THEORY OF AUTOMATA AND FORMAL LANGUAGE (BCS-402)

AKTU New Session (2024-25)

SIMPLE AND EASY EXPLANATION + HANDWRITTEN NOTES PDF

ALL

UNIT-1

TOPIC

Basic Concepts and Automata Theory: Introduction to Theory of Computation- Automata, Computability and Complexity, Alphabet, Symbol, String, Formal Languages, Deterministic Finite Automaton (DFA)- Definition, Representation, Acceptability of a String and Language, Non Deterministic Finite Automaton (NFA), Equivalence of DFA and NFA, NFA with ε-Transition, Equivalence of NFA's with and without ε-Transition, Finite Automata with output- Moore Machine, Mealy Machine, Equivalence of Moore and Mealy Machine, Minimization of Finite Automata.

Engineering Express

Unit	Theory of Automata and Formal Languages
I	**Basic Concepts and Automata Theory:** Introduction to Theory of Computation- Automata, Computability and Complexity, Alphabet, Symbol, String, Formal Languages, Deterministic Finite Automaton (DFA)- Definition, Representation, Acceptability of a String and Language, Non Deterministic Finite Automaton (NFA), Equivalence of DFA and NFA, NFA with ε-Transition, Equivalence of NFA's with and without ε-Transition, Finite Automata with output- Moore Machine, Mealy Machine, Equivalence of Moore and Mealy Machine, Minimization of Finite Automata.
II	**Regular Expressions and Languages:** Regular Expressions, Transition Graph, Kleen's Theorem, Finite Automata and Regular Expression- Arden's theorem, Algebraic Method Using Arden's Theorem, Regular and Non-Regular Languages- Closure properties of Regular Languages, Pigeonhole Principle, Pumping Lemma, Application of Pumping Lemma, Decidability- Decision properties, Finite Automata and Regular Languages
III	**Regular and Non-Regular Grammars**: Context Free Grammar(CFG)-Definition, Derivations, Languages, Derivation Trees and Ambiguity, Regular Grammars-Right Linear and Left Linear grammars, Conversion of FA into CFG and Regular grammar into FA, Simplification of CFG, Normal Forms- Chomsky Normal Form(CNF), Greibach Normal Form (GNF), Chomsky Hierarchy, Programming problems based on the properties of CFGs.
IV	**Push Down Automata and Properties of Context Free Languages**: Nondeterministic Pushdown Automata (NPDA)- Definition, Moves, A Language Accepted by NPDA, Deterministic Pushdown Automata(DPDA) and Deterministic Context free Languages(DCFL), Pushdown Automata for Context Free Languages, Context Free grammars for Pushdown Automata, Two stack Pushdown Automata, Pumping Lemma for CFL, Closure properties of CFL, Decision Problems of CFL, Programming problems based on the properties of CFLs

Basic concepts and automata theory.

TAFL= Automated theory is a subject which explains or tells the behavior of automatic machines mathematically.

Introduction to theory of computation

What is automata? And why we study it?

The word automata mean automatic or self-controlled. Like computer atm and any automatic machine. That works automatically.

or

In other words.

It is an area of CS that deals with the study of abstract machines as well as the computational problems that can be solved using these machines.

Block diagram for any automatic machine.

NOTE *Other names. Automated theory.
Theory of computation.
Theory of computer science.
And computer theory.

Why we study automata?

1. Allow us to think in an ordered manner about what machine do without going into hardware details.

2. Learning of languages and computational techniques. Or (mathematical ways)
3. Designing of theoretical models (concepts) for machines.

Computability Theory
It describes the problem is decidable or undecidable.
Complexity Theory
It describes the time and space for a decidable problem.

The basic terminologies of automated theory.

Symbol Alphabet String Language

1. Symbols
Symbol are an entity or individual objects, which can be any letter, alphabets or any picture.
For example= 1, a, b, #, @ etc.

2. Alphabets
It Is finite, non- empty sets of symbols.
It Is denoted by Sigma [Σ]
For example= Binary symbol: Σ = {0,1}.
All lower-case letters: Σ ={a, b, c..... z}.
Alphanumeric: Σ = {a-z, A-Z, 0-9}.

3. Strings
It is a finite collection of symbols selected from the alphabets (Σ).
A string can be empty, which is represented by (ε) Epsilon.
The string is denoted by Omega. (ω)
For Example= {a, b} then ω= {ab, aa, bb, ba.....}

NOTE: -

Length of string= The length of strength is denoted by $|\omega|$. The number of positions or number of symbols in a string is called length of string. Like

For Example= {a, b} then ω= {ab, aa, bb, ba.....}

Length of string $|\omega| = 2$

*Empty string= The empty strength is the string with zero occurrence of symbols. The empty string is represented by (ε).

For example= Σ^*= {ε, 0, 1, 01, 10, 000, 0000....}

{0} *= {ε, 0, 00, 000, 0000....}

4. Language

A language is a collection of appropriate string, a language which is formed over Σ can Be finite or infinite.

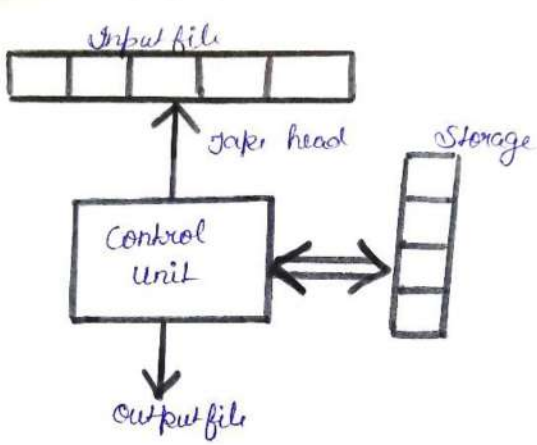
For examples= L1: {A set of strings of length, 2}

= {aa, bb, ab, b a}

FINITE AUTOMATA

What is Finite Automata?

Finite automata consist of finite memory called input file or input tape. A read only head and control unit. The input is provided on the file and control unit reads the symbol, one by one. The movement is decided by control unit. When input is finished, control unit decides the validity of machine. The input by acceptance or rejection. It does not Write anything which is in a limitation of this model. Nd the output that is display by output unit. The input tape is divided into cells,

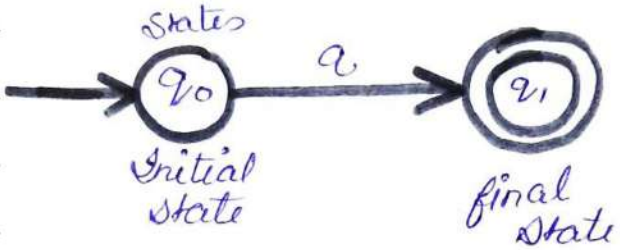


and each cell contain one symbol from the input alphabet signal.

The control unit reads the symbols one by one from the Input tape. And finite control the next configuration(state). The control unit cannot move backward or forward once the input symbol has been read. So finite automata can't remember its previous read symbol. This is one of the major limitations of finite automata, which show that it cannot be used in such computation where knowledge of previous input is necessary. Finite automata also known as FSM or FSA.

Representation of finite automata

1. Transition diagram.



2 Transition table.

Present State	Next State	
	a	b
q_0	q_1	
* q_1		

Types of finite automata.

There are two types of finite automata.

1. Deterministic finite automata.
2. Non deterministic finite automaton.

Deterministic Finite Automata

What is DFA?

1. It is deterministic behavior or Deterministic Finite Automata
2. It is also known as finite automata.

3. It followed two properties.
   1. There is only one next state for input a from state q_0
   2. There is transition from state q on every input symbol in Sigma.
4. It is a collection of five objects i.e. $M = [Q, \Sigma, \delta, q_0, F]$.

1. Q: Finite sets of states.
   $= \{q_1, q_2, q_3 \ldots\ldots q_n\}$.
2. Σ: Finite sets of alphabets
   = binary number, lower case letters, alphanumeric.
3. q_0: Initial states or starting state
4. δ : Transition function $\delta : Q \times \Sigma = Q$
5. F: Set of final state.

Language of DFA.

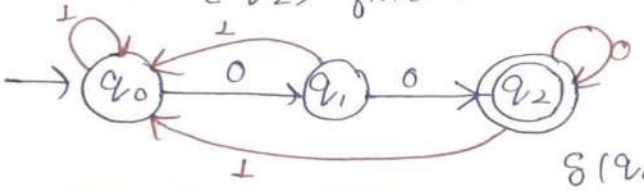
Language of DFA

$$L(M) = \{w \mid \delta(q_0, w) \text{ is in } F\}$$

QUESTION 1: Design a finite-automata that accepts set of strings such that every string end with 00 in alphabet. $\Sigma = \{0,1\}$

Solution =

Soln Let DFA, $M = (Q, \Sigma, \delta, q_0, F)$
$q_0 = \{q_0\}$ (initial state)
$\Sigma = \{0,1\}$ is given
$L = \{00, 100, 0100, 0110100 \ldots\ldots\}$
$Q = \{q_0, q_1, q_2\}$
$F = \{q_2\}$ final state



Present State	final State	
	a	b
→ q_0	q_1	q_0
q_1	q_2	q_0
* q_2	q_2	q_0

$\delta(q_0, 0) = q_1$
$\delta(q_0, 1) = q_0$
$\delta(q_1, 0) = q_2$
$\delta(q_1, 1) = q_0$
$\delta(q_2, 0) = q_2$
$\delta(q_2, 1) = q_0$

Ques Design DFA which accepts set of binary string divisible by 3

Remainder set of 3 = {0, 1, 2}

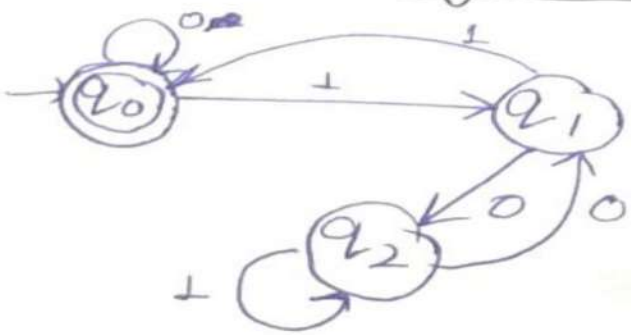
0 → q_0
1 → q_1
2 → q_2
} remainder set represent states

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

Formula used $Q \times \underset{(0,1)}{\Sigma} + \underset{(0,1)}{\Sigma}$

$\delta(q_0, 0) = 2 \times \text{Stat} + \Sigma = 2 \times 0 + 0 = 0 = q_0$
$\delta(q_0, 1) = 2 \times 0 + 1 = 1 = q_1$
$\delta(q_1, 0) = 2 \times 1 + 0 = 2 \quad q_2$
$\delta(q_1, 1) = 2 \times 1 + 1 = 3 \to 0 \; q_0$
$\delta(q_2, 0) = 2 \times 2 + 0 = 4 \to 1 = q_1$
$\delta(q_2, 1) = 2 \times 2 + 1 = 5 = 2 = q_2$

Transition diagram



★ q_0 is initial & final because if we take 0000 input then the q_0 state will repeat untill we get 0000 that is why

Questions for practice.

QUESTION 1: DFA of language containing substrings aba over. $\Sigma = \{a, b\}$.

QUESTION 2: Design A finite automata, which accept strings containing exactly four ones in every string over alphabet. $\Sigma = \{1, 2\}$.

#Non-deterministic finite Automaton

What is NFA?

The concept of NFA is exactly reverse of DFA The finite automata is called NFA. When it. Belongs many parts for a specific input from current state to next state.

1 It is a collection of five objects i.e. $M = [Q, \Sigma, \delta, q_0, F]$.

2 In NFA only δ is changed.

3 Properties or characteristics of NFA

a. There can be more than one next state for input a from state q_0

b. Not compulsory that all the states have to read all the input symbols.

LET US SEE ALL FIVE OBJECTS

1. Q: Finite sets of states.
   $= \{q_1, q_2, q_3 \ldots\ldots q_n\}$.

2. Σ: Finite sets of alphabets
   = binary number, lower case letters, alphanumeric.

3. q_0: Initial states or starting state

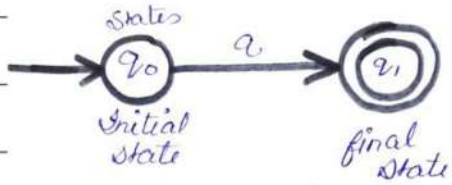
4. δ : Transition function $\quad \delta : Q \times \Sigma \rightarrow 2^Q$

   2^Q= POWER SET OF Q (subsets)

5. F: Set of final state

Language of NFA $\quad L(M) = \{w \mid \hat{\delta}(q_0, w) \cap F \neq \phi\}$

Difference between DFA and NFA.



DFA NFA

a. There is transition from state q on every input symbol in Sigma	Not compulsory that all the states have to consume all the input symbols.
b. Multiple choices are not available Corresponding to one input.	Multiple choice are available Corresponding to one input.
c. Designing and understanding is difficult.	Designing and understanding is easy.
d. Epsilon move is not allowed.	Epsilon movies allowed.
e. Digital computers are deterministic.	Nondeterministic features is not associated with digital computer.

#Questions for NFA

QUESTION 1. Design an NFA for the language L, which accept all the strings in which the 3rd symbol from right and is always 0 over. $\Sigma = \{0, 1\}$.

SOLUTION =

Sol^n $L = \{ 100, 101, 111 \ldots \}$

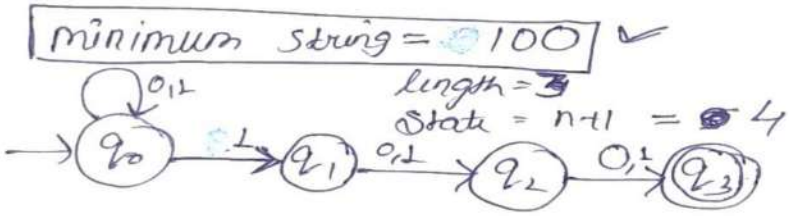
$\Sigma = 0, 1$

$W = \underline{0,1}\ \underline{1}\ \underline{0,1}\ \underline{0,1}$

(III rd place)

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

Permutation $\Rightarrow (0+1)^*\ 1\ \underline{0,1}\ \underline{0,1}$



Present state	final state	
	0	1
→ q_0	q_0, q_1	q_0
q_1	q_2	q_2
q_2	q_3	q_3
* q_3	—	—

QUESTION 2: Design NFA In which all strings are And with aba.

Σ = {a, b}.

SOLUTION=

Sol^n $\Sigma = a, b$

Condition = end with aba

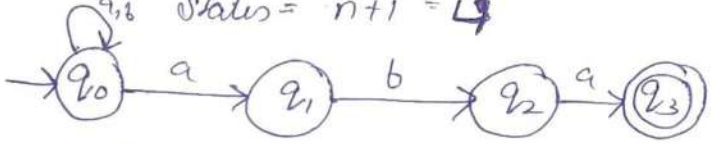
$L = \{ aba, aaba, baba, aaaba \ldots \}$

$q_0 \cdot q_0$

minimum string = aba

(n) length = 3

Status = n+1 = 4



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

Present state	final state	
	a	b
→ q_0	$q_0\ q_1$	q_0
q_1	—	q_2
q_2	q_3	—
* q_3	—	—

#QUESTIONS FOR PRACTICE

QUESTION 1: Design NFA In which all strings contain aba as substring $\Sigma = \{a, b\}$.

QUESTION 2: Design an NFA for the language L, which accept all the strings in which the 3rd symbol from right and is always a over. $\Sigma = \{a, b\}$.

#Conversion of equivalence DFA(MI) From given NFA(M)

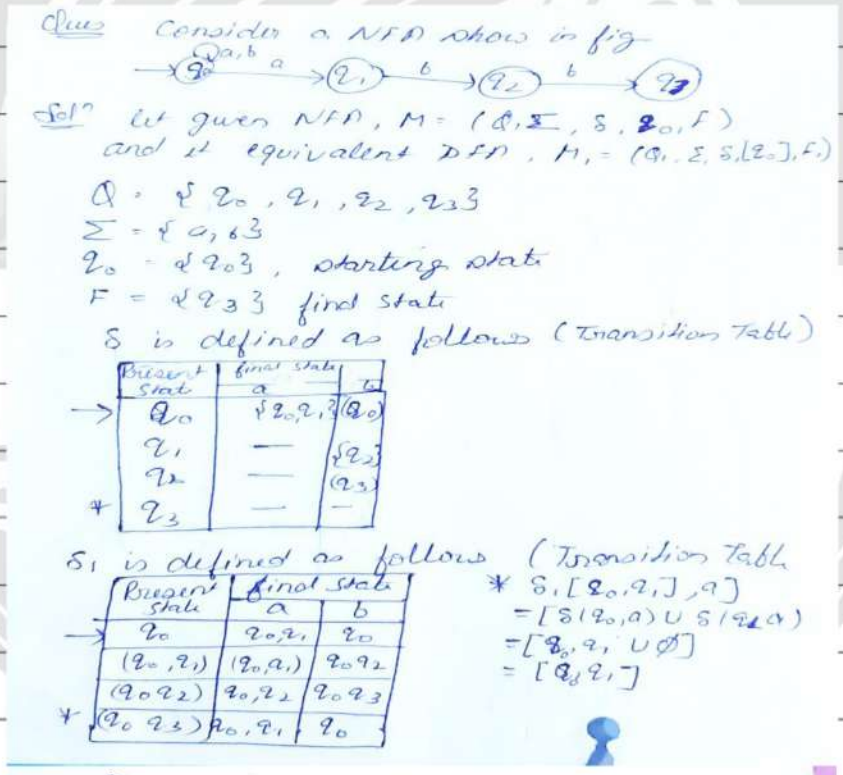
STEP 1 : Write all five objects of NFA [(M)= Q, Σ, δ, q_0, F].

