based on production rule.

During parsing, we have to take two decisions.

Leftmost derivation

In the leftmost derivation, the input is scanned and replaced with the production rule from left to right.

1. $E = E+E$
2. $E = E-E$
3. $E = a/b$

Input: $a-b+a$

The leftmost derivation is:

1. $E = E+E$ (R1)
2. $E = E-E+E$ (R_2)
3. $E = a-E+E$ (R_3)
4. $E = a-b+E$ (R_3)
5. $E = a-b+a$ (R_3)

Rightmost derivation

In rightmost derivation, the input is scanned and replaced with the production rule from right to left

1. $E = E+E$
2. $E = E-E$
3. $E = a/b$

Input : a - b + a

The rightmost derivation is :

1. E - E = E
2. E = E - E + E
3. E = E - E + a
4. E = E - b + a
5. E = a - b + a

Qu 1 :

1. S → AB / ε
2. A → aB
3. B → Sb

Derive abb

⇒

1. S → AB (R1)
2. S → aBB (R2)
3. S → aSbB (R3)
4. S → abB (S → ε)
5. S → abSb (R3)
6. S → abb (S → ε)

Leftmost

Rightmost

1. S → AB (R1)
2. S → aASb (R3)
3. S → Ab (S → ε)
4. S → aBb (R2)
5. S → aSbb (R3)
6. S → abb (S → ε)

Qu 2 :

1. S → aB / bA
2. A → a / aS / bAA
3. B → b / bS / aBB

Derive aabbabba

⇒ Leftmost

S → aB
S → aaBB
S → aaaBBB

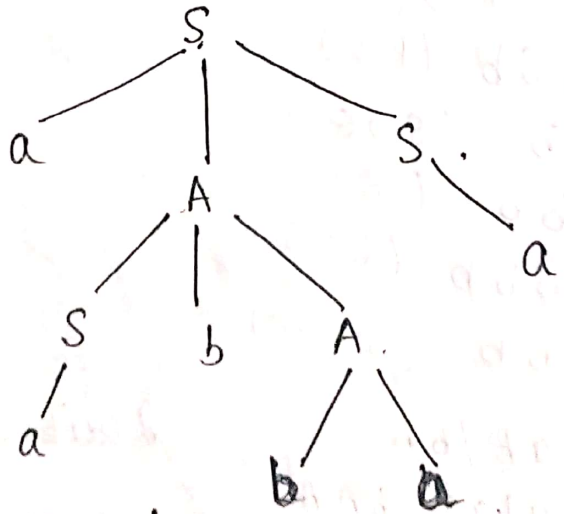
S → aabB
S → aab

S → aB (S → aB)
S → abS
S → aaBB (B → aBB)
S → aabB (B → b)
S → aabbS (B → bS)
S → aabbaB (S → aB)
S → aabbabS (B → bS)
S → aabbabbA (S → bA)
S → aabbabba (A → a)

S → aB
S → aaS
S → aabA
S → aabbAA
S → aabbaA
S → aabbabAA
S → aabbabaAA
S → aabbabaaA
S → aabbabaaa
S → aabbabAA

Q1) S → aAS / a Derive aabbaa
A → SbA / SS / ba

⇒ S → aAS (S → aAS)
S → aSbAS (A → SbA)
S → aabAS (S → a)
S → aabbaS (A → ba)
S → aabbaa (S → a)



Derivation Tree / Parse Tree for the string aabbaa.

* Recursively enumerable language

$A \to A\alpha$

This type of problem is known as left recursive problem

Q) $S \to aSb / SS / \epsilon$

Derive the string aabb also draw the DT.

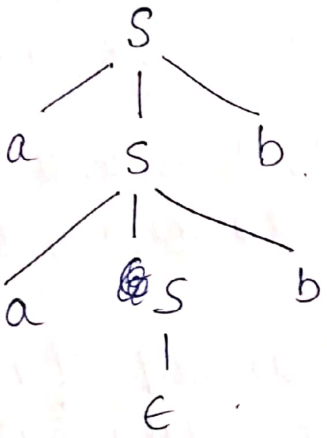
$(S \to aSb)$

$\Rightarrow$ $S \to aSb$

$S \to aaSbb \quad (S \to aSb)$

$S \to aabb \quad (S \to \epsilon)$

~~$S \to aabb \quad (S \to \epsilon)$~~



Parse Tree 1

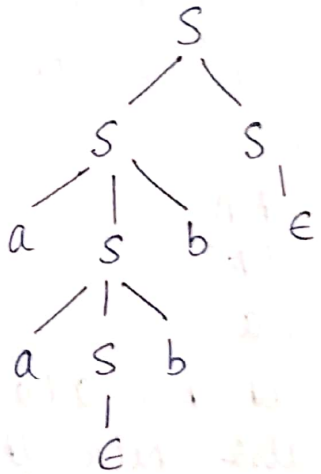
$S \to SS \quad (S \to SS)$

$S \to aSbS \quad (S \to aSb)$

$S \to aaSbbS \quad (S \to aSb)$

$S \to aabbS \quad (S \to \epsilon)$

$S \to aabb \quad (S \to \epsilon)$



Parse Tree 2

This type of grammar is called Ambiguous grammar. If string that produces two types of parse trees from the grammar is called Ambiguous grammar.