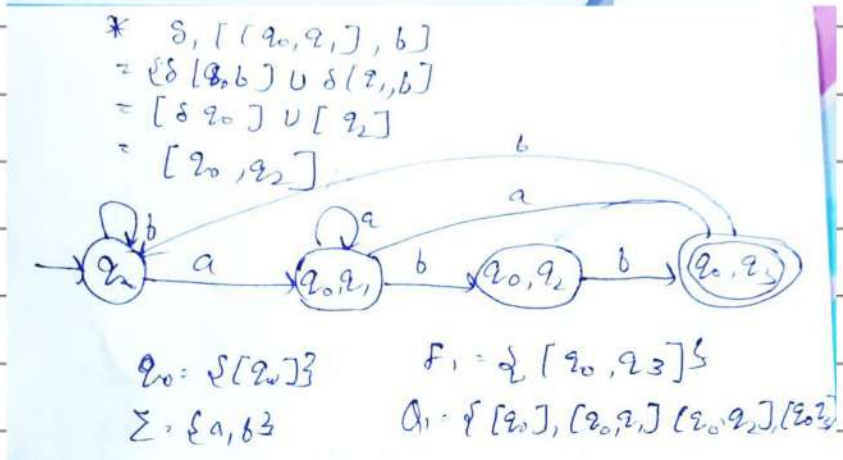
STEP 2 : Convert Transition diagram into Transition Table. For NFA

STEP 3 : Make transition table for DFA by finding δ values

STEP 4 : Last draw transion diagram for DFA by states

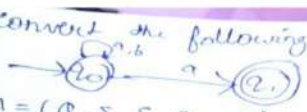
QUESTION 1.





QUESTION 2:

Convert the following into DFA (without null transition)



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

$Q = \{q_0, q_1\}$

$q_0 = q_0$

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

$F = \{q_1\}$

Now, Transition table

	Present state	final state	
		a	b
→	q_0	q_0, q_1	q_0
*	q_1	—	—

For DFA, we construct $M' = (Q', \Sigma', \delta', [q_0], F')$

$[q_0] = [q_0]$

$Q' = 2^Q = 2^2 = 4 = \{\phi, q_0, q_1, q_0q_1\}$

$\delta'[[q_0]a] = \delta(q_0, a) = [q_0, q_1]$

$\delta'[[q_0]b] = \delta(q_0, b) = [q_0]$

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

$= (q_0, q_1) \cup [\phi]$

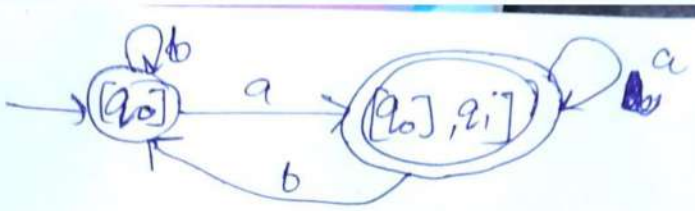
$= q_0, q_1$

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

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

$= [q_0) \cup \phi$

$= [q_0]$ as it is exist that means it converted to DFA



PRACTICE QUESTION

Ques Convert the following into DFA

2 QUESTIONS FOR REVISION

Design NFA

① Design NFA

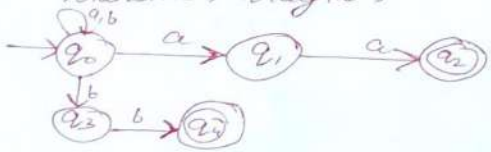
Ques! Design NFA in which strings over {a,b} ends with aa or bb.

L = {aa, bb, aaa, bbb, baa, abb ...}

w = — aa
— bb

q_0 = starting state

Transition diagram

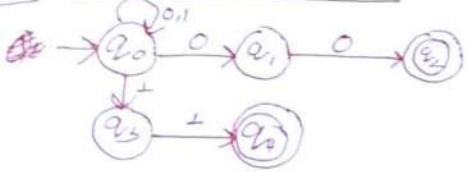


Transition Table

Present state	final state a	b
→ q_0	q_0, q_1	q_0, q_3
q_1	q_2	—
q_2	—	—
q_3	—	q_4
* q_4	—	—

Convert NFA to DFA

Ques NFA to DFA



Step 1 Convert Transition diagram into table

Present state	final state 0	1
→ q_0	q_0, q_1	$q_0 q_3$
q_1	q_2	—
* q_2	—	—
q_3	—	q_4
* q_4	—	—

Step 2 Convert NFA table to DFA Table

	Present state	final State	
		0	1
→	q_0	$\{q_0, q_1\}$	$\{q_0, q_2\}$
	$\{q_0, q_1\}$	$\{q_0, q_1, q_2\}$	$\{q_0, q_2\}$
	$\{q_0, q_2\}$	$\{q_0, q_1\}$	$\{q_0 q_2, q_4\}$
*	$\{q_0, q_1, q_2\}$	$\{q_0, q_1, q_2\}$	$\{q_0, q_2\}$
*	$\{q_0, q_2, q_4\}$	$\{q_0 q_1\}$	$\{q_0 q_1, q_2\}$

Step 3 Rename the states that found in DFA Table

$q_0 \rightarrow A$

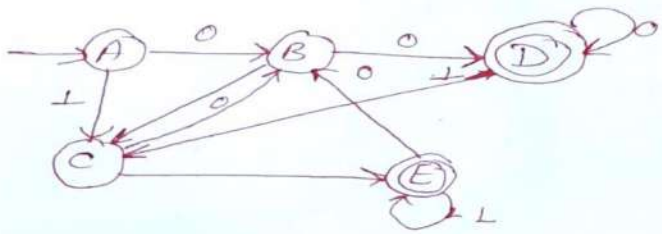
$q_0, q_1 \rightarrow B$

$q_0 \rightarrow q_2 \rightarrow C$

$q_0, q_2, q_2 \rightarrow D$

$q_0, q_2, q_4 \rightarrow E$

Step 4 Draw Transition diagram for DFA by these state.



#Convert NFA to DFA with Epsilon

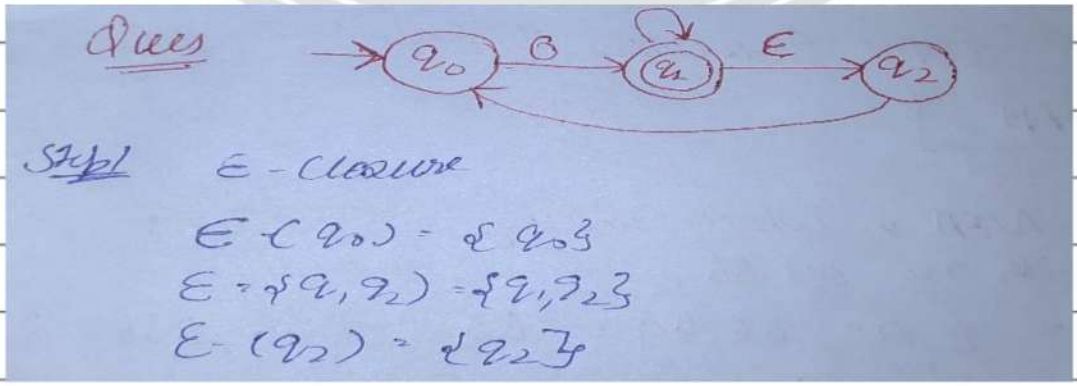
What is Epsilon closer?(ε) or (^)

The epsilon-closure is a set of all states which are. Reachable from state q on ε transition such that.or ^

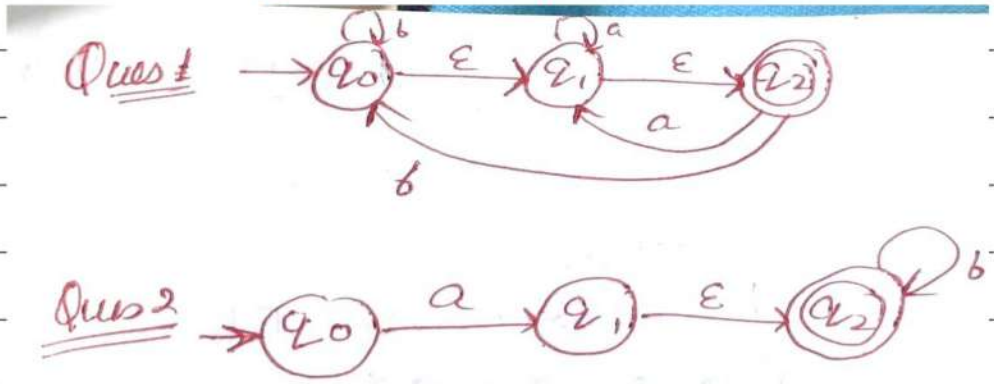
ε -closure (q)=q , where q. where q ε Q

Question. Find Eclosuer for the. Following nfa with epsilon

Solution=



Two questions for practice



Conversion from with null NFA to without null NFA

In this method, we try to remove all the E transition from given nfa. The method will be.

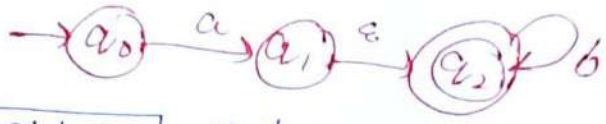
STEP 1= Find out all the E transitions from each state, from Q that will be called as E closuer.{qi},where qi=Q

STEP 2= Then δ, transition can be obtained. The δ, transition means an E closuer on δ, moves.

STEP 3= Step two is repeated for each input symbol and for each state of given nfa. Whenever a new state is found.

STEP 4= Using the resultant state, the transition table for equivalent and dfa, without, ε can be built

Q. Construct the given NFA with null (ε) to NFA without ε



Soln **Step 1** find out ε-Closure of all states $\{q_0, q_1, q_2\}$

ε-Closure $(q_0) = \{q_0\}$

ε-Closure $(q_1) = \{q_1, q_2\}$

ε-Closure $(q_2) = \{q_2\}$

Step 2 Now find δ' transition on each symbol is obtained

Formula: $\delta' = \varepsilon\text{-Closure}[\delta(\varepsilon\text{-Closure}(q), \Sigma)]$

$\delta'(q_0, a) = \varepsilon\text{-Closure}[\delta(\varepsilon\text{-Closure}(q_0), a)]$

$= \varepsilon\text{-Closure}[\delta(q_0, a)]$

$= \varepsilon\text{-Closure}[q_1]$

$= \{q_1, q_2\}$

$\delta'(q_0, b) = \varepsilon\text{-Closure}[\delta(\varepsilon\text{-Closure}(q_0), b)]$

$= \varepsilon\text{-Closure}[\delta(q_0, b)]$

$= \varepsilon\text{-Closure}(\emptyset)$

$= \{\emptyset\}$

$\delta'(q_1, a) = \varepsilon\text{-Closure}[\delta[\varepsilon\text{-Closure}(q_1), a)]$

$= \varepsilon\text{-Closure}[\delta(q_1, a)]$

$= \varepsilon\text{-Closure}[\delta(q_1, a) \cup \delta(q_2, a)]$

$= \varepsilon\text{-Closure}[\emptyset \cup \emptyset]$

$= \{\emptyset\}$

$\delta'(q_1, b) = \varepsilon\text{-Closure}[\delta(\varepsilon\text{-Closure}(q_1), b)]$

$= \varepsilon\text{-Closure}[\delta(q_1, q_2) b)]$

$= \varepsilon\text{-Closure}[\delta(q_1, b) \cup (q_2, b)]$

$= \varepsilon\text{-Closure}[\emptyset \cup q_2]$

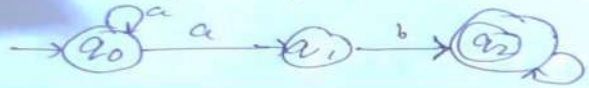
$= q_2$

Step 3

Present state	final state	
	a	b
→ q_0	{a, q3}	∅
q_1	∅	{q2}
* q_2	∅	{q2}

Step 4

diagram of transition



Conversion of NFA with ε to DFA

(i) Considered $M = (Q, \Sigma, \delta, q_0, F)$ is a NFA with ε. We have to convert this NFA with ε to equivalent DFA denoted by $M' = (Q', \Sigma, \delta', q_0', F')$ then obtain

ε Closure $(q_0) = \{p_1, p_2, p_3 \dots p_n\}$ then $[p_1, p_2, p_3 \dots p_n]$ become start state of state $\in Q_1$

(ii) We will obtain δ transition on $[p_1, p_2, p_3 \dots p_n]$ for each input

$\delta'([p_1, p_2, p_3 \dots p_n], a) = \varepsilon\text{-Closure}(\delta(p_1, a) \cup \delta(p_2, a) \cup \delta(p_3, a) \cup \delta(p_n, a))$

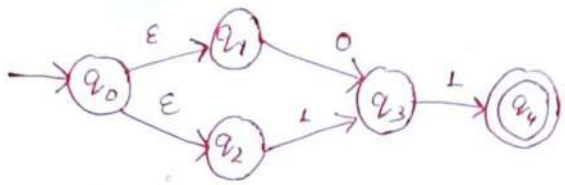
$\Rightarrow \bigcup_{i=1}^{n} \varepsilon\text{-Closure}(q_i = a)$

where a is input $\in \Sigma$

(iii) The state obtained $[q_1, q_2, q_3 \dots q_n] \in Q$

The states containing final state in p_i is a final state in DFA

Ques Convert the given NFA with ε to its equivalent DFA



Soln **Step 1** find ε-closure of all states $\{q_0, q_1, q_2, q_3, q_4\}$

ε-closure$(q_0) = \{q_0, q_1, q_2\}$

ε-closure$(q_1) = \{q_1\}$

ε-closure$(q_2) = \{q_2\}$

ε-closure$(q_3) = \{q_3\}$

ε-closure$(q_4) = \{q_4\}$

formula

$[\delta' = \varepsilon\text{-closure}[\delta(\varepsilon\text{-closure}(q), \Sigma]$

Step 2 find new transition states by ε closures.

Now let ε-closure $(q_0) = \{q_0, q_1, q_2\}$ be state A

$\delta'(A,0) = \varepsilon$-closure $[\delta(q_0, q_1, q_2), 0]$

$= \varepsilon$-closure $[\delta(q_0,0) \cup \delta(q_1,0) \cup \delta(q_2,0)]$

$= \varepsilon$-closure $[\phi \cup q_3 \cup \phi]$

$= \varepsilon$-closure (q_3)

$= \{q_3\}$ be state B (New state)

$\delta'(A,1) = \varepsilon$-closure $[\delta(q_0, q_1, q_2), 1]$

$= \varepsilon$-closure $[\delta(q_0,1) \cup \delta(q_1,1) \cup \delta(q_2,1)]$

$= \varepsilon$-closure $[\phi \cup \phi \cup q_3]$

$= \{q_3\}$ be state B (New state as same previous)

$\delta'(B,0) = \varepsilon$-closure $[\delta(q_3,0)]$

$= \varepsilon$-closure $[\phi]$

$= [\phi]$

$\delta'(B,1) = \varepsilon\text{-Closure}[\delta(q_3,1)]$

$= \varepsilon\text{-Closure}[q_4]$

$= \{q_4\}$ be state C (New state)

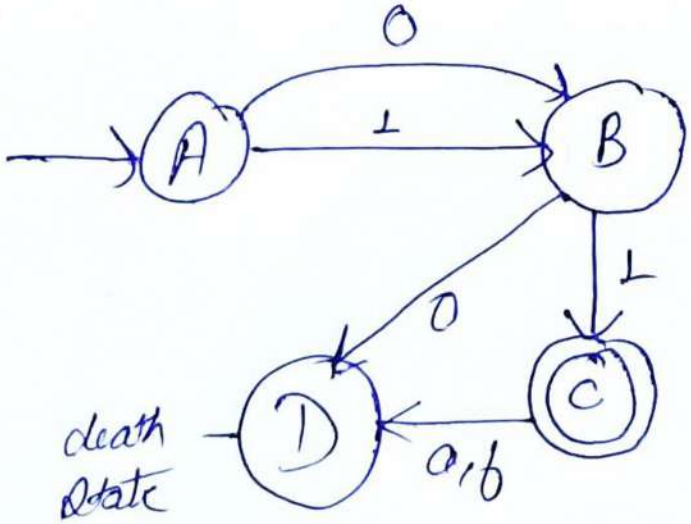
$\delta'(C,0) = \emptyset$

$\delta'(C,1) = \emptyset$

Step 3 Draw transition table for DFA

	Present State	final State	
		0	1
→	A	B	B
	B	∅	C
*	C	∅	∅

Step 4 Draw transition diagram for DFA



Limitations and Applications of FA

Applications

1. For the designing Of compilers and interpreters for the Programming language.
2. Used in text Editors.
3. Used in robotics. To design and control the behavior of the robots.
4. For the implementations of Spell Checker.
5. For the designing of transducer.

Limitations

1. FA can only count finite inputs.
2. It can have only string pattern.
3. Head movement is in only one direction.
4. Limited memory.

Finite state machine.(FSM) or Transducer

A finite state machine is similar to. And except that it has the additional capability of producing output.

FSM= FA + Output Capability

Types of finite state machine.

1. Moore machine
2. Mealy machine

Moore machine

If the output of FSM is depend on present state only then this model of FSM is known as Moore machine

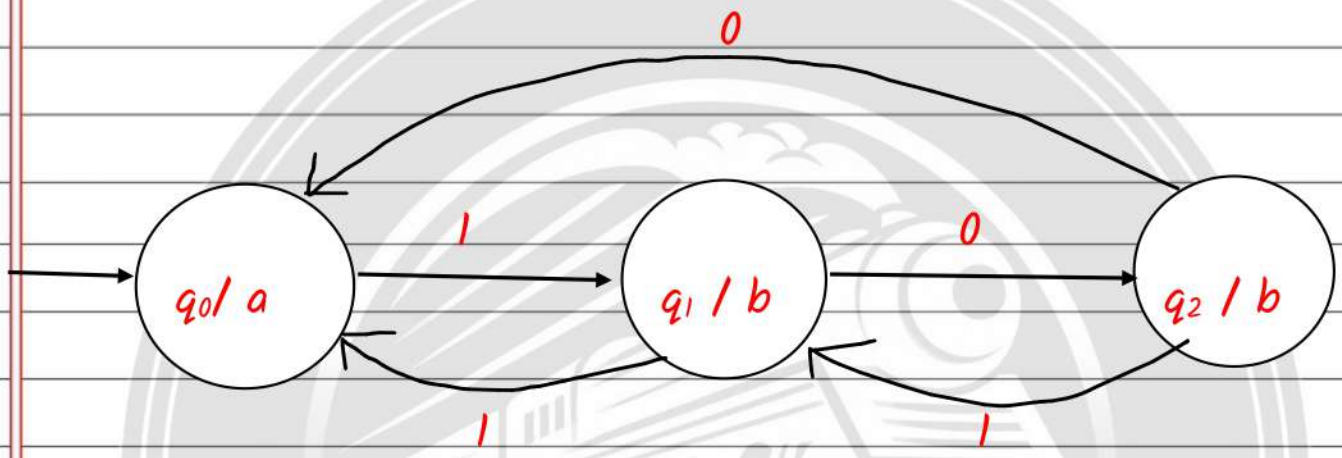
A moore machine can be described by six tuple or objects.

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

Where,

1. Q= It is a finite set of states.
2. Σ= It is the input alphabet
3. Δ= It is the output alphabet
4. δ = It is transition function which maps $\delta : Q \times \Sigma \rightarrow Q$
5. q_0 = It is the initial state.
6. λ= It is the Output function mapping. $\lambda : Q \rightarrow \Delta$

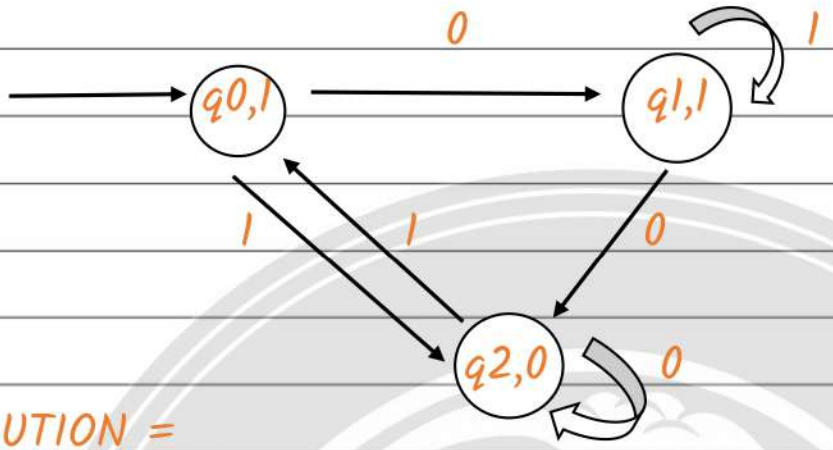
For exaqmple=



Σ= (0,1) , Δ= (a,b) Q= (q_0,q_1,q_2)

Question on moore machine to get output

QUESTION 1= Consider the Moore machine shown in figure below, construct the transition table. What is the output for input string 0110?

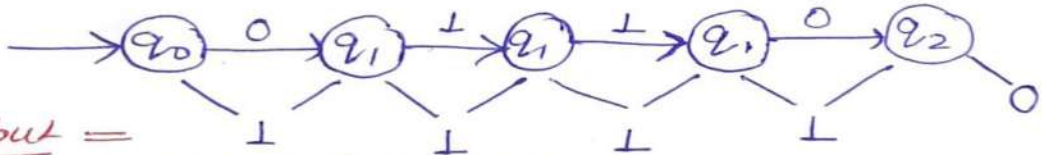


SOLUTION =

Step 1 Design Transition Table from Transition diagram

Present State (PS)	Input 0	Input 1	Output
	Next State	Next State	
→ q_0	q_1	q_2	1
q_1	q_2	q_1	1
q_2	q_2	q_0	0

Step 2 Design the transition graph for the string 0110 is (n) length of input = 4



output = 1 1 1 1 0

length of output = n+1 = 5

Step 3 So the output is 11110 for the input & the output is $\lambda(q_0) = 1$

output formula $Q \rightarrow \Delta$

$q_0 \rightarrow 1$
$q_1 \rightarrow 1$
$q_2 \rightarrow 0$

Mealy machine

If the output of FSM is dependent on present state and present input, then this model of FSM is known as Mealy machine.

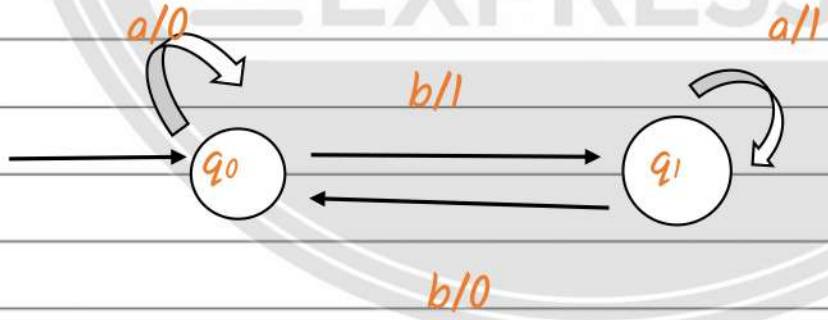
A moore machine can be described by six tuple or objects.

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

Where,

1. Q= It is a finite set of states.
2. Σ= It is the input alphabet
3. Δ= It is the output alphabet
4. δ = It is transition function which maps $\delta : Q \times \Sigma \rightarrow Q$
5. q_0= It is the initial state.
6. λ= It is the Output function mapping. $\lambda : Q \times \Sigma \rightarrow \Delta$

For example=



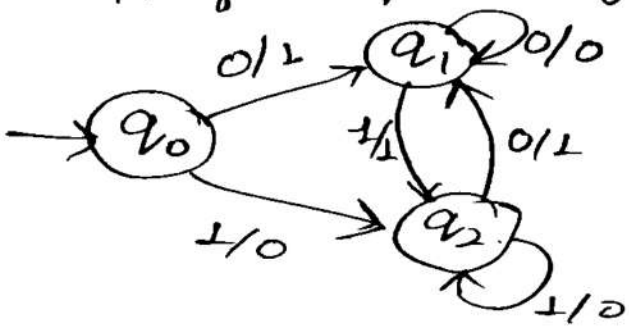
Σ= (a,b) . Δ= (0,1) Q= (q_0, q_1, q_2)

Present State	Next State			
	Input (a)		Input (b)	
	State	output	state	output
→ q_0	q_0	0	q_1	1
q_1	q_1	1	q_0	0

Question on moore machine to get output

QUESTION 1

Ques Considered the Mealy machine shown in figure Construct the transition table find O/P for Input string 01010



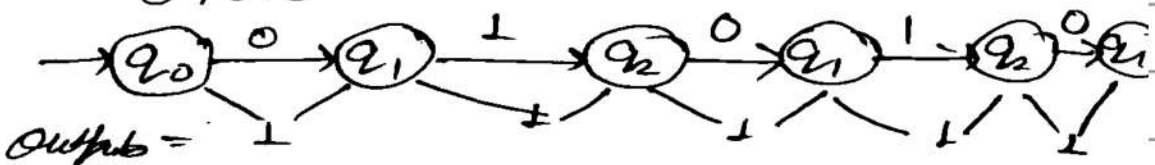
Sol?

Step 1

P.S	Input 0		Input 1	
	NIS	O/P	NIS	O/P
→ q_0	q_1	1	q_2	0
q_1	q_1	0	q_2	1
q_2	q_1	1	q_2	0

Step 2

Now Transition sequence for the input string 01010



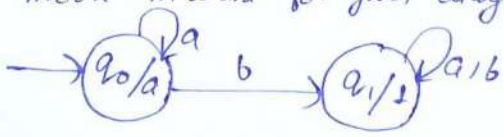
length of input = n

length of output = n

PRACTICE QUESTION

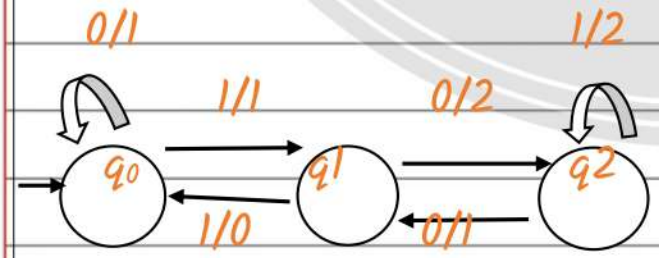
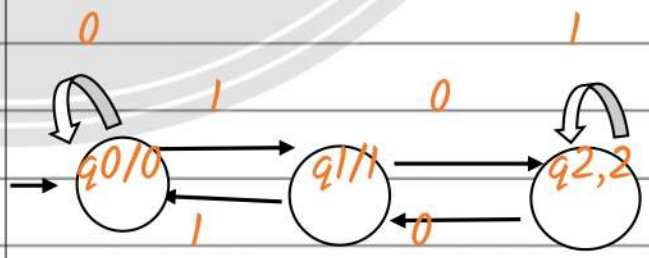
Ques 1 Design mealy machine for given regular expression
re = (0+1)* (00 & 11)
the string will end with 00 or 11

Ques 2 Design moore machine for given diagram for string 0001 and find output



#Difference between MoorE and Mealy Machine.

Moore machine v/s Mealy machine

Mealy machine	Moore machine
Output depends only on present state, and present input.	Output depends on present state only
Length of input string=(n) Length of output string=(n)	Length of input string=(n) Length of output string=(n+1)
Output function. $\lambda: Q \times \Sigma \rightarrow \Delta$	Output function. $\lambda: Q \rightarrow \Delta$
Example.  	Example. 
Mealy output is asynchronous	Moore output is synchronous with clock

#Equivalence of Moore & Mealy conversion

The meaning of word equivalence is that two machines, which accept the same language, hence equivalence of moore and Mealy machine means both the machines generate same output String for same input string.

Procedure for transforming a moore machine to corresponding to mealy machine (Construction)

STEP 1: We have to define. Output function (λ). For mealy machine as a function of present state

We define $\lambda(q, a) = \lambda(\delta(q, a))$ for all state q and input symbol a.

STEP 2: The transition function is the same as that of the given moore machine.

FOR EXAMPLE

QUESTION 1

Ques Construct a Mealy machine which is equivalent to the moore machine given in transition table

Moore Table ⇒

Present State (PS)	Inputs 0 NS	Inputs 1 NS	O/P
→ q_0	q_3	q_1	0
q_1	q_1	q_2	1
q_2	q_2	q_3	0
q_3	q_3	q_0	0

Sol?

$\lambda'(q_0,0) = \lambda(\delta(q_0,0)) = \lambda(q_3) = 0$

$\lambda'(q_0,1) = \lambda(\delta(q_0,1)) = \lambda(q_1) = 1$

$\lambda'(q_1,0) = \lambda(\delta(q_1,0)) = \lambda(q_1) = 1$

$\lambda'(q_1,1) = \lambda(\delta(q_1,1)) = \lambda(q_2) = 0$

$\lambda'(q_2,0) = \lambda(\delta(q_2,0)) = \lambda(q_2) = 0$

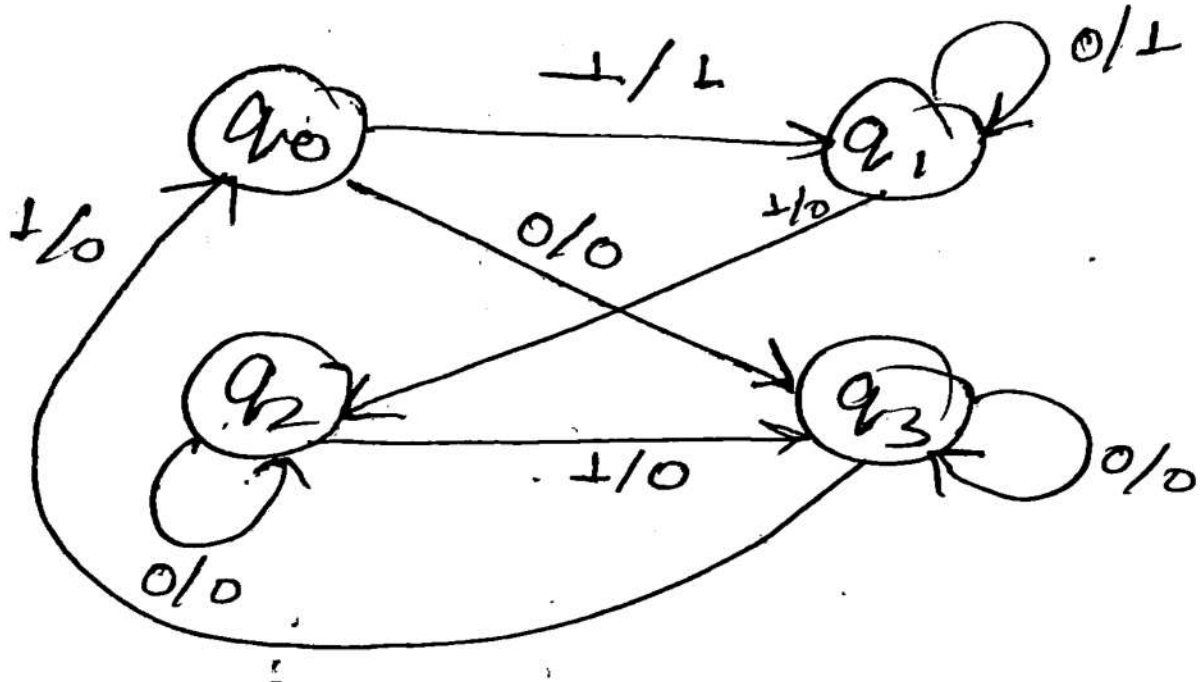
$\lambda'(q_2,1) = \lambda(\delta(q_2,1)) = \lambda(q_3) = 0$

$\lambda'(q_3,0) = \lambda(\delta(q_3,0)) = \lambda(q_3) = 0$

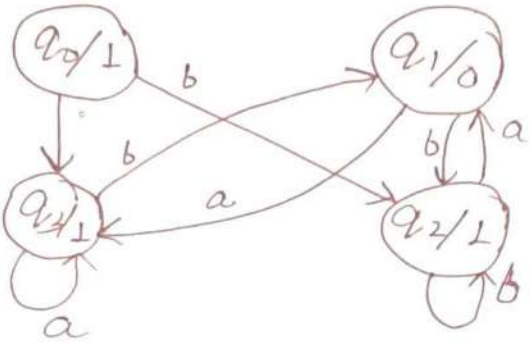
$\lambda'(q_3,1) = \lambda(\delta(q_3,1)) = \lambda(q_0) = 0$

Mealy Table

Present State	Input			
	0		1	
	NS	O/P	NS	O/P
q_0	q_3	0	q_1	1
q_1	q_1	1	q_2	0
q_2	q_2	0	q_3	0
q_3	q_3	0	q_0	0



Ques⇒ Change moore machine to melly m/c



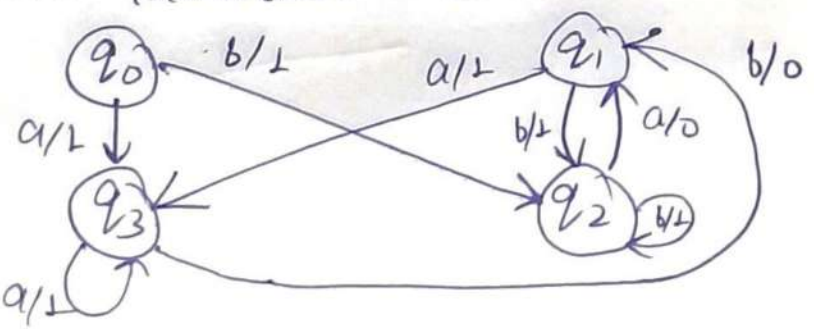
⇒ **Step 1** Construct moore machine transition table

Present State	Input (NS)		O/P
	a	b	
→ q_0	q_3	q_2	1
q_1	q_3	q_2	0
q_2	q_1	q_2	1
q_3	q_3	q_1	1

Step 2 Convert moore machine transition table to melly machine transition table

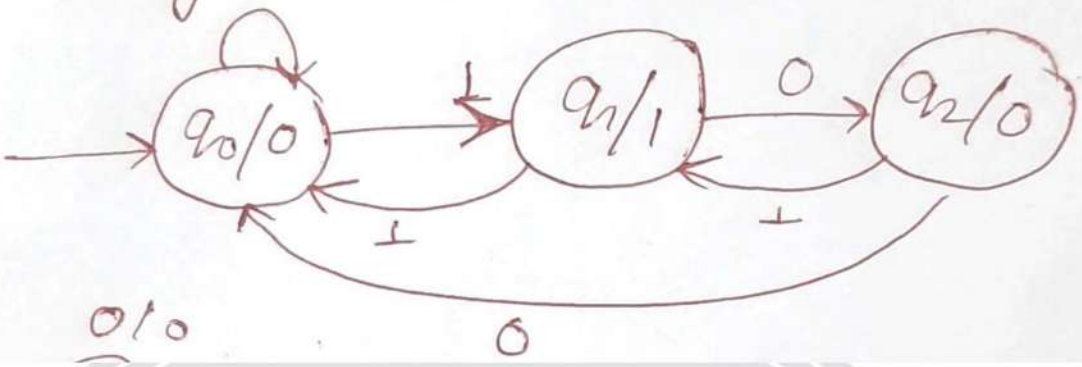
Present State	Next State			
	a	O/P	b	O/P
→ q_0	q_3	1	q_2	1
q_1	q_3	1	q_2	1
q_2	q_1	0	q_2	1
q_3	q_3	1	q_1	0

Step 3 Draw Transition diagram of mealy machine from Transition table



QUESTION FOR PRACTICE

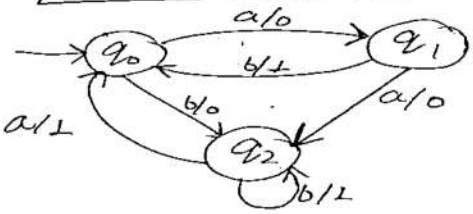
Question Change given moore machine into mealy machine



Procedure for transforming a mealy machine to corresponding to moore machine (Construction)

Mealy M/c to moore M/C

Ques



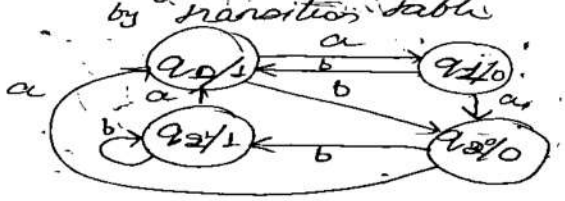
Step 1 construct mealy transition table from the given transition graph

P/S	N/S			
	a	O/P	b	O/P
q_0	q_1	0	q_2	0
q_1	q_2	0	q_0	1
q_2	q_0	1	q_2	1

Step 2 Convert mealy transition table to moore transition table

P/S	Input (N/S)		output
	a	b	
q_0	q_1	q_{20}	1
q_1	q_{20}	q_0	0
q_{20}	q_0	q_{21}	0
q_{21}	q_0	q_{21}	1

Step 3 Now form moore transition diagram by transition table



QUESTION FOR PRACTICES

Ques ⇒

Present State	Input 0		Input 1	
	NS	O/P	NS	O/P
→ q_1	q_3	0	q_2	0
q_2	q_1	1	q_4	0
q_3	q_2	1	q_1	1
q_4	q_4	1	q_3	0

Change mealy to moore machine

Minimization of DFA

Algorithm

Step 1: Eliminate the dead state & unreachable state from the given dfa, (if any)

Step 2: Draw the transition table for rest states after removing the dead state and unreachable state.