We have to remove the ambiguity of the grammar. so that it produces only 1 parse tree.

Date: 21/11/

* $E \to I$
  $E \to E+E$
  $E \to E*E$
  $E \to (E)$
  $I \to \epsilon \mid 0 \mid 1 \mid 2 \mid \ldots \mid 9$

Derive $3*2+5$

$\Rightarrow$ (i)
$E \to E*E \ (R_3)$
$E \to E*E+E \ (R_2)$
$E \to I*E+E \ (R_1)$
$E \to I*I+E \ (R_1)$
$E \to I*I+I \ (R_1)$
$E \to 3*I+I \ (R_5)$
$E \to 3*2+I \ (R_5)$
$E \to 3*2+5 \ (R_5)$

(ii)
$E \to E+E \ (R_2)$
$E \to E*E+E \ (R_3)$
$E \to I*E+E \ (R_1)$
$E \to I*I+E \ (R_1)$
$E \to I*I+I \ (R_1)$
$E \to 3*I+I \ (R_5)$
$E \to 3*2+I \ (R_5)$
$E \to 3*2+5 \ (R_5)$

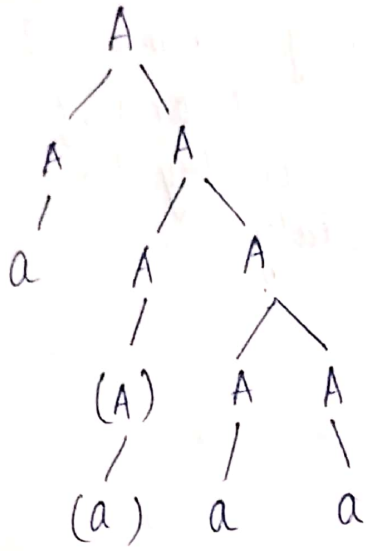
* If we give higher priority to the *, then second one is not supported. It is ambiguous one.

Ambiguity here removed with the help of higher precedence of the * as (*) has higher priority than (+).

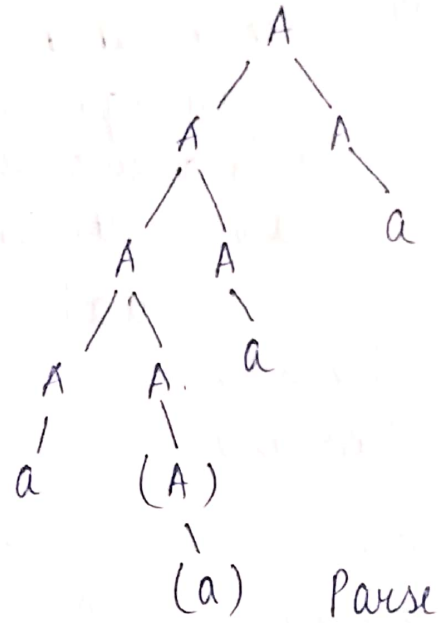
* $A \to AA$
  $A \to (A)$
  $A \to a$

Check whether a(a)aa string is produced or not and also check the ambiguity.

$\Rightarrow$ $A \to AA \ (A \to AA)$
$A \to AAA \ (A \to AA)$
$A \to AAAA \ (A \to AA)$
$A \to A(A)AA \ (A \to (A))$
$A \to a(A)AA \ (A \to a)$
$A \to a(a)AA \ (A \to a)$
$A \to a(a)aA \ (A \to a)$
$A \to a(a)aa \ (A \to a)$



Parse Tree 1



Parse Tree 2

We have to change the grammar that has the ability to produce the same string

Sol:

$A \to AA$	$A \to AaA$
$A \to A\,(A)$	$A \to A(A)$
$A \to a$	$A \to a$

* Left Associative operators :-

$a+b$, $a-b$, $a*b$, a/b

* Right Associative operators :

$ab^{\wedge} \to a^{1}b^{\wedge} \to a^{b}$

To convert the ambiguous grammer to the unambiguous grammar, there are two rules -

(i) If the left associative operators $(+, -, /, *)$ are used in the production rule, then apply the left recursion in the production rules. $X \to Xa$

(ii) If the right associative operators $(\wedge)$ are used in the production rule then apply the right recursion in the production rule.

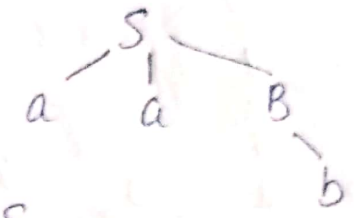
$X \to aX$

* $S \to AB / aaB$
$A \to a / Aa$
$B \to b$

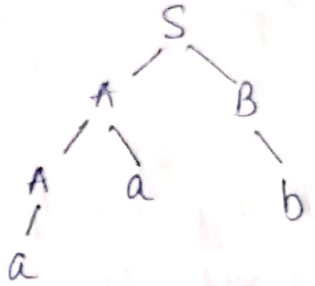
Check whether the grammar is ambiguous or not and if it is ambiguous, then try to remove the ambiguity

aab

$\Rightarrow$ $S \to aaB$ $(S \to aaB)$
$S \to aab$ $(B \to b)$



$S \to AB$ $(S \to AB)$
$S \to AaB$ $(A \to Aa)$
$S \to aaB$ $(A \to a)$
$S \to aab$ $(B \to b)$