Step 3: Split the transition table into two tables, T1 and T2. Where,

T1= Contains all rows which start with states from Q-F

T2= Contains all rows which start with states from F.

Step 4: Find the similar row from T1 such that.

$\delta(q, a) = p$

$\delta(r, a) = p$

Step 5: Repeat step 4 till there is no similar row are available in T1.

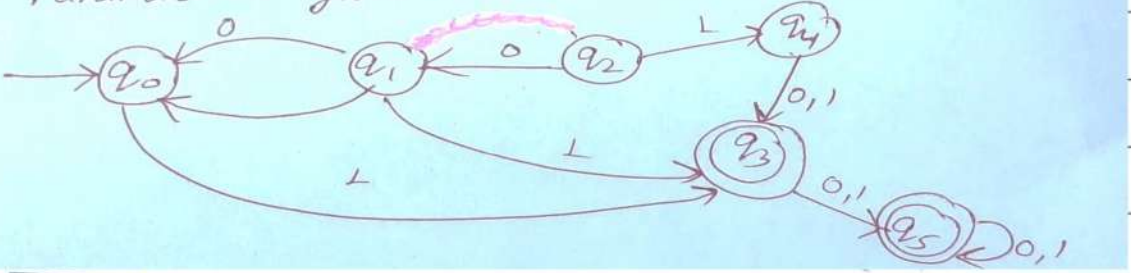
Step 6: Repeat the step 4 and step 5 for table T 2 also.

Step 7: Now combine their reduced T1 and T2 the combined transition table is transition table of minimized DFA

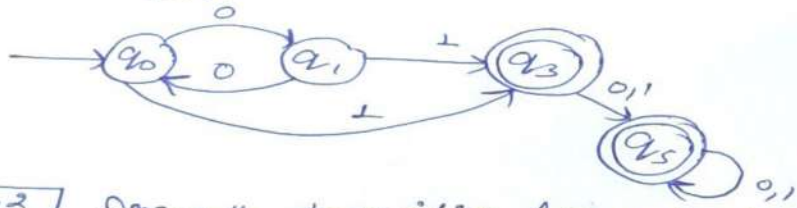
FOR EXAMPLE=

Minimize the Given DFA

Ques Minimize the given DFA



Step 1 Remove the unreachable state q_2 & q_4 from the DFA



Step 2 Draw the transition for rest of the states

	δ/Σ	0	1
→	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 3 Split the transition table into two table T_1 & T_2

T_1 =

δ/Σ	0	1
→ q_0	q_1	q_3
q_1	q_0	q_3

T_2 =

δ/Σ	0	1
* q_3	q_5	q_5
* q_5	q_5	q_5

Step 4 Now consider the table T_1 for similar rows

T_1 =

δ/Σ	0	1
→ q_0	q_1	q_3
q_1	q_0	q_3

Step 5 Now consider the table T_2 for finding similar rows

T_2 =

δ/Σ	0	1
* q_3	q_5 q_3	q_5 q_3
* q_5	q_5	q_5

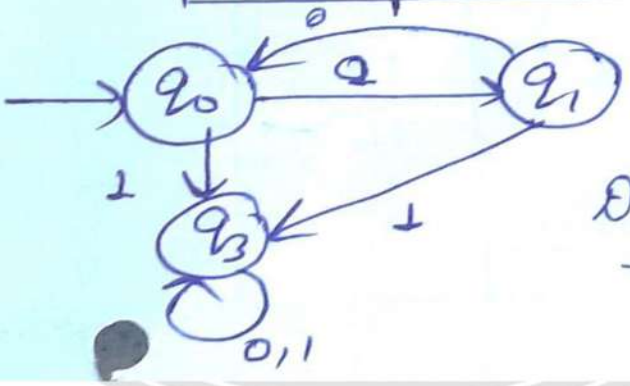
T_2 =

δ/Σ	0	1
* q_3	q_3	q_3

Step 6 Now combined the reduced table T_1 & T_2 the combined transition table is the transition table of minimized DFA

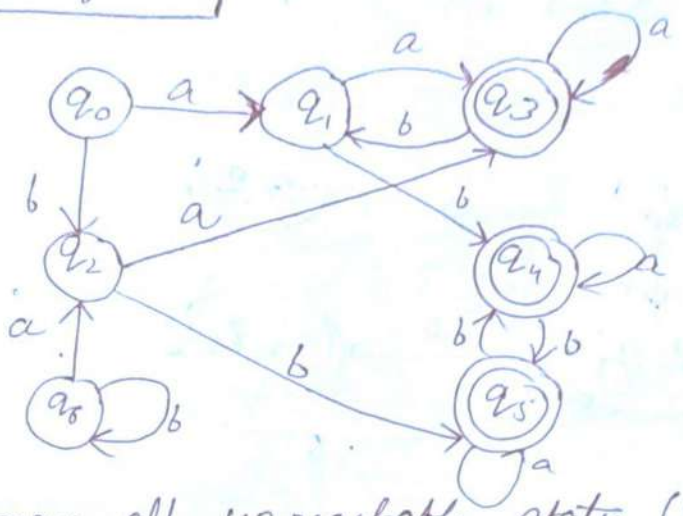
δ/Σ	0	1
→ q_0	q_1	q_3
q_1	q_0	q_3
* q_3	q_3	q_3

Step 7

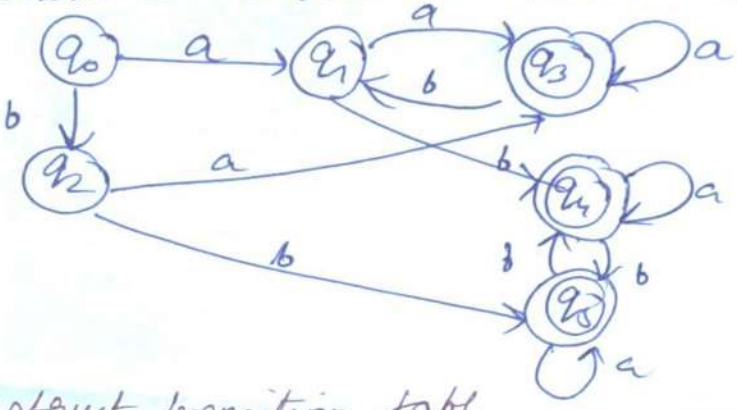


Transition Diagram of minimized DFA

Minimization of DFA



Step 1 Remove all unreachable states (q_6)



Step 2 Construct transition table

	PS	NS	
		a	b
→	q_0	q_1	q_2
	q_1	q_3	q_4
	q_2	q_3	q_5
*	q_3	q_3	q_1
*	q_4	q_4	q_5
*	q_5	q_5	q_4

Step 3 Convert States into equivalent classes

π_0 = $T_1 = q_0, q_1, q_2$

$T_2 = q_3, q_4, q_5$

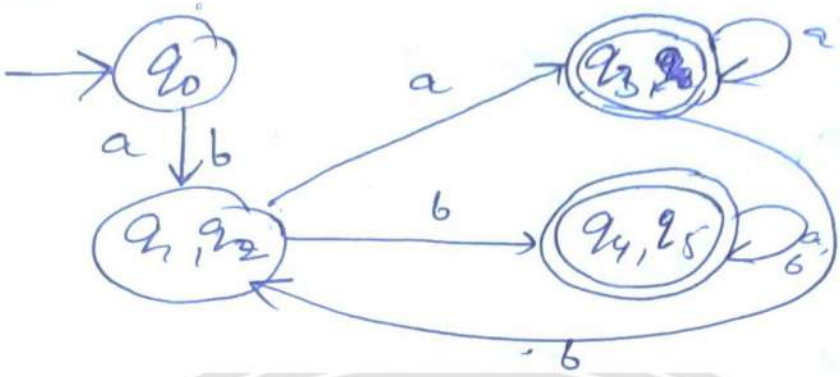
Step 4 Check whether they lie in same group

π_1 = $T_{11} = \{q_0\}$ $T_{12} = \{q_1, q_2\}$

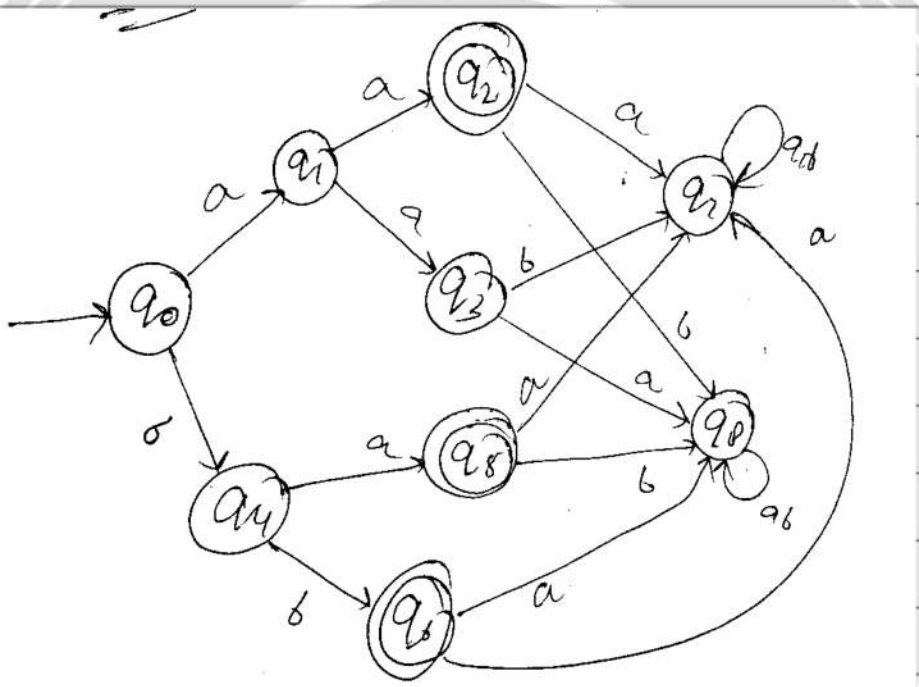
$T_{21} = \{q_3\}$ $T_{22} = \{q_4, q_5\}$

$\pi_2 = \{q_0\}, \{q_1, q_2\}, \{q_3\}, \{q_4, q_5\}$

Step 5 Draw transition diagram



#Question for practice



Thank You

Engineering Express: Simplifying Engineering Education

Are you drowning in complex engineering concepts? Struggling to decipher cryptic notes? Fear not! Welcome to Engineering Express, where we decode the mysteries of engineering subjects for you. 🚀

What We Offer:

Clear Explanations: Our notes break down intricate topics into bite-sized, easy-to-understand language. No jargon, no headaches!

Real-Life Connections: Ever wondered how those abstract formulas apply in the real world? We've got you covered with relatable examples.

Maximize Your Marks: Our mission? Boost your exam scores. With our straightforward notes, you'll be well-prepared to conquer any engineering challenge.

Follow us on social media:

YouTube - https://www.youtube.com/@EngineeringExpress.?sub_confirmation=1

WhatsApp - https://chat.whatsapp.com/H16tpUl2Sm03o6vlcD2D1u

Instagram - https://www.instagram.com/engineeringexpress2312/

Telegram - https://t.me/engineeringexpressofficial

Join the Engineering Express community today and unlock the power of simplified learning! 🎓🔥

THEORY OF AUTOMATA AND FORMAL LANGUAGE (BCS-402)

AKTU New Session (2024-25)

SIMPLE AND EASY EXPLANATION + HANDWRITTEN NOTES PDF

ALL

UNIT-2

TOPIC

Regular Expressions and Languages: Regular Expressions, Transition Graph, Kleen's Theorem, Finite Automata and Regular Expression- Arden's theorem, Algebraic Method Using Arden's Theorem, Regular and Non-Regular Languages- Closure properties of Regular Languages, Pigeonhole Principle, Pumping Lemma, Application of Pumping Lemma, Decidability- Decision properties, Finite Automata and Regular Languages

Engineering Express

Unit	Theory of Automata and Formal Languages
I	**Basic Concepts and Automata Theory:** Introduction to Theory of Computation- Automata, Computability and Complexity, Alphabet, Symbol, String, Formal Languages, Deterministic Finite Automaton (DFA)- Definition, Representation, Acceptability of a String and Language, Non Deterministic Finite Automaton (NFA), Equivalence of DFA and NFA, NFA with ε-Transition, Equivalence of NFA's with and without ε-Transition, Finite Automata with output- Moore Machine, Mealy Machine, Equivalence of Moore and Mealy Machine, Minimization of Finite Automata.
II	**Regular Expressions and Languages:** Regular Expressions, Transition Graph, Kleen's Theorem, Finite Automata and Regular Expression- Arden's theorem, Algebraic Method Using Arden's Theorem, Regular and Non-Regular Languages- Closure properties of Regular Languages, Pigeonhole Principle, Pumping Lemma, Application of Pumping Lemma, Decidability- Decision properties, Finite Automata and Regular Languages
III	**Regular and Non-Regular Grammars**: Context Free Grammar(CFG)-Definition, Derivations, Languages, Derivation Trees and Ambiguity, Regular Grammars-Right Linear and Left Linear grammars, Conversion of FA into CFG and Regular grammar into FA, Simplification of CFG, Normal Forms- Chomsky Normal Form(CNF), Greibach Normal Form (GNF), Chomsky Hierarchy, Programming problems based on the properties of CFGs.
IV	**Push Down Automata and Properties of Context Free Languages**: Nondeterministic Pushdown Automata (NPDA)- Definition, Moves, A Language Accepted by NPDA, Deterministic Pushdown Automata(DPDA) and Deterministic Context free Languages(DCFL), Pushdown Automata for Context Free Languages, Context Free grammars for Pushdown Automata, Two stack Pushdown Automata, Pumping Lemma for CFL, Closure properties of CFL, Decision Problems of CFL, Programming problems based on the properties of CFLs.
V	**Turing Machines and Recursive Function Theory** : Basic Turing Machine Model, Representation of Turing Machines, Language Acceptability of Turing Machines, Techniques for Turing Machine Construction, Modifications of Turing Machine, Turing Machine as Computer of Integer Functions, Universal Turing machine, Linear Bounded Automata, Church's Thesis, Recursive and Recursively Enumerable language, Halting Problem, Post's Correspondance Problem, Introduction to Recursive Function Theory.

UNIT – 2

#Regular expressions and languages

Regular expression

1 The language accepted by a finite automata can be easily described by simple expression called regular expression, or regular language.
2 It is the most effective way to represent any language.
3 A regular expression can also be described as a sequence of pattern that defines a string.
4 Regular expression are used to match character combination in String.
5 String searching algorithm use this pattern to find the operations on a string.

Contain three operations

1 Union.(+)
2 Concatenation.(.)
3 Kleene closure(*)

Note* Positive closer.(+)

Special operators.

1 a^*= {ε, a, aa, aa a, ----}
1 a+= {a, aa, aa a, ----}

Primitive regular expression.

1) ∅ = Is a regular expression represented by the language= { }
2) ε =Is a regular expression represented by the language={ε}

3) a ε Z=Is a regular expression represented by the language={a}

4) Union of two regular expression is represented or written as R1 + R2.

For example,= R1 =a R2=b.

Union= (R1 + R2)=a+b= {a,b}

5) Concatenation of two regular expression is written as R1.R2 . For example,= R1 =a R2=b.

Concatenation.=(R1.R2)=a.b={ab}

6) Kleene Closure Offer regular expression is written as.

R.E= $a^*=\{a^0, a^1, a^2, a^3\text{------}\}$

= {ε, a, aa, aa a, ----}

Value of *= {0,1,2,3------}

Some algebraic laws for. Regular expression

i) Associative law.

L+M=M+L

ii) Commutative law

(L+M)+N=L+(M+N)

(LM)N=L(MN)

iii) Identity law.

Ø+L = L+Ø = L

ε L = L ε = L

iv) Annihilators law

ØL = LØ = Ø

v) Distributive law.