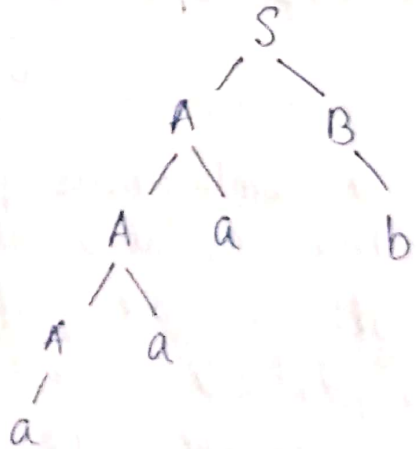
It is ambiguous

$S \to AB$
$A \to Aa / a$
$B \to b$

aaab

By removing the aaB string, we remove ambiguity.

$S \to AB$ $(S \to AB)$
$S \to AaB$ $(A \to Aa)$
$S \to AaaB$ $(A \to Aa)$
$S \to aaaB$ $(A \to a)$
$S \to aaab$ $(B \to b)$



Q1) $S \to ABA$
$A \to aA / \epsilon$
$B \to bB / \epsilon$

any strings
aa

2) $E \to E+E$
$E \to E*E$
$E \to id$

Check id+id*id

aaa

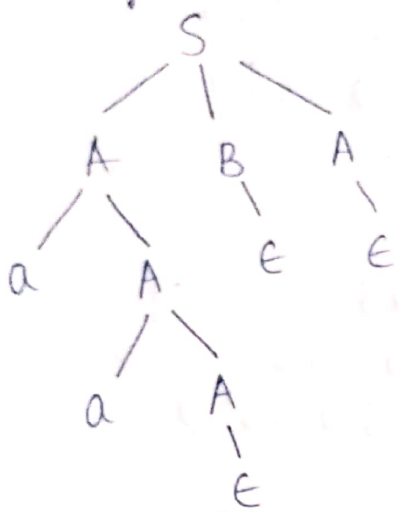
3) $S \to S+S$
$S \to S*S$
$S \to S^S$
$S \to a$

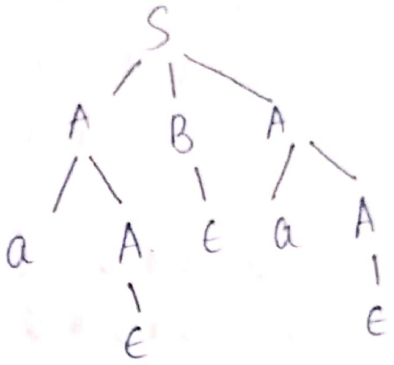
Check the ambiguity.

Q1) S → ABA (S → ABA)
S → aABA (A → aA)
S → aaABA (A → aA)
S → aaBA (A → ∈)
S → aaA (B → ∈)
S → aa (A → ∈)

S → ABA (S → ABA)
S → aABA (A → aA)
S → aBA (A → ∈)
S → aA (B → ∈)
S → aaA (A → aA)
S → aa (A → ∈)

It is ambiguous





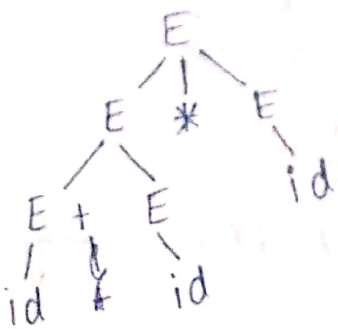
S → AB
A → aA / ∈
B → bB / ∈
} Non-ambiguous.

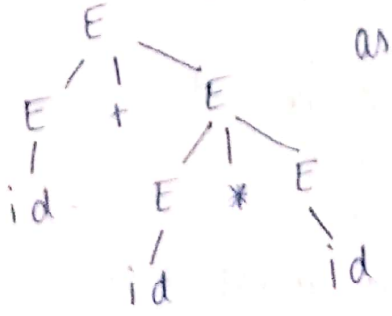
S → aAB (A → aA)
S → aaAB (A → aA)
S → aaB (A → ∈)
S → aa (B → ∈)

Q2) E → E * E (E → E * E)
E → E + E * E (E → E + E)
E → id + E * E (E → id)
E → id + id * E (E → id)
E → id + id * id (E → id)

E → E + E (E → E + E)
E → E + E * E (E → E * E)
E → id + E * E (E → id)
E → id + id * E (E → id)
E → id + id * id (E → id)

↘ Not supported as (*) priority is more than +.