L(M+N)=LM+LN

L+(M.N)=L+M . L+N

vi) Idempotent law.

L+L=L

vii) Laws involving closuer (*)

1. $(L^*)^*=L^*$
2. $(\varepsilon)^* = \varepsilon$
3. $\emptyset^* = \varepsilon$
4. $L^*=L^+ + \varepsilon$
5. $L^+=L+L^*$
6. $L^* L^*=L$
7. $LL^* = L^*L$
8. $\varepsilon + RR^* = R^* = R^*R$
9. $(LM)^*L=L(ML)^*$
10. $(L+M)^* = (L^*M^*)^* = (L^*+M^*)^*$

Question1: Find the regular expression whose length of string is equal to.2

Solution: $L=\{aa,ab,ba,bb\}$ over $\Sigma = \{a, b\}$.

Thus, Regular expression: UNION of L= aa + ab + b a +bb

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

R.E = (a + b) (a + b)

If length of string is 1=?

If length of string is 3=?

Question2: Find the regular expression whose length is at least 2

Solution: L= {aa, ab, b a, bb, aaa, bb b, aba, b ab, -------} over Σ = {a, b}.

Thus, regular expression: UNION of L

= (aa + ab + b a + bb) + aa a +b bb + aba +b ab +---

= (a + b) (a + b) (a + b) *

Question3: Find the regular expression whose length is at most two.

Solution: length= {0,1,2}

$L = \{\varepsilon, a, b, aa, bb, ab, b\,a\}$ over $\Sigma = \{a, b\}$.

$R.E = \varepsilon + a + b + aa + bb + ab + b\,a$

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

$= \varepsilon + (a + b)[\varepsilon + (a + b)]$

Question 4: Find the regular expression whose length is even.

Solution: length= {0,2,4,6....}

$L = \{\varepsilon, aa, bb, ab, b\,a, aaaa, ...\}$ over $\Sigma = \{a, b\}$.

$^*=0$, $R.E = \varepsilon$

$^*=1$, $R.E = (a + b)(a + b)$

$^*=2$, $R.E = (a + b)(a + b)(a + b)(a + b)$

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

Question 5: Write a regular expression for the language where string starts with 0 and ends with one over $\Sigma = \{0, 1\}$.

Solution: L= {01,001,0001,011, 0111, ...}

W= {0_1}

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

Question 6: Write a regular expression for the language where string begins or ends with 00 and 11.

Solution: $R.E = [(00 + 11)(0+1)^*] + [(0+1)^*(00 + 11)]$

Question 7: Write a regular Expression for the language. That contain at least two 0's

Solution: $(1 + 0)^* 0\, 1^* 0\, (1 + 0)^*$ or $1^* 0\, 1^* 0\, (1 + 0)^*$

Question 8: Write a regular Expression for the language, That contain only string in language i.e L={abba}

Solution: R. E= abba

Practice questions

Question 1: Find a regular expression whose length is odd.

Question 2: Find a regular expression whose length is divided by 3.

Question 3: Find the regular expression where number of a's is exactly two over. **Σ** = {a, b}.

Question 4: Find a regular expression where a string ends with B.

Question 5: Find the regular situation where string Starts with or ends with different symbol.

Practice questions

Question 1: Find a regular expression whose length is odd.

Solution: length= {1,3,5,7....}

$L = \{\varepsilon, aaa, bbb, aba, baa, aaaaa, ...\}$ over $\Sigma = \{a, b\}$.

$^*=0$, R. E= $(a + b)$

$^*=1$, R. E= $(a + b)\ (a + b)\ (a + b)$

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

Question 2: Find a regular expression whose length is divided by 3.

Solution: length= {3,6,9....}

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

Question 3: Find the regular expression where number of a's is exactly two over. $\Sigma = \{a, b\}$.

Solution: w= _a _a _

Length = {aa, aba, babab, ...}

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

Question 4: Find a regular expression where a string ends with B.

Solution: w= _ b

Length = {b, ab, aa b, bb b, ...}

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

Question 5: Find the regular situation where string Starts with or ends with different symbol.

Solution: w= (a _b)

(b_ a)

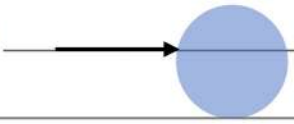
Length= {ab, ba, aab, baa, ...}

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

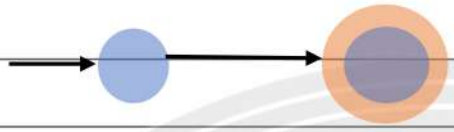
#Transition Graph

Basic symbols and their position graph to note.

1) $\emptyset$ =

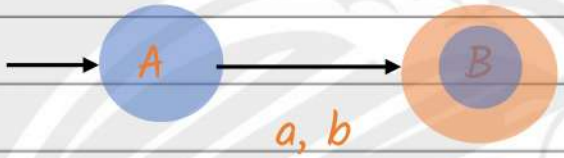


2) a=

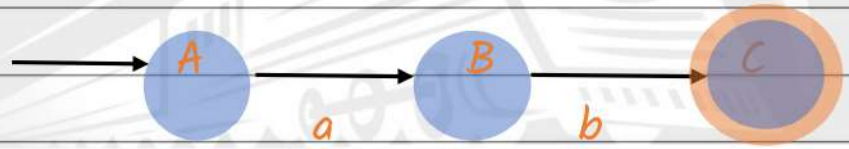


initial state final state

3) a+b =



4) a . b=

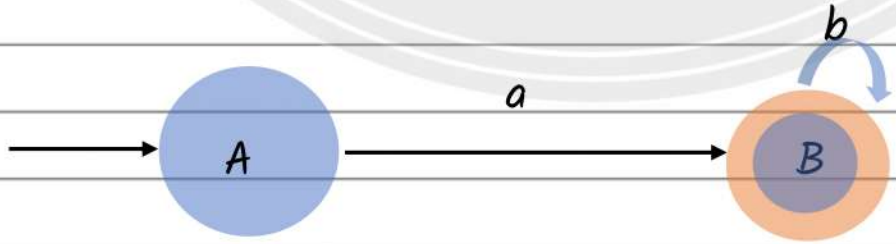


5) a^* = {ε, a, aa, aa a, ----}

Question 1: Construct FA equivalent to RE.$(a \cdot b^*)$

Solution : Regular expression = $a \cdot b^*$

Finite automata transition diagram



Practice question

Question 2: Construct FA equivalent to RE.$(ab + ba)^*$

Kleene's Theorem(R. E to FA)

This theorem state that any regular language can be accepted by FA. Or the language accepted by any FA is regular language.

Or

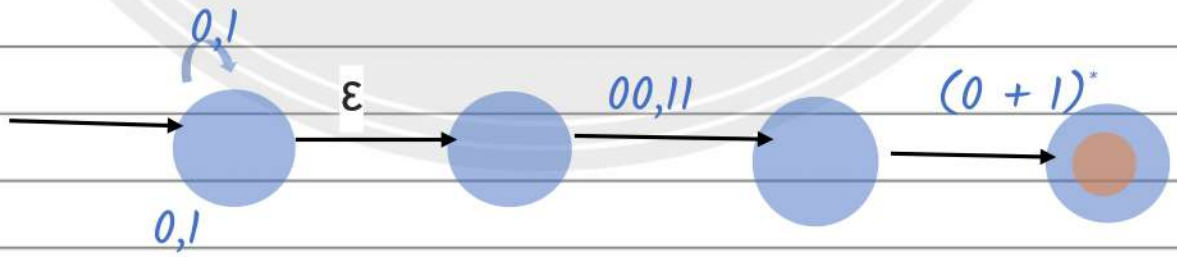
For any regular expression r that represents language.L(r) There is a finite automaton that accepts same language.
Say R1 and R2 be 2 regular expression then.,

1) r1 + r2 is a regular expression too , whose correspoding language is L(r1)U L (r2)
2) r1.r2 is a regular expression too , whose correspoding language is L(r1). L (r2)
3) r1* Is a regular expression to whos corresponding language is L(r1)*

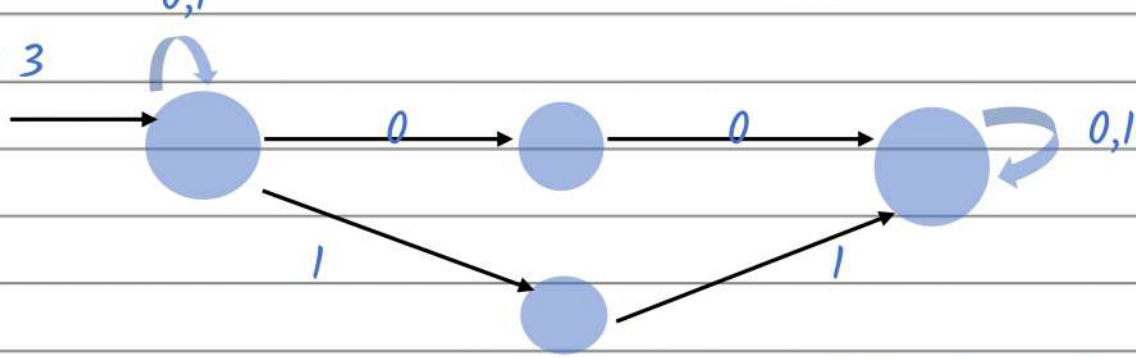
Question: Construct FA equivalent to RE. $(0 + 1)^*(00 + 11)(0 + 1)^*$

Solution: step 1

Step 2



step 3



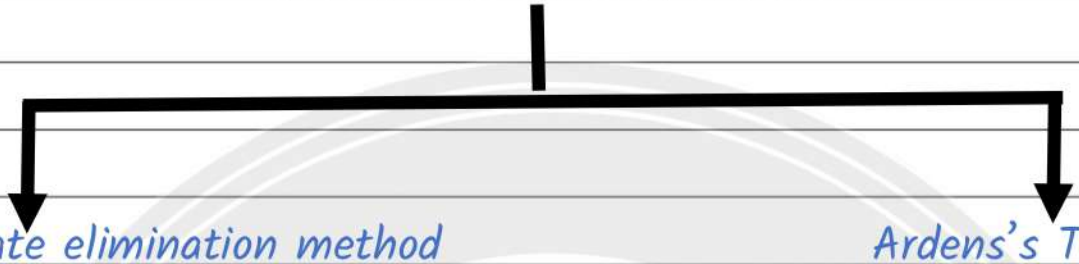
Question: Construct FA equivalent to RE 01 [((10)*+ 111) + 0]*1

(for pratice)

Question: Construct FA equivalent to RE 10 + (0 + 11)0*1

(for pratice)

#FA TO RE BY 2 METHODS



Ardens's Theorem

"This theorem state that. If P and Q are two regular expression. Over Σ, and If P does not contain. ε, then The following equation in R given by

$$R = Q + RP. \text{ -------(i)}$$

Has a unique solution, that is,

$$R = QP^* \text{ --------(ii)}$$

That means, whenever we get any equation in the form of $R = QP^*$ Then we can directly replace by $R = QP^*$. So here first we will prove that $R = QP^*$ Is the solution of this equation. And then we will also prove that it is the unique solution of this equation.

(by replacing with eqn (ii))

After replacing R by $R = QP^*$ we get,

$$R = Q + QP^* P$$

Taking Q as common,

$$R = Q(\varepsilon + P^* P)$$

$$R = QP^*$$

As we know that $\varepsilon + R^*R = R^*$ Hence proved.

Thus , $R = QP^*$ is the solution of the equation (i)

(by replacing with eqn (i))

Using eqn (i), $R = Q+RP$

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

$R = Q+ QP + RP^2$

$R = Q + QP +(Q + RP)P^2$

$R = Q + QP+ QP^2 + RP^3$

If we substitute the value of R recursively

$R=Q+QP+QP^2+ QP^3+-------$

$R=Q(\hat{}+ P+ P^2+ P^3+-------)$

$\boxed{R=QP^*}$

FA to RE by Arden's Theorem

Step 1: Transition diagram should not have Epsilon transition.

Step 2: Must have only one single initial state.

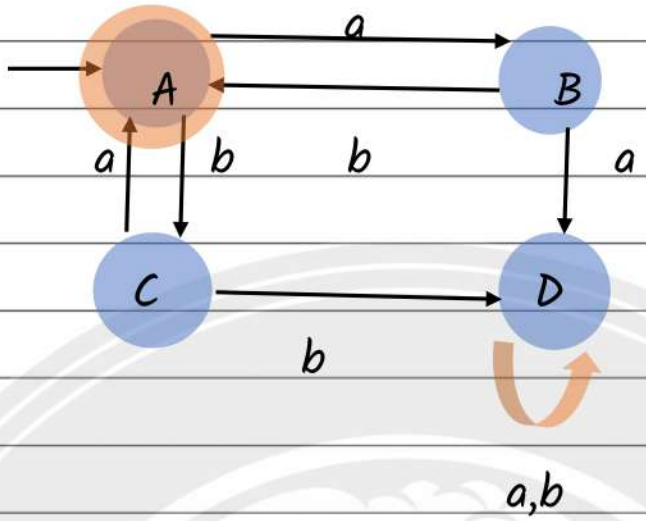
Step 3: Vertices are suppose. Q1, q2, q3.....qn, where qi is the final state

Step 4: Wij Denotes the regular expression representing set of labels of edges from qi to qj

Step 5: Write a equations for each state and check their incoming edges.

Step 6: Take the final state and simplify the equation for the same final state in the form of Arden's Theorem expression.

Question 1: Derive regular expression for the following finite automata.



Solution:

Step 1: Write the equation for each state and check their incoming edges. $A = \varepsilon, Bb + Ca$ ------(i)

$B = Aa$ ------(ii)

$C = Ab$ -------(iii)

$D = Da + Db + Ba + Cb$ -------(iv)

Sep 2: Take the final state and simplify the equation from the same final state in the form of Aren's theorem

$A = \varepsilon\ Bb + Ca$

$A = \varepsilon + Aab + Aba$

$A = \varepsilon + A\{ab + ba\}$

$B = Aa$

$C = Ab$

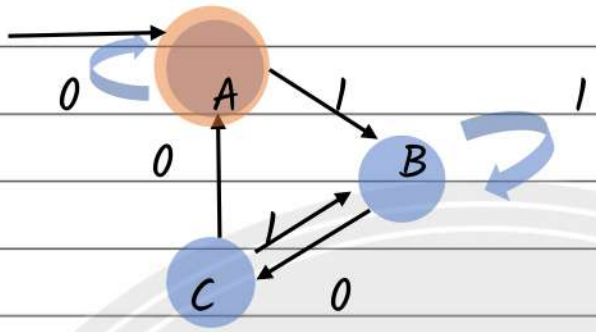
If we compare this eqn with $R = Q + RP$ then

$R = QP^*$

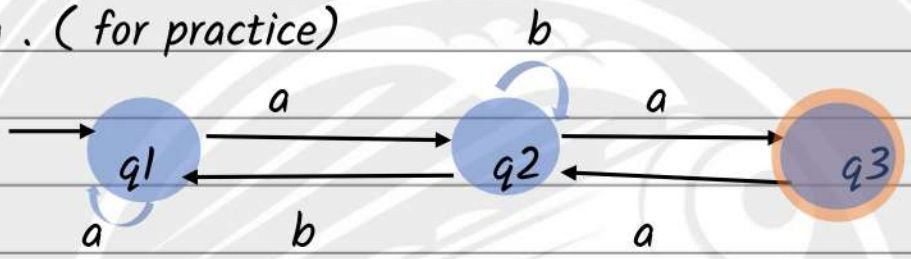
Therefor, $A = \varepsilon.\{ab + ba\}^*$

$A = (ab + ba)^*$

Question 1: Derive the regular expression from the following finite automata . (for practice)



Question 2: : Derive the regular expression from the following finite automata . (for practice)



Algebraic method using Adren's theorem.

Some basic identities of regular expression.