(3) $S \to S+S$ ($S \to S+S$)

$S \to S+S*S$ ($S \to S*S$)

$S \to S+S*S+S$ ($S \to S+S$)

$a+a*a^a+a$

$S \to S+S*SS\hat{}+S$ ($S \to SS\hat{}$)

(1) $S \to a+S*SS\hat{}+S$ ($S \to a$)

$S \to a+a*SS\hat{}+S$ ($S \to a$)

$S \to a+a*aS\hat{}+S$ ($S \to a$)

$S \to a+a*aa\hat{}+S$ ($S \to a$)

$S \to a+a*aa\hat{}+a$ ($S \to a$)

(2) $S \to S*S$ ($S \to S*S$)

$S \to S+S*S$ ($S \to S+S$)

$S \to S+S*S+S$ ($S \to S+S$)

$S \to S+S*SS\hat{}+S$ ($S \to SS\hat{}$)

~~$S \to S+S$~~

$S \to a+S*SS\hat{}+S$ ($S \to a$)

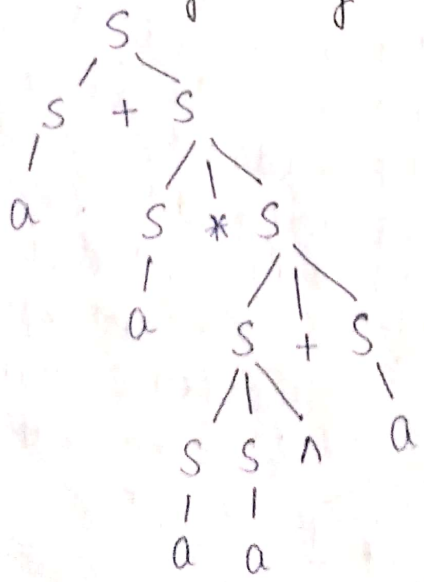
$S \to a+a*SS\hat{}+S$ ($S \to a$)

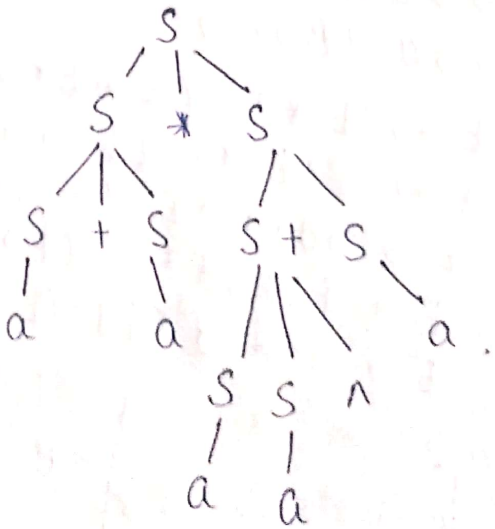
$S \to a+a*aS\hat{}+S$ ($S \to a$)

$S \to a+a*aa\hat{}+S$ ($S \to a$)

$S \to a+a*aa\hat{}+a$ ($S \to a$

Considering string $a+a*aa\hat{}+a$





* priority is more than + so we will consider only *.

Reduction of Context Free Grammar

→ We must eliminate useless symbols, those variables and terminals that do not appears in any derivation of a terminal string from start symbol.

→ We must eliminate ϵ productions those are of this form $X \to \epsilon$ for some variable X.

→ We must eliminate unique productions those are the form $X \to Y$.

Eg: $S \to AB / a$
$A \to b$ ⟶ [B does not contribute to the production so remove it]

→ $S \to a$
$A \to b$ [But as there is no A as it does not contribute]

Final: $S \to a$

Eg: ~~$S \to AB$~~ $S \to aB / bX$
$A \to BAd / bSX / a$
$B \to aSB / bBX$
$X \to SBD / aBa / ad$

→ A, X useful as they are producing terminals directly
S → may be may be not useful
↳ $S \to bX$ [Now Directly useful]

$B \to aSB / bBX$
↳ Both will be useless as B and in no way we cannot generate terminal symbol as B will be repeated.

Remove all B's:
$S \to bX$
$A \to bSX / a$
$X \to ad$

From S we cannot move to A, so remove A:

$S \to bX$
$X \to ad$ } Final.

Q) $A \to uyz / Xyzz$
$X \to Xz / uYz$
$Y \to yYy / Xz$
$Z \to Zy / z$

⇒ A, Z → Useful

We can remove X and Y as they are not producing any terminal symbol.

$A \to uyz$
~~Z~~ $\to Zy / z$

As A is not dependent on Z, we can remove Z.

$A \to uyz$ → Final

Q) $S \to aC / SB$
$A \to bSCa$
$B \to aSB / bBC$
$C \to aBC / ad$

⇒ C, S → Useful → A may also be useful [S is useful for aC]

In B, we cannot generate any terminal symbol, so we remove it

$S \to aC$ $C \to ad$
$A \to bSCa$

As S is not dependent on A, we can remove it

$S \to aC$

$C \to ad$

Removal of Unique Productions:

Q) $S \to AB$

$A \to a$

$B \to C / d$

$C \to D$

$D \to E$

$E \to a$

** $A \to B$ (Non terminal → Non terminal)

⇒ $B \to C$, $C \to D$, $D \to E$ } Unique production rules.

$S \to AB$	$S \to AB$	$S \to AB$	$S \to AB$
$A \to a$	$A \to a$	$A \to a$	$A \to a$
$B \to C / d$	$B \to C / d$	$B \to a / d$	$B \to a / d$
$C \to a$	$C \to D$	$D \to a$	Final.
$D \to a$	$D \to a$	$E \to a$	
$E \to a$	$E \to a$		

Assg

Q) $I \to a / b / Ia / Ib / I0 / I1$. HW

$F \to I / (E)$

$T \to F / T * F$

$E \to T / E + T$

⇒ Useless

~~I, F → Us~~ I, F → Usefull

T, E can be removed as they are not producing terminal symbol.

I → a / b / Ia / Ib / I0 / I1

F → I

As F I does not depend on F, so we remove it.

I → a / b / Ia / Ib / I0 / I1

Unique

I → a / b / Ia / Ib / I0 / I1

F → I / (E)

T → F / T*F

E → T / E+T

⇒ I → a / b / Ia / Ib / I0 / I1

F → a / b / Ia / Ib / I0 / I1 / (E)

T → I / (E) / T*F

E → F / T*F / E+T

Q) S → A / bb

A → B / b

B → S / a

HW

⇒ Remove B as S does not depend on B

S → A / bb

A → b

Unique

S → A

A → B

B → S

S → a / bb

A → a / b } Remove

B → a / bb }

OR

~~B → b~~

S → a / bb

Removal of ∈ productions

Date: 4/12/24

If any $N \to \epsilon$

↓

Nullable Non-terminal

Ex: $S \to aA$
$A \to b/\epsilon$

⇒ $S \to aA/a$
$A \to b$
} After removal of ∈

Ex: $S \to ABAC$
$A \to aA/\epsilon$
$B \to bB/\epsilon$
$C \to c$

⇒) When producing $A \to \epsilon$

$S \to ABAC$
$S \to BC$
$S \to bBC$
$S \to bC$

$S \to ABAC$
$\to BAC \ (A \to \epsilon)$

$S \to ABAC$
$\to ABC \ (A \to \epsilon)$

$S \to ABAC$
$\to ABC \ (A \to \epsilon)$
$\to BC \ (A \to \epsilon)$

$S \to BAC/ABC/BC/AAC$
$A \to aA/a$
$B \to bB/b$

When producing $B \to \epsilon$

$S \to ABAC$
$\to AAC \ (B \to \epsilon)$

$B \to bB$
$\to b \ (B \to \epsilon)$

Final:

~~$S \to BAC/ABC/AAC/$~~

When producing both
$A \to \epsilon$ and $B \to \epsilon$

$S \to ABAC$
$\to C \quad (A \to \epsilon, B \to \epsilon)$
$\to c \quad (C \to c)$

Final:

$S \to ABAC / BAC / ABC / BC / AAC / AC / C$

$A \to aA / a$

$B \to bB / b$

$C \to c$

Q) $S \to AB$

$A \to aAA / \epsilon$

$B \to bBB / \epsilon$

⇒ On putting $A \to \epsilon$

$S \to AB$

$\to B \quad (A \to \epsilon)$

$A \to aAA$

$\to aA \quad (A \to \epsilon)$

$A \to aAA$

$\to aA \quad (A \to \epsilon)$

$\to a \quad (A \to \epsilon)$

On putting $B \to \epsilon$

$S \to AB$

$\to A \quad (B \to \epsilon)$

$B \to bBB$

$\to bB \quad (B \to \epsilon)$

$B \to bBB$

$\to bB \quad (B \to \epsilon)$

$\to b \quad (B \to \epsilon)$

Final:

$S \to AB / A / B$

$A \to aAA / aA / a$

$B \to bBB / bB / b$

Q) $S \to a / Xb / aYa$

$X \to Y / \epsilon$

$Y \to b / X$

Eliminate ϵ production

⇒ On putting $X \to \epsilon$

$S \to Xb$

$\to b \quad (X \to \epsilon)$

$Y \to X$

$\to \epsilon$

$S \to aYa$

$\to aa \quad (Y \to \epsilon)$

$Y \to b / X$

$Y \to b / Y \quad (X \to Y)$

Final:

$S \rightarrow a/xb/aYa/b/aa$

$X \rightarrow Y$

$Y \rightarrow b/Y$

Push Down Automata

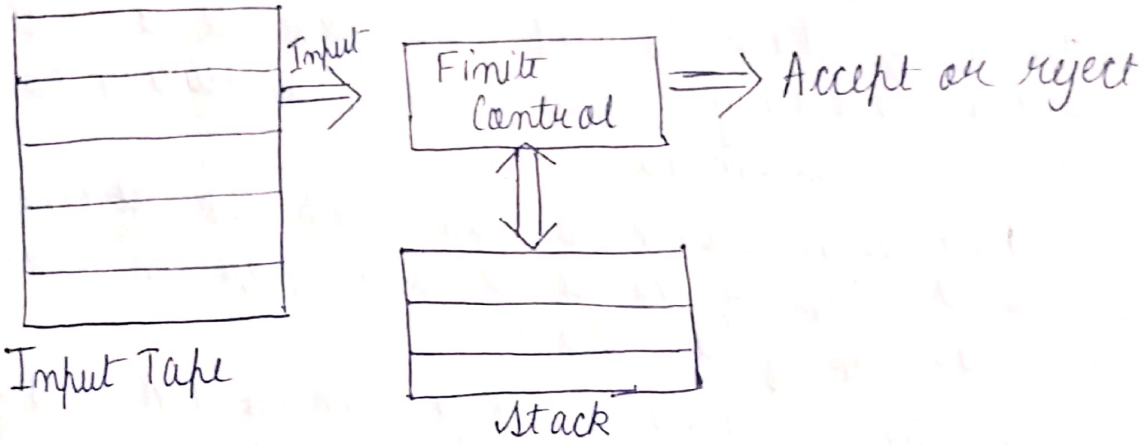


Fig: Push Down Automata

Formal definition of PDA

The PDA can be defined as a collection of 7 components:

Q: The finite set of states

Σ: The input set

Γ: A stack symbol which can be pushed and popped from the stack

q_0: The initial state

Z: a start ~~signal~~ symbol which is in Γ

F: a set of final states.