- $r = R.E$
- $\emptyset = 0$
- $\varepsilon = 1$
- $\emptyset + r = r$
- $\emptyset . r = r . \emptyset = \emptyset$
- $\varepsilon . r = r . \varepsilon = r$
- $\varepsilon^* = \varepsilon$ and $\emptyset + r = r$
- $r + r = r$
- $r^* . r^* = r^*$
- $r . r^* = r^* . r = r^+$
- $(r^*)^* = r^*$
- $\varepsilon + r . r^* = r^*$
- $(P.Q)^* P = P(QP)^*$
- $(P+Q)^* = (P+Q)^* = (P^* + Q^*)^*$
- $(P+Q).r = P.r + Q.r$ and $r.(P+Q) = r.P + r.Q$
- If $R = Q + RP$, then the unique solution is : $R = QP^*$, i.e ARDEN'S THEOREM

Question 1: Prove

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

Solution:

taking L.H.S

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

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

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

$10^*(\varepsilon+(0+10^*1)^*(0+10^*1))$ {by $\varepsilon+ r.r^* = r^*$ }

Let $(0+10^*1)=R$,

$10^*(\varepsilon+R^*R)$ {by $\varepsilon+ r.r^* = r^*$ }

$10^*(0+10^*1)^* = R.H.S$

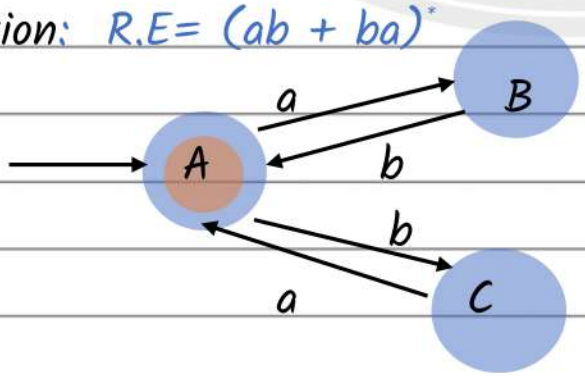
Question for practice

Prove that the R.E , $R=(\varepsilon + 1^*(011)^*(1^*(011)^*)^*$ all describ e same set of strings

Practice question

Question 1: Construct FA equivalent to RE.$(ab + ba)^*$

Solution: $R.E= (ab + ba)^*$

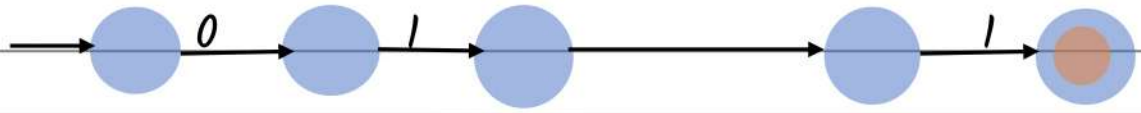


Question 2: Construct FA equivalent to RE 01 [((10)*+ 111) + 0]*1

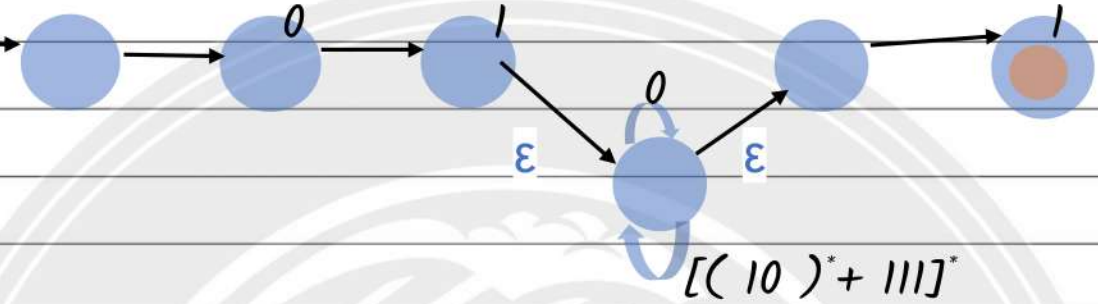
Solution:

$[((10)^* + 111) + 0]^*$

Step 1:

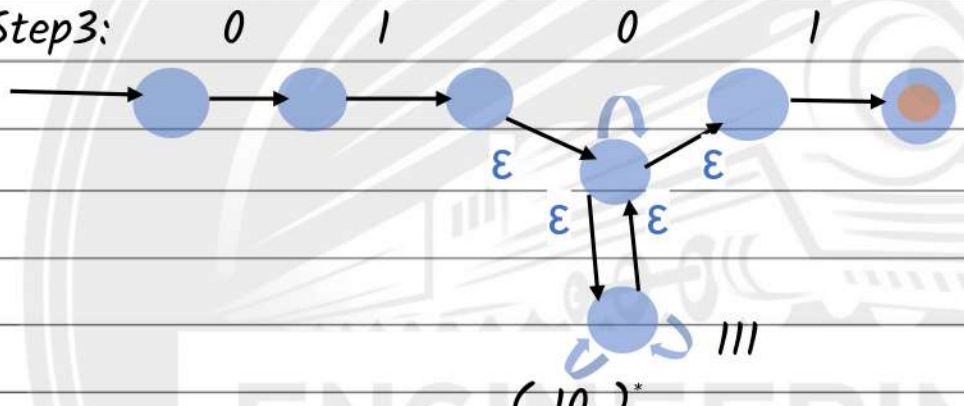


Step2:

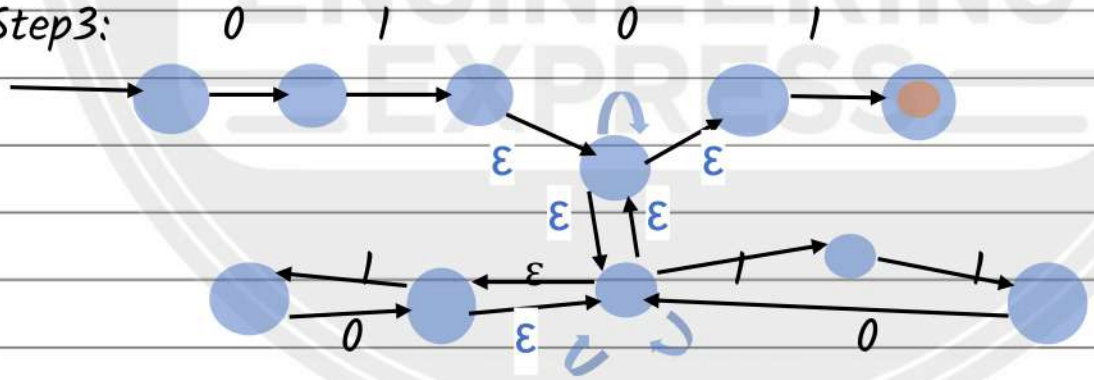


$[(10)^* + 111]^*$

Step3:



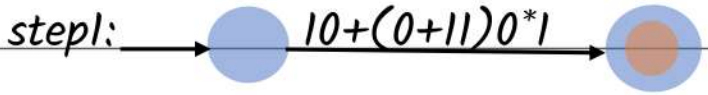
Step3:



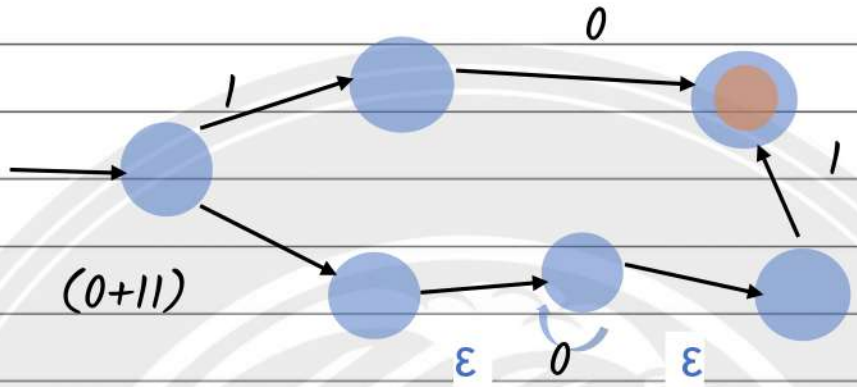
Question3: Construct FA equivalent to RE 10 + (0 + 11)0*1

Solution:

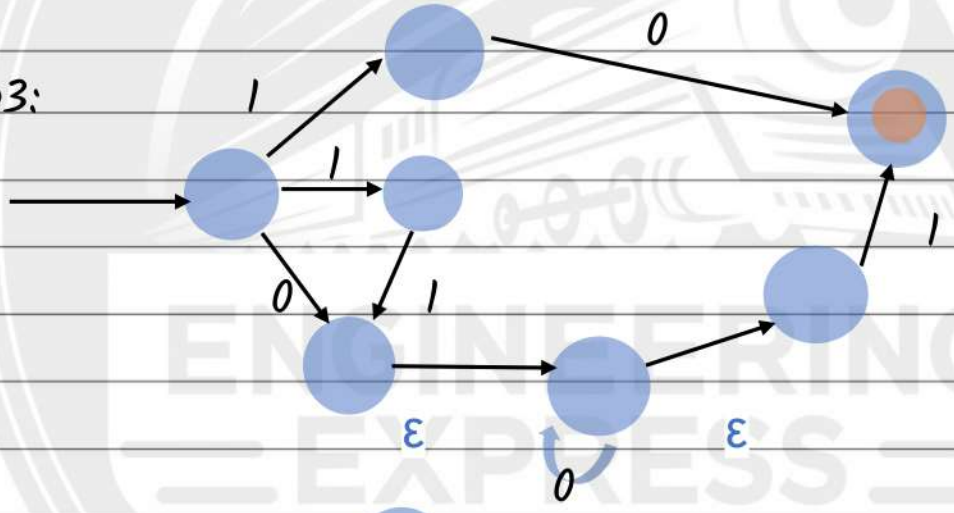
step1:



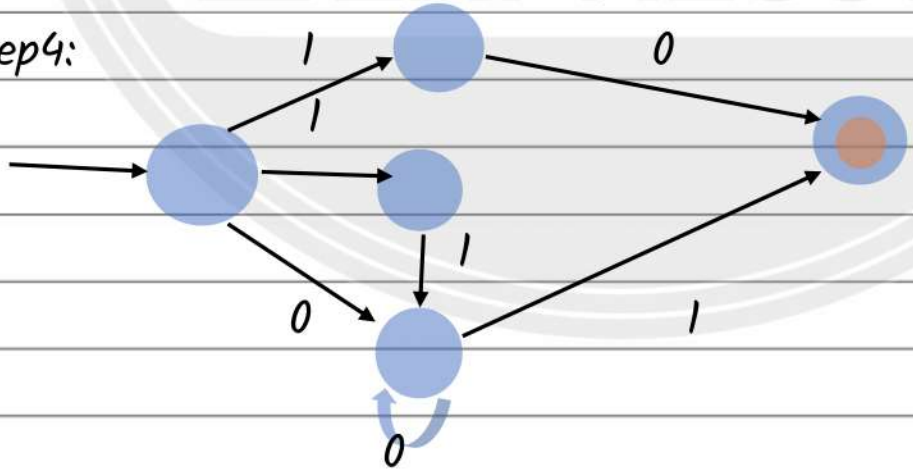
Step 2 :



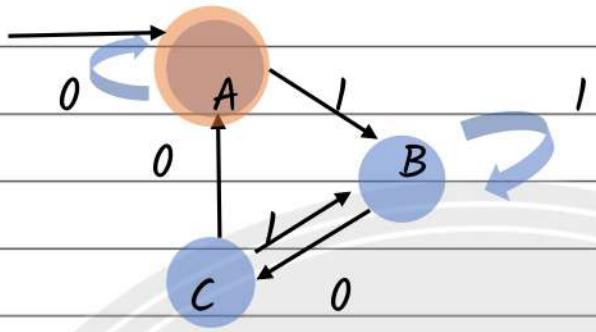
Step3:



Step4:



Question 4: Derive the regular expression from the following finite automata . (for practice)



Solution: $A=A0+C0+\wedge$

$B=A1+B1+C1$

$C=B0$

Substituting this C in B

$B=A1+B1+B01$

$B=A1+B(1+01)$

$B=A1(1+01)^*$

Substituting this B in C

$C= A1(1+01)^*0$

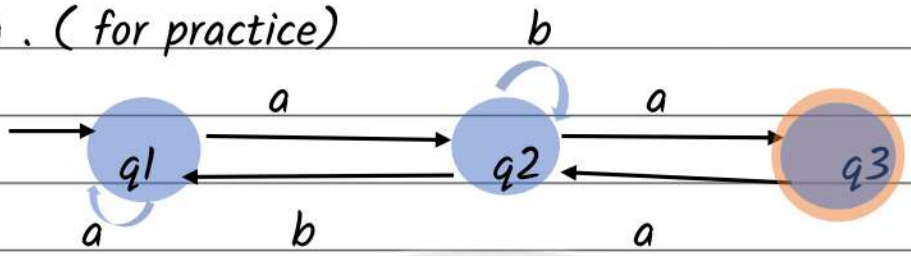
Substituting this C in A

$A=A0+A1(1+01)^*00+\wedge$

$A=\wedge+A[0+1(1+01)^*00]$

$A=[0+1(1+01)^*00]^*$

Question 5: Derive the regular expression from the following finite automata . (for practice)



Solution: : $q1 = q1a + q2b + \wedge$

$q2 = q1a + q2b + q3a$

$q3 = q2a$

Substituting value of q3 in q2

$q2 = q1a + q2b + q2aa$

$q2 = q1a + q2(b + aa)$

$R \quad Q \quad R \quad P$

$$\left\{ \begin{array}{c} \text{if } R = Q + RP \\ R = QP^* \end{array} \right\}$$

$q2 = q1a(b + aa)$

Substituting this q2 in q1

$q1 = q1a + q1a(b + aa)^*b + \wedge$

$q1 = \wedge + q1a + q1a(b + aa)^*b$

$q1 = \wedge + q1[a + a(b + aa)^*b]$

$q1 = [a + a(b + aa)^*b]$

Substituting this q1 in q2

$q2 = [a + a(b + aa)^*b]^*a(b + aa)^*$

Substituting this q2 in q3

$q3 = [a + a(b + aa)^*b]^*a(b + aa)^*a$

Question 6: Prove that the R.E , $R=(\varepsilon + 1^*(011)^*(1^*(011)^*)^*$ all describ e same set of strings

Solution: $R=(\varepsilon + 1^*(011)^*(1^*(011)^*)^*$

$R= \varepsilon + PP^*$, where $P=1^*(011)^*$

$R= P^*$ [Since Basic Identity, $\varepsilon + PP^* = \varepsilon + P^*P = P^*$]

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

$= (P^*Q^*)^* = (P+Q)^*$ [Basic Identity]

Thus, $R=(1+(011))^*$

Hence it describes the same set of strings

Finite automaton to Regular expression using state elimination method.

Steps required to find the regular expression for any given DFA by state elimination method are as follows:-

Step1:- Initial state should not have any incoming edges.

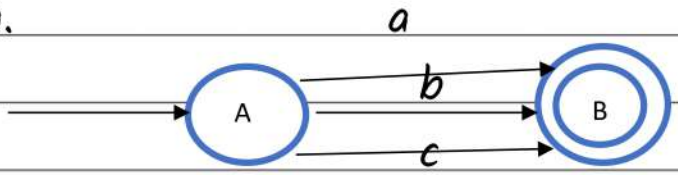
Eg.

Step2:- Final state should not have any outgoing edge.

Eg.

Step3:- Other than the initial state and the final state, delete or eliminate other states one by one.

Question1:- Derive a regular expression from the following finite automata.

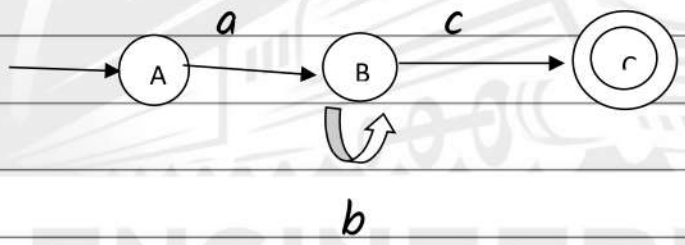


Solution:-Step1:-Initial state has no incoming edges, thus no chnage

Step2:-Final state has no outgoing edges, thus no change.

Step3:- Thus, the final regular expression: (a+b+c)

Question2:- Derive a regular expression from the following finite automata.

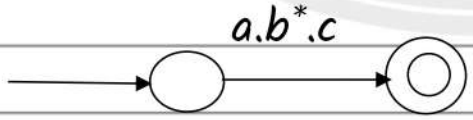


Solution:-Step1:-Initial state has no incoming edges, thus no chnage

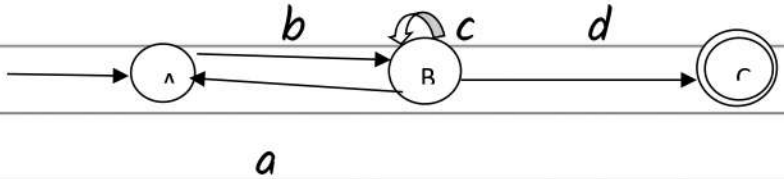
Step2:-Final state has no outgoing edges, thus no change.

Step3:- Other than the initial state and the final state, delete the state numbers B as it is in other state.

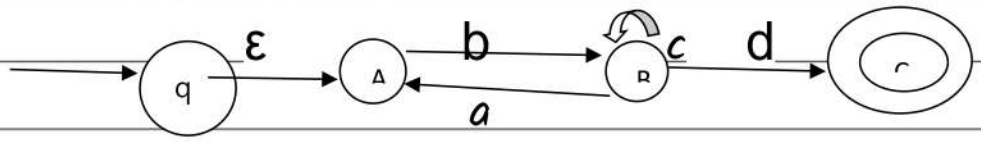
Step4:- Thus, the final regular expression: (a.b*.c)



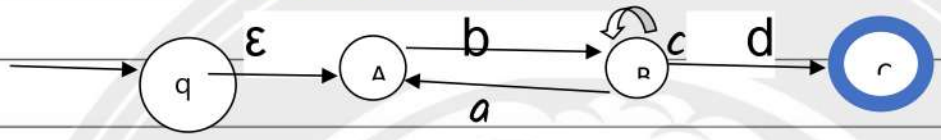
Question3:- Derive a regular expression from the following finite automata.



Solution:- Step1:- Initial state has incoming edge. Thus the resultant FA.

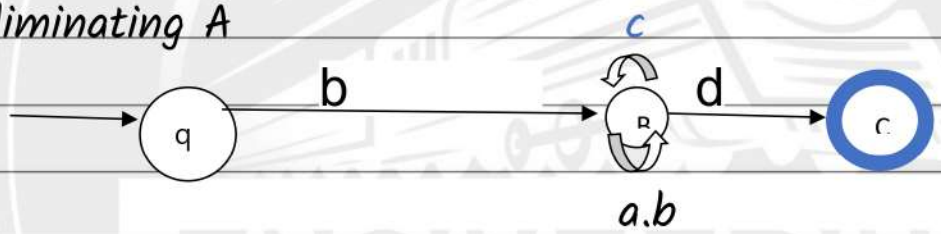


Step2:- Final state has no outgoing edges. thus the resultant faith

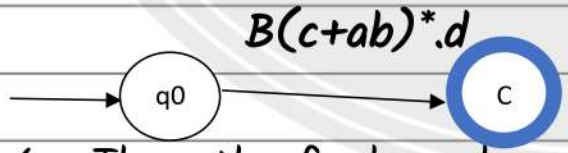


Step3:- Other than the initial state and the final state delete or eliminate other states one by one.