δ: Mapping function which is used for moving from current state to next state.

Pushdown automata (PDA) is a way to implem a context free grammar in the same way we design a DFA for a regular grammar. A DFA can remember a finite amount of information but a PDA can remember an infi amount of information.

Pushdown automata is simply a NFA augmented with an external stack memory. The addition of stack is used to provide a last in first out memory management capability to pushdown automata.

Pushdown automata can push an element on to the top of the stack and pop an element from the top of the stack.

A PDA is more powerful than FA. Any language which can be accepted by FA can also be accepted by pushdown automata but the reve is not true.

Diagram:

3 components in PDA:

1) Input Tape

It is divided into many cells / symbols. The inpu head is read only and may only move from left to right ~~with~~ by processing one symbol at a time.

2) Finite Control

The finite control has some pointer which point to the current symbol to be read.

3) Stack

In PDA the stack is used to store the item temporarily.

Instantaneous description of PDA (ID)

(P, b, T) ⇔ (q, w, α) — Used to describe configuration of PDA

$q \rightarrow$ Describes current state

$w \rightarrow$ describes remaining input/string

$\alpha \rightarrow$ Describe content of stack

Notation

Q: Define a pushdown automata for language $\{a^n b^n \mid n>0\}$

Sol:- $M =$ where $Q = \{q_0, q_1\}$ and $\Sigma = \{a, b\}$ and $F = \{A, Z\}$ and δ is given by -

$\delta(q_0, a, Z) = \{(q_0, AZ)\}$

$\delta(q_0, a, A) = \{(q_0, AA)\}$

$\delta(q_0, b, A) = \{(q_1, \epsilon)\}$

$\delta(q_1, b, A) = \{(q_1, \epsilon)\}$

$\delta(q_1, \epsilon, Z) = \{(q_1, \epsilon)\}$

P. St	Input	δ	Stack	Stack after move
q_0	aaabbb		Z	
q_0	aabbb	R2	AZ	q_0
q_0	abbb	R2	AAZ	q_0
q_0	bbb	R2	AAAZ	q_0
q_0	bbb	R3	AAZ	q_1
q_1	bb	R4	AZ	q_1
q_1	b			

P.St	input	S	Stack	State after move
q_0	aaabbb		Z	
q_0	aaabbb	R_1	AZ	q_0
q_0	aabbb	R_2	AAZ	q_0
q_0	abbb	R_2	AAAZ	q_0
q_0	bbb	R_3	AAZ	q_1
q_1	bb	R_4	AZ	q_1
q_1	b	R_4	Z	q_1
q_1	ϵ	R_5	ϵ	q_1

Date: 11/12/24

Q) Design a PDA which accepts the language $L = \{ w \in \{a, b\}^* \}$ which has equal no. of a's and b's.

1. $((S, \epsilon, \epsilon), (q, C))$
2. $((q, a, C), (q, aC))$
3. $((q, a, a), (q, aa))$
4. $((q, a, b), (q, \epsilon))$
5. $((q, b, C), (q, bC))$
6. $((q, b, b), (q, bb))$
7. $((q, b, a), (q, \epsilon))$
8. $((q, \epsilon, C), (f, \epsilon))$

Input string: abbbaaba

⇒

P.St	input	S	Stack	stack after move
S	abbbaaba	R_1	C	
q	abbbaaba	R_2	aC	q
q	bbbaaba	R_7	C	q
q	bbaaba	R_5	bC	q
q	baaba	R_6	bbC	q
q	aaba	R_4	cbbC	

S.No	Present State	Unread i/p	Stack	Tr
1.	S	abbbaaba	∈	–
2.	q	abbbaaba	C	1
3.	q	bbbaaba	aC	2
4.	q	bbaaba	C	7
5.	q	baaba	bC	5
6.	q	aaba	bbC	6
7	q	aba	bC	4
8.	q	ba	C	4
9.	q	a	bC	5
10.	q	∈	C	4
11	f	∈	∈	8

Remove the stack symbol C and design the same automata.

OR

× Wrong format

P.St	input	δ	Stack	State after move
S	abbbaaba		∈	
S	abbbaaba	R1	C	q
q	abbbaaba	R2	AC	q
q	bbbaaba	R7	C	q
q	bbaaba	R5	BC	q
q	baaba	R6	BBC	q
q	aaba	R4	BC	q
q	aba	R4	C	q
q	ba	R5	BC	q
q	a	R4	C	q
q	∈	R8	∈	f

Q) Design the same automata without the stack symbol C.

Q) Design a PDA which accepts the language $L = \{ w \in \{a,b\}^* \}$ w has equal no. of a's and b's,

1. $((S, a, \epsilon), (q, a))$
2. $((S, b, \epsilon), (q, b))$
3. $((q, a, a), (q, aa))$
4. $((q, b, b), (q, bb))$
5. $((q, a, b), (q, \epsilon))$
6. $((q, b, a), (q, \epsilon))$
7. $((q, \epsilon, \epsilon), (f, \epsilon))$

⇒

S.No	State	Unread i/p	Stack	Transition
1.	S	abbbaaba	ϵ	—
2.	q	abbbaaba	a	1
3.	q	bbbaaba	aa	3
4.	q	bbaaba	a	6
5.	q	baaba	ϵ	6 → Stoping at this positi
6.	q	aaba	b	2
7.	q	aba	ϵ	5
8.	q	ba	a	1
9.	q	a	ϵ	6
10.	q	ϵ	a	1
11.	q	ϵ		

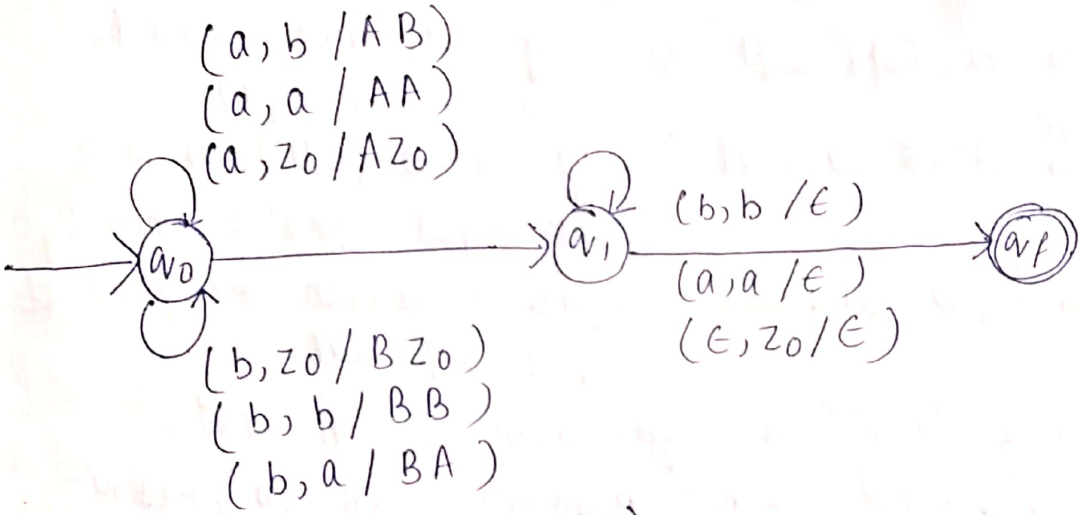
Q) $L = \{ ww^R \mid w \in \{a,b\}^+ \}$

[∵ $w \to abb$, $w^R \to bba$]

Q) $L = \{ wc\,w^R \mid w \in \{a,b\}^+ \}$

⇒ Ex: a b b c b b a

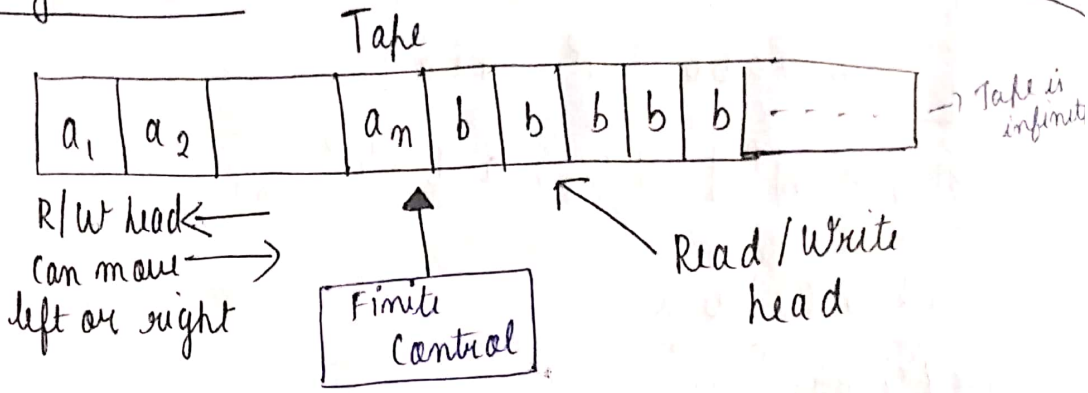
w w^R



1) $((q_0, a, Z_0), (q_0, AZ_0))$
2) $((q_0, b, Z_0), (q_0, BZ_0))$
3) $((q_0, a, A), (q_0, AA))$
4) $((q_0, b, B), (q_0, BB))$
5) $((q_0, a, B), (q_0, AB))$
6) $((q_0, b, A), (q_0, BA))$
7) $((q_0, c, A), (q_1, A))$ } ×
8) $((q_0, c, B), (q_1, B))$
9) $((q_1, b, B), (q_1, \epsilon))$
10) $((q_1, a, A), (q_1, \epsilon))$
11) $((q_1, \epsilon, Z), (q_2, \epsilon))$

Turing Machine

Date: 18/12/



A TM is expressed as a 7 tuple $(Q, T, B, \Sigma, \delta, q_0, F)$ where:

→ Q is a finite set of states

→ T is the tape alphabet (symbols which can be written on tape)

→ B is blank symbol (every cell is filled with B except input alphabet initially)

→ Σ is input alphabet (symbols which are part of input alphabet)

→ δ is a Transition function which maps $Q \times T \rightarrow Q \times T \times \{L, R\}$ Depending on its present state and present tape alphabet (pointed by head pointer), it will move to new state, changes the tape symbol (may or may not) and the head pointer will move in either right or left direction.