Eliminating A



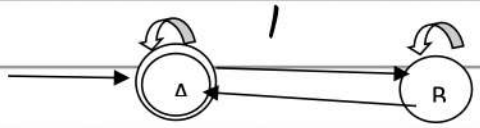
Eliminating B



Step4:- Thus, the final regular expression: B(c+ab)*.d

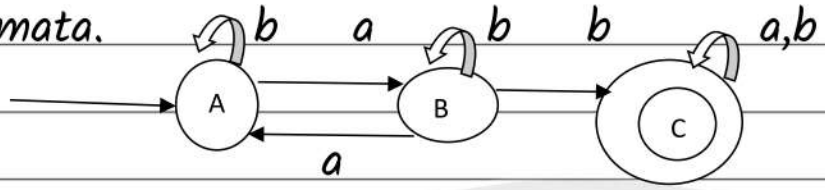
Practice questions

Question1:- Derive a regular expression from the following finite automata. 0 0



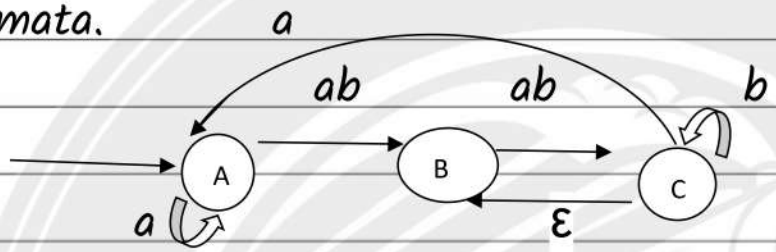
1

Question2:- Derive a regular expression from the following finite automata.



Question3:- Derive a regular expression from the following finite automata.

.



#Closure Properties

1.Closure properties on regular language are defined as certain operations on regular language, which are guaranteed to produce regular language.

2.Closure refers to some operations on a language, resulting in a new language that is of same 'type' as originally operated on. That is regular.

Properties are:-

1. Kleene Closure
2. Positive closure
3. Complement
4. Concatenation
5. Union
6. Intersection
7. Reverse or transpose
8. Set Difference
9. Homomorphism
10. Inverse Homomorphism

- Kleene closure:- If L1 is a regular language, it's kleene closure $L1^*$ will also be Regular

  Example = L1=(aUb)

  $L1=(aUb)^*$

- Positive closure:- If L1 is a regular language, it's Positive closure $L1^+$ will also be Regular

  Example = L1=(aUb)

  $L1=(aUb)^+$

- Complement:-If L(G) is a regular language , its complement L'(G) will also be regular. Complement of a language can be found by subtracting string which are in L(G)From all possible strings.

  Example =1. $L(G)= \{a^n / n>3\}$

  $L'(G)= \{a^n / n<=3\}$

  2.L1=length equal to even

  $= \{0,2,4,6,8,10,....\}$

  L1'=length equal to odd

  $= \{1,3,5,7,9,11.....\}$

- Union:- If L1 and if L2 are 2 regular languages, their union , L1 U L2 will Also be regular.

  Example = $L1=\{a^n / n>0\}$

  $L2=\{b^n / n>0\}$

  Union L3=L1 U L2

  $=\{a^n \cup b^n / n>0\}$is also regular

- Intersection:- If L1 and if L2 are two. Regular languages, the intersection. $L1 \cap L2$ will also be regular

Example = $L1=\{a^m b^n / n>0\}$

$L2=\{ a^m b^n \cup b^n a^m / n>0 \text{ and } m>0\}$

Intersection $L3= L1 \cap L2$

$=\{a^n b^n / n>0 \text{ and } m>0\}$ is also regular

- Concatenation:- If L1 and if L2 are two. Regular languages, the concatenation. L1.L2 will also be regular

  Example= $L1=\{a^n / n>0\}$

  $L2=\{b^n / n>0\}$

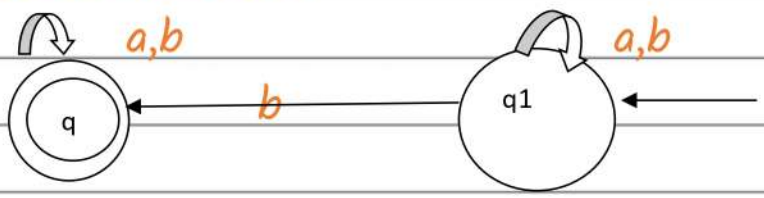
  concatination $L3=L1 . L2$

  $=\{a^n . b^n / n>0 \text{ and } m>0\}$ is also regular

- Reverse or transpose:- If a language L is a regular language, then it's transpose or reverse should be a regular language. We follow the following steps.

  1. Transition diagram should only one final state.
  2. We interchange final state as initial and initial state as final state.
  3. We reverse the direction of the edges between the states for Eg. if we have

About diagram having only one final state, so we can directly interchange final state as initial state and initial state as final state.



$L^R= (a+b)^*ba^*$

- Set difference:- If L1 and if L2 are two. Regular languages, the concatenation. L=L1-L2 will also be regular

  Example = $L1=a^*+b(a+b)^*$

  $L2=b^*a(a+b)^*$

  Set difference $L=L1-L2$

  $=L1 \cap L2'$ is also regular

- Homomorphism:-A homomorphism on an alphabet is a function that gives a sting for each symbol in that alphabet.

  Example= $h(0)=ab$; $h(1)=bb$

  If $L=\{00,11\}$ then

  $h(L) = \{abab,abbb\}$

  Therefore h(L) is a homomorphic language for the language L.

- Inverse Homomorphism:- Let h be a homomorphism and L language whose alphabet is the output language of h.

  The inverse homomorphism: $h^{-1}(L)=\{w|h(w) \text{ is in } L\}$

  Example= let $h(0)=a$, $h(1)=ab$ & $L=\{abab\}$

  Thus $h^{-1}(L)=\{0101,22,201,012\}$

#To identify laguage is regular or note

Basic questions

Note:-

- [if power =1 = always regular]
- [if there are different power= always regular]
- [when condition is finite= always regular]
- [same powers= not regular]]

Question1:- Identify whether the following language are regular or not. $L= \{a^n|n>=0\}$

Solution:- [note: if power =1 = always regular]

Thus $\{a^n|n>=0\}$ is acceptable by finite automata

Question2:- Identify whether the following language are regular or not. $L= \{a^n b^m | n,m>=1\}$

Solution:- [note: if there are different power= always regular]

Thus $\{a^n b^m | n,m>=1\}$ is acceptable by finite automata

Question3:- Identify whether the following language are regular or not. $L= \{a^n b^n | n<=20^3\}$

Solution:- [note: when condition is finite= always regular]

Thus $\{a^n b^n | n<=20^3\}$ is acceptable by finite automata

As there exists a finite value of n

Question4:- Identify whether the following language are regular or not. $L= \{a^n b^n | n>=1\}$

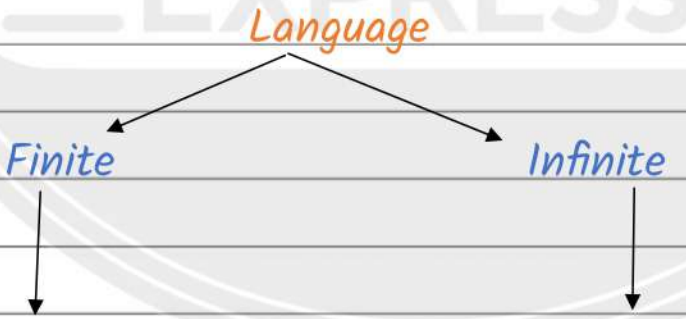
Solution:- [note: same powers= not regular]

Thus $\{a^n b^n | n>=1\}$ not a regular language

as not finite value exists

Pumping lemma(it's a negative test)

Pumping lemma is used to prove that a language is not regular



Always regular

Regular or may non- regular can't Defined

Pumping lemma state that, if L is a regular language then the L has Pumping length 'n' i.e , no of state in FA , such that any string 'w' where |W|>=n may be divided in 3parts w=xyz such that th following conditions must be true .

Steps required to satisfy the pummping lemma are as follows:-

Step1:- let n be the no of states the corresponding automata

Step2:- select a string W such that th length of W should be $|W|>=p$. using pumping lemma

$W=xyz$

1. $|y|$ not equal to ^
2. $|xy|<=p$

Step3:- find the suitable integer I such that

xy^iz not belongs to L

This contradicts our assumption L is non regular

Question1:- Using pumping lemma proof that the following language is not regular. $L=\{a^n b^n / n>0\}$

Solution =

Step1:-let 'L' be regular language and a pumping length be 'p'

Step2:-let choose a string $w=a^pb^p$ and divide the string into 3 parts x,y and z

Step3:- let us assume that p =7 , than w=aaaaaaabbbbbbb [7a's and 7b's]

Case1:- let 'y' be in 'a' part: w= a a a a a a a b b b b b b b

x y z

Case2:- let 'y' be in 'b' part: w= a a a a a a a b b b b b b b

x y z

Case3:- let 'y' be in both 'a' and 'b' part:

$w = \underbrace{a\ a\ a\ a\ a}_{x}\ \underbrace{a\ a\ b\ b}_{y}\ \underbrace{b\ b\ b\ b\ b}_{z}$

Step4:- let i=2/0

Case1:- thus xy^iz: $W = a\ a\ a\ a\ a\ \ a\ a\ \ b\ b\ b\ b\ b\ b\ b$

$W = a^5.a^4.b^7 = a^9.b^7$ (not valid to string)

Case2:- thus xy^iz: $W = a\ a\ a\ a\ a\ a\ a\ \ b\ b\ b\ \ b\ b\ b\ b$

$W = a^7.b^6.b^4 = a^7b^{10} =$ (not valid to string)

Case3:- thus xy^iz: $W = a\ a\ a\ a\ a\ \ a\ a\ b\ b\ \ b\ b\ b\ b\ b$

$W = a^5.a^2.b^2.a^2.b^2.b^5 = a^9b^9$

These 3 cases are not satisfied it means this language is not regular

Practice Questions

Question 1:- Show that $L-\{a^p | P \text{ is prime}\}$ is not regular

Question2:- Show that $L=\{0^i1^i \mid i>=1\}$ is not regular

Question3:- Show that $L= \{a^{2^n} | n>1\}$ is not regular

Applications of pumping lemma.

This theorem can be used to prove that certain sets are not regular. The general steps in its applications are.

Step1:- Assume L is a regular, let n be the no of states the corresponding automata

Step2:- select a string W such that th length of W should be $|W|>=p$. using pumping lemma

$W=xyz$

1. $|y|$ not equal to ^
2. $|xy|<=p$

Step3:- find the suitable integer I such that xy^iz not belongs to L

This contradicts our assumption L is non regular

Pigeonhole principle.

1. It stated that if n pigeons are assigned to m pigeon holes, then at least one pigeonhole contains two or more pigeons.$(m<n)$
2. The pigeonhole principle is sometimes useful in counting methods.

Example= Show thatin a group of 50 students atleast 5 are born in same month.

Solution= There are $m=12$ months (pigenhole)

And $Km+1=50$

$K12=50-1$

$K=49/12$

Remainder 1 , atleast $4=1$ staudents are born in same months

- In automata theory, pigeons are the strings of 0's, The pigeonholes are the states & correspondings associated each string eith the state to which goes when the string is i/p
- This principle is used to prove that certain infinite languages are not regular

Decision Properties

A decision property for a class of language is an algorithm that takes a formal description of a language, example, DFA and tells whether or not some property holds.

Approximately all the properties are decidable in case of finite automata.

1. Emptiness
2. Non- Emptiness
3. Finiteness
4. Infiniteness
5. Membership
6. Equality

These are explained as following below.

- Emptiness ends. Non emptiness:- [$L=\emptyset$ or $L\neq\emptyset$]

  Step1= Select the state that can not be reached from the initial state and delete them, (remove unreachable states.)

  Step2= If the resulting machine contain at least one final state, so then the finite automata, except the non empty language.

  Step3= If the resulting machine is free from final state, then finite automata, except empty language.

- Finiteness and infiniteness:- [L=finite or L= infinite]

  Step1= Remove the unreachable states.

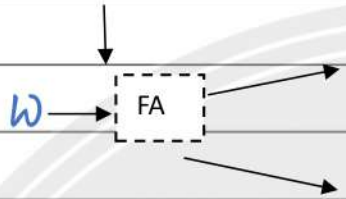
  Step2= Remove dead states.

  Step3= If the resulting machine contained loops or cycles, then the finite automata, except infinite language.

Step4= If the resulting machine do not contain loop or cycle, then the finite automata except finite language.

- Membership:- [if L= R.L, then W ∈ L or W ∉ L]

L=R.L => FA



- Equality:-[L1=L2 or L1 ≠ L2]

Two finite state automata M1 and M2 is set to be equal if they accept the same language. Minimize the finite state automata and the Minimal. dfa will be unique.

L1= FA= MINIMIZATION=M1, if M1=M2 then L1=L2
L2= FA= MINIMIZATION=M2, otherwise L1 ≠ L2

Thank you

Engineering Express: Simplifying Engineering Education

Are you drowning in complex engineering concepts? Struggling to decipher cryptic notes? Fear not! Welcome to Engineering Express, where we decode the mysteries of engineering subjects for you.

What We Offer:

Clear Explanations: Our notes break down intricate topics into bite-sized, easy-to-understand language. No jargon, no headaches!

Real-Life Connections: Ever wondered how those abstract formulas apply in the real world? We've got you covered with relatable examples.

Maximize Your Marks: Our mission? Boost your exam scores. With our straightforward notes, you'll be well-prepared to conquer any engineering challenge.

Follow us on social media:

YouTube - https://www.youtube.com/@EngineeringExpress.?sub_confirmation=1

WhatsApp - https://chat.whatsapp.com/H16tpUl2SmQ3o6vlcD2DIu

Instagram - https://www.instagram.com/engineeringexpress2312/

Telegram - https://t.me/engineeringexpressofficial

Join the Engineering Express community today and unlock the power of simplified learning!

THEORY OF AUTOMATA AND FORMAL LANGUAGE (BCS-402)

AKTU New Session (2024-25)

SIMPLE AND EASY EXPLANATION + HANDWRITTEN NOTES PDF

ALL

UNIT-3

TOPIC

Regular and Non-Regular Grammars: Context Free Grammar(CFG)-Definition, Derivations, Languages, Derivation Trees and Ambiguity, Regular Grammars-Right Linear and Left Linear grammars, Conversion of FA into CFG and Regular grammar into FA, Simplification of CFG, Normal Forms- Chomsky Normal Form(CNF), Greibach Normal Form (GNF), Chomsky Hierarchy, Programming problems based on the properties of CFGs.

Engineering Express

Unit	Theory of Automata and Formal Languages
I	**Basic Concepts and Automata Theory:** Introduction to Theory of Computation- Automata, Computability and Complexity, Alphabet, Symbol, String, Formal Languages, Deterministic Finite Automaton (DFA)- Definition, Representation, Acceptability of a String and Language, Non Deterministic Finite Automaton (NFA), Equivalence of DFA and NFA, NFA with ε-Transition, Equivalence of NFA's with and without ε-Transition, Finite Automata with output- Moore Machine, Mealy Machine, Equivalence of Moore and Mealy Machine, Minimization of Finite Automata.
II	**Regular Expressions and Languages:** Regular Expressions, Transition Graph, Kleen's Theorem, Finite Automata and Regular Expression- Arden's theorem, Algebraic Method Using Arden's Theorem, Regular and Non-Regular Languages- Closure properties of Regular Languages, Pigeonhole Principle, Pumping Lemma, Application of Pumping Lemma, Decidability- Decision properties, Finite Automata and Regular Languages
III	**Regular and Non-Regular Grammars**: Context Free Grammar(CFG)-Definition, Derivations, Languages, Derivation Trees and Ambiguity, Regular Grammars-Right Linear and Left Linear grammars, Conversion of FA into CFG and Regular grammar into FA, Simplification of CFG, Normal Forms- Chomsky Normal Form(CNF), Greibach Normal Form (GNF), Chomsky Hierarchy, Programming problems based on the properties of CFGs.
IV	**Push Down Automata and Properties of Context Free Languages**: Nondeterministic Pushdown Automata (NPDA)- Definition, Moves, A Language Accepted by NPDA, Deterministic Pushdown Automata(DPDA) and Deterministic Context free Languages(DCFL), Pushdown Automata for Context Free Languages, Context Free grammars for Pushdown Automata, Two stack Pushdown Automata, Pumping Lemma for CFL, Closure properties of CFL, Decision Problems of CFL, Programming problems based on the properties of CFLs.
V	**Turing Machines and Recursive Function Theory** : Basic Turing Machine Model, Representation of Turing Machines, Language Acceptability of Turing Machines, Techniques for Turing Machine Construction, Modifications of Turing Machine, Turing Machine as Computer of Integer Functions,

Headline – Black Color

Sub Headings – Orange Color

Paragraph – Blue color

Note – Red color

UNIT – 3

#REGULAR AND NON REGULAR GRAMMAR

Context free grammar(CFG)

CFG stands for context free grammar. It is a formal grammar which is used to generate all possible patterns of string in a given formal language.

Context free grammar is denoted by G.

or

Mathematically, contest free grammar defined as four tuples i.e

$G=[V,T,P,S]$ or $[V_N, \Sigma, P,S]$ or $[V_N,V_T,P,S]$

Where,

V / V_N = A finite , non-empty set of variables(non-terminal) generally represented by capital letters.
e.g. A,B,C,D.....

Σ/ V_T / T = A finite , non-empty set of terminals(constant) generally represented by small letters.
e.g. a,b,c,d.....

S= Starting non terminal call starting symbol of grammar.

$S \in V_N$

P= Set of rules or production values in CFG.

Following observations are considered regarding the production rules.

1. Reverse subscription is not permitted

   If [S-> AB] is a production Then we can replace s by AB but we can't replace AB by S.