A Turing Machine consists of a tape of infinite length on which read / write operations can be performed. The tape consists of infinite cells on which each cell either contains i/p symbol or a special symbol called Blank. It also contains a head pointer which points, to cell currently being read and it can move in both the direction.

and play in the field of computer science.

In the content of automata theory and TOC, the turing machines are used to study the properties of algorithm and used to determine what problems or what type of problems can or cannot be solved by the computers. They provide a way to model the behaviour of algorithms and to analyze their computational complexity.

$\delta : Q \times T \to Q \times T \times \{L, R\}$

↳ It can be in any state Q and consuming T tape symbol and will move to another state Q or remain in same state with another T tape symbol which can move in either left or right direction.

* Design a TM for 2's complement
* Design a TM which will recognize strings in format $\{ 0^n 1^m 2^n , n \geq 1 \}$
* $\{ a^n b^{n+1} ; n \geq 1 \}$
* Discuss about P and NP problem. Give examples of NP complete problem and NP hard problem

* Let us construct a Turing Machine for $L = \{0^n 1^n$

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

$\Gamma = \{0, 1, X, Y, B\}$ while B represents blank

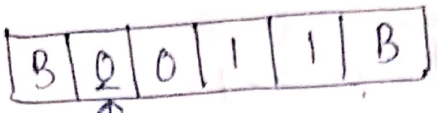
$\Sigma = \{0, 1\}$

$F = \{q_3\}$

Transition function δ is given in Table 1 as

	0	1	X	Y	B
q_0	(q_1, X, R)			(q_3, Y, R)	
q_1	$(q_1, 0, R)$	(q_2, Y, L)		(q_1, Y, R)	
q_2	$(q_2, 0, L)$		(q_0, X, R)	(q_2, Y, L)	
q_3				(q_3, Y, R)	Halt

* Suppose, machine contains the string of pattern 0011 and initially head points to zero.



q_0

$\delta(q_0, 0) \rightarrow (q_1, X, R)$

$\delta(q_1, 0) \rightarrow (q_1, 0, R)$

$\delta(q_1, 1) \rightarrow (q_2, Y, L)$

$\delta(q_2, 0) \rightarrow (q_2, 0, L)$

$\delta(q_2, X) \rightarrow (q_0, X, R)$

$\delta(q_0, \cancel{0}) \rightarrow (q_1, X, R)$

$\delta(q_1, Y) \rightarrow (q_1, Y, R)$

$\delta(q_1, 1) \rightarrow (q_2, Y, L)$

$\delta(q_2, Y) \rightarrow (q_2, Y, L)$

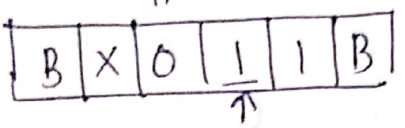
$\delta(q_2, X) \rightarrow (q_0, X, R)$

$\delta(q_0, Y) \rightarrow (q_3, Y, R)$

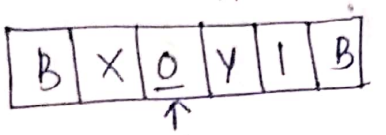
$\delta(q_3, Y) \rightarrow (q_3, Y, R)$

$\delta(q_3, B) \Rightarrow$ Halt

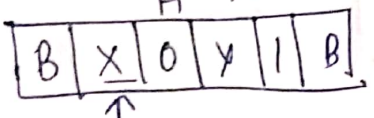
q_1



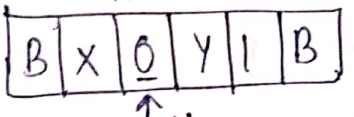
q_1



q_2

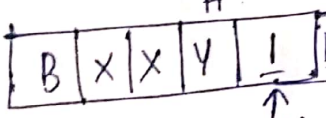


q_2

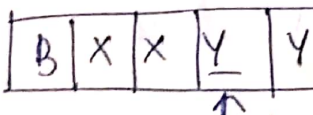


q_0

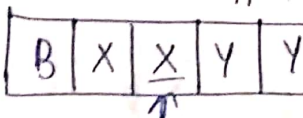
q_1



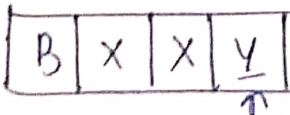
q_1



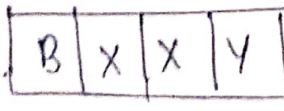
q_2



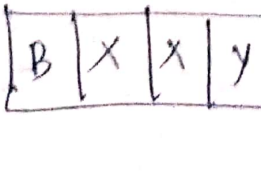
q_2



q_0



q_3



q_3 Halt