2. No inversion operation is permitted.

   If [S-> AB] is a production , it is not necessary that AB->S is a production

Notation used:

We will use following notations:- alpha , beta , gamma

1. All capital letters A,B,C.... are use as variables
2. All small letters a,b,c.... are use as terminals
3. X,y,z,w denote string of terminal
4. α, β, γ (alpha , beta , gamma) denote the elements of $(V_N \cup \Sigma)^*$
   i.e. denote string of terminals or variables or both including null staring
5. Any symbol to the power zero is equivalent to null. That is,
   $X^0 = \varepsilon$ (null)
   Where, $X \in (V_N \cup \Sigma)$

If A is any set, then A^* denotes the set of all string over A. A^* denotes $A^* = \{\varepsilon\}$

Where, ε is the empty string

Derivation [meaning of ' -> ' = derive]

If [α-> β] Is a production in grammar 'G' ,then [α=> β] is called one step derivation or production

If $\alpha \rightarrow \alpha 1 \Rightarrow \alpha 2 \Rightarrow \alpha 3 \ldots\ldots \Rightarrow \alpha n \Rightarrow \beta$ Than

$$\alpha \overset{n}{\underset{G}{\Rightarrow}} \beta$$ is called n step derivation.

The notation $\overset{G}{\Rightarrow}$ Represents relation for grammar(G).

E.g=

Question1:-consider a grammar G define as G=({S} ,{a,b} {s-> aS, S-> ε},S)

Solution:- we have, V_n={S}

Σ = {a,b}

P= {s-> aS, S-> ε}

S= starting symbol

S=>aS [by using S->aS]

S=>aaS [by using S->aS]

S=>aaaS [by using S->aS]

$S \Rightarrow a^n S$ [as a is repeats many time (n)]

$S \Rightarrow a^n \varepsilon$ [by using S-> ε]

$S \Rightarrow a^n$

Therefore, language $L(G)=\{a^n | n>=0\}$

Question2:- Consider a grammar G=({S},{0,1},{S->0S1, S-> ε}, S)

Solution:- we have, V_n={S}

Σ = {0,1}

P= {s-> 0S1, S-> ε}

S= starting symbol

S=>0S1 [by using S->0S1]

S=>00S11 [by using S->0S1]

S=>000S111 [by using S->0S1]

$S \Rightarrow 0^n S 1^n$ [as a is repeats many time (n)]

$S \Rightarrow 0^n \varepsilon 1^n$ [by using S-> ε]

$S \Rightarrow 0^n 1^n$

Therefore, language $L(G)=\{0^n 1^n | n>=0\}$

Question3:- For generating a language that generates equal number of a's and b in the form, a^nb^n, then CFG will be defined as.

G=({S,A} , {a,b} , {S-> aAb, A->aAb/ ε})

Solution:- given,

V_n={S,A}

Σ = {a,b}

P= { S-> aAb, A->aAb/ ε}

Now,

S=>aAb

S=>aaAbb

S=>aaaAbbb

S=>aaa ε bbb

S=> a^n ε b^n

S=> a^n b^n

Therefore, language L(G)={a^nb^n|n>=0}

#Practice question

Question1:- Consider a grammar

G=({S} , {a,b} , {S-> SS,}S)

Find L(G)=?

Question2:- Consider a grammar G=({S,A,B},{a,b},P,S), where P consist of S-> aAAa, A-> baABb,

B->Aab, aA->baa, bBb->abab

test wether W=baabbabaaabba is in G

Question3:- Constructors CFG for a language L= a^n b^{2n} where n>=1

Derivation

Derivation in the sequence of production group. It is used to get the input string through these production rules.

During parsing, we have to take two decisions.
These are as follows.

1. We have to decide the non terminal, which is to be replaced.
2. We have to decide the production rule by which the non terminal will be replaced.

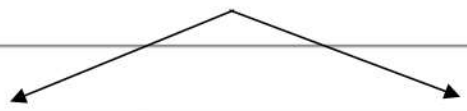
Derivation tree / Parse tree.

Derivation tree is a graphical representation for the derivation of the given production rule. For a given CFG,
the derivation tree is also called parset tree.

The followings are the properties of any derivation tree:-

1. The root node is always a node indicating start symbol.
2. The derivation is used from left to right.
3. The leaf nodes are always terminal nodes.
4. The interior node are always the non-terminal nodes.
5. The collection of leaves from left to right denotes the string W.

There are two types of derivation



Left most derivation [left to right]

Right most derivation. [right to left]

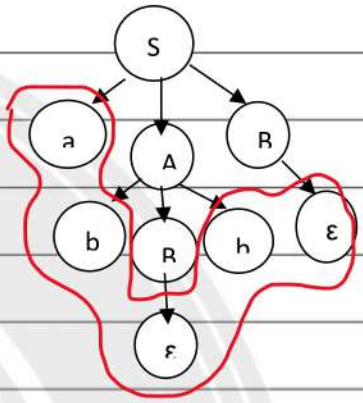
1. Leftmost derivation:- If at each step we replace the left most variable by one of its production bodies, such derivation is called left most derivation.

Or

The geometrical representation of leftmost derivation is called as a leftmost derivation tree.

E.g:- given , S->aAB
A->bBb
B->A/ ε

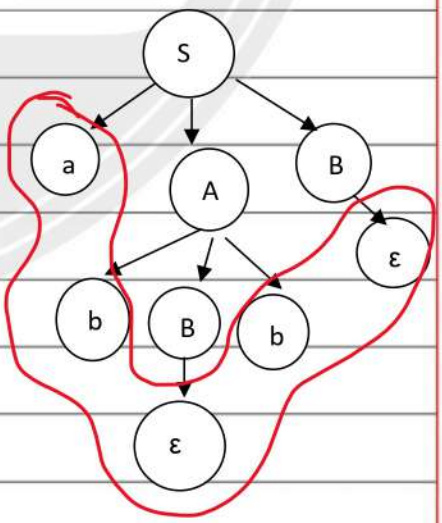
Solution= S->aAB [by using S->aAB]
S-> abBbB [by using A->bBb]
S-> ab ε bB [by using B-> ε]
S-> abb ε [by using B-> ε]
S-> abb



2. Right most derivation:- If at each step we replace the right most variable one by one of its production bodies, such derivation is called right most derivation.

E.g:- given , S->aAB
A->bBb
B->A/ ε

Solution= S->aAB [by using S->aAB]
S-> aA ε [by using B-> ε]
S-> aA
S-> abBb [by using A->bBb]
S-> ab ε b [by using B-> ε]
S->abb



Practice Question

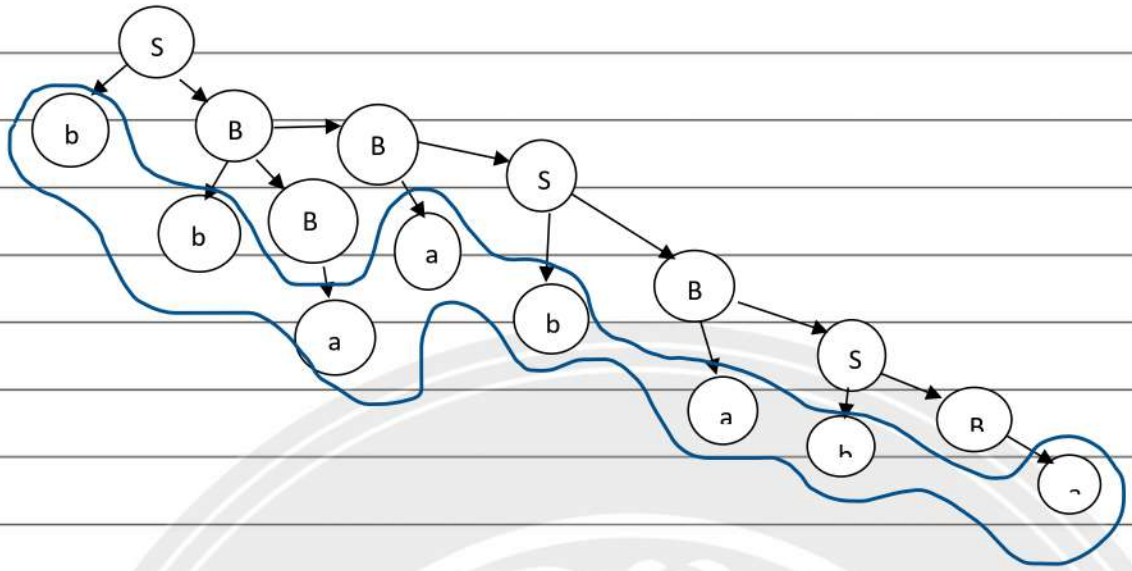
Question1:-Consider the production rule of a grammar 'G': S->S+S|S*S|a|b. Find the left most and right most derivation for the string W:-a*a+b

Question2:-derive the string W:- aabbabba with left most derivation and right most derivation using a CFG given by
S-> aB|bA
A-> a|aS|bAA
B-> b|bS|aBB

#Derivation Form & Parse Tree

Question1:- consider the grammar {S->bB /aA}
{A->b / bS / aAA}
{B-> a / aS / bBB}
For the string w=bbaababa. Find leftmost derivation , right most derivation, parse tree
Solution:- * LEFT MOST DERIVATION:-

S->bB
S-> bbBB [by using B->bBB]
S->bbaB [by using B->a]
S->bbaaS [by using B->aS]
S->bbaabB [by using S->bB]
S->bbaabaS [by using B->aS]
S->bbaababB [by using S->bB]
S->bbaababa [by usingB->a]



*RIGHT MOST DERIVATION

S->bB

S->bbBB [by using B->bBB]

S->bbBaS [by using B->aS]

S->bbBabB [by using S->bB]

S->bbBabaS [by using B->aS]

S->bbBAbabB [by using S->bB]

S->bbBababa [by usingB->a]

S->bbaababa

Practice Question

Question1:- Consider the Grammar- S->A|B

A->0A| ε

B->0B|1B| ε

For the string W=00101 find, left most derivation, roght most derivation , parse tree

Question2:- Consider the Grammar- S->bAb|a|b

For the string W=bbabb find, left most derivation, roght most derivation , parse tree

Ambiguity or Amiguous

A grammar G is said to be ambiguous if. The grammar produce more than one parsec tree for same string.

An ambiguous grammar is one that produces more than one leftmost or rightmost derivation for same string.

*NOTE

1. A Grammar G is unambiguous if there exist exactly one Parse Tree or left most derivation | Right most derivation for all string $W \in L(G)$

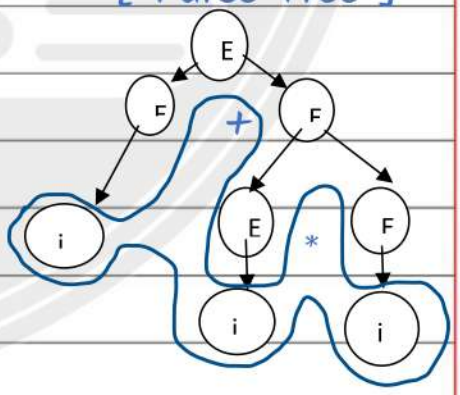
2.If a Grammar G is ambiguous , it doesn't mean language (L)is ambiguous

E.g= E->E+E
E->E*E
E->id

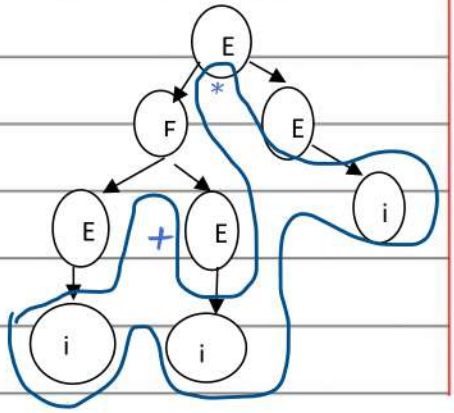
Find for the string w= id+id*id

Solution:- :Left most derivation [Parse Tree]

1. E->E+E [By using E->E+E]
E->id+E [By using E->id]
E->id+E*E [By using E->E*E]
E->id+id*id [By using E->id]



2. E->E*E [By using E->E*E] [Parse Tree]
E->E+E*E [By using E->E+E]
E->id+E*E [By using E->id]
E->id+id*E [By using E->id]
E->id+id*id [By using E->id]



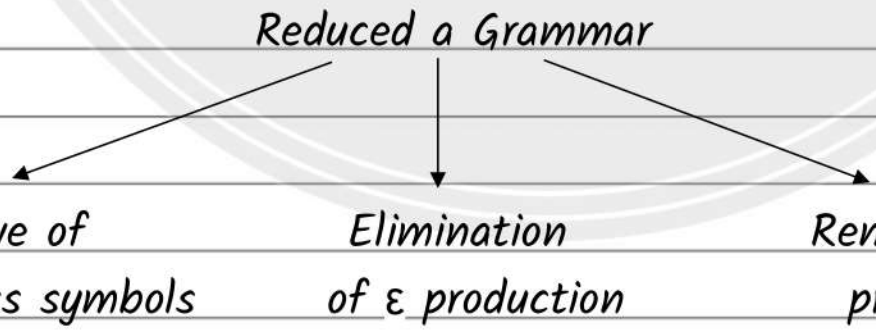
Question:- What is ambiguity? Show that S->aS|Sa|a is an Ambiguity grammar for the string w= aaa

Question:- Explain why the grammar below that generate string. S->0A|1B with an equal no of zeros A->0AA|1S|1 and one's is ambiguous B->1BB|0S|0 W=000111

#Simplification of CFG

In CFG, sometimes all the production rules and symbols are not needed for the derivation of strings. Beside this, there may also be some null productions and unit productions. Elimination of these productions and symbol is called simplification of cfg.

1. Each Variable and each terminal of G appears in the derivation of some string in L
2. There should not be any production as X->Y where X and Y are non terminals
3. If ε is not in the language L then there is no need of the production X-> ε



Removal of useless Symbols:-if the productions having useless symbols remove that production. Any symbol is uselful When it appears on the right hand side in the production rule and generates some terminal string. If no such derivation exists, then it's supposed to be an useless symbol.

Question:- G->(V_N , Σ, P,S) where V_N={S,T,X}; Σ={0,1};
S->0T|1T|0|1|x(Rule 1)
T->00(Rule 2)

Solution:- S->0T	S->1T	S->0	S->1	S->X
S->000	S->100			
(Useful)	(useful)	(usefull)	(useful)	(useful)

Remove S->X
Reduce grammar is
P->[S->0T | 1T | 0 | 1]
T->00

V_n={S,T}
Σ={0,1}
S-> Starting Symbol

Question:- Find CFG with no useless symbol Equivalent to
S->AB|CA ; B->AC|AB ; A->a ; C->aB|b

Solution:- S->AB	S->CA	B->BC	B->AB	A->a
S->aB	S->ba		B->AB	C->b
S->aBC				
(Useless)	(useful)	(useless)	(useless)	(useless)

P= S->CA ; C->b ; A->a V_N= {S,A,C} Σ={a,b}
S-> Starting Symbol

For practice

Question Remove useless symbols from the following grammar
S->aA|bB ; A->aA|a ; B-> bB ; D->ab| ε a ; ε ->aC|d

Elimination of ε production:- If CFG non-terminal variable or symbol a is nullable variable, if there is a production A-> ε or there is a derivation that starts at A and leads to ε

Step1:- Find out all the nullproductio and then find out all nullable variables (If it directly produce A-> ε or after some steps , it generates null then that variable is called nullable variable)

Step2:- After finding nullable variable , go to R.H.S of every variable and wherever that variable is present ,write that with it or without it , which means that with A or without A and than eliminate A-> ε.

Question1:- Remove the null production

S->aA ;A->b| ε

Solution:- So if we replace

S->a ε [by using A-> ε]

S->a

Therefore , S->ab |a

A->b

Question2:- consider the following grammar nd remove null prduction. S->ABAC ; A->aA| ε ; B->bB| ε ; C->c

Solution:- if we replace non terminals in S by giving values one by one we get

S->ABAC	A->aA -> a By using A-> ε	B->bB->b By using B-> ε
S-> ABAC By using A-> ε S->BAC	S-> ABAC By using B-> ε S->AAC	S-> ABAC By using A-> ε ABC
S-> ABAC By usingC->c A/B-> ε S->Ac	S-> ABAC By using C->c A-> ε S->Bc	S-> ABAC By using A/B-> ε S->C

Therefore, S->ABAC | BAC |AAC |ABC |Ac |Bc | C

A->aA|a

B->bB|b

C->c

Practice question

Question:- consider the following grammar and remove the null

1. S->aSa
   S->bSb| ε

2. S->a|Xb|aYa
   X->Y| ε
   Y->b| ε

3. S->AbbA
   A->aA|bA| ε

Removal of unit production:- The unit production are the productions in which one non- terminal gives another non terminal.[α-> β] and α, β ∈ V_N

*NOTE:-

1. Only variable to variable production is considered
2. small letter and more than one variable is production is not considered
3. Always remember there should produce same language before and after removal of unit vectors

E.g = S->AB ; A->a ; B->C|b ; C->D ; D->E ; E->a

Solution= so unit variables are :- B->C ; C->D ; D->E

B->C->D->E->a

That means we can replace all productions As

S->AB

A->a

B->a|b

C->a

D->a

E->a

Therefore , the new productions are S->AB

A->a

B->a|b

E.g= S->aA|B

A->ba|bb

B->A|bba

Solution:- the unit productions are S->B

B->A

S->B->A

That means we replace all productions are

S->a|bba|ba|bb

A->ba|bb

B->bba|ba|bb

Therfore , the new productions are S->a|bba|ba|bb

A->ba|bb

Practice Questions

Question1:- consider CFG and remove unit production

S->A|bb ; A->B|b ; B->S|a

Question2:- S->0A|1B|C A->0S|00 B->1|A C->01

Then remove the unit vector

#Normal Forms of CFG

1. Chomsky Normal Form (CNF)
2. Greiback Normal Form (GNF)

Chomsky Normal Form (CNF):-

A context free grammar is saod to be in CNF . If all its production rules are of types-

1. <Non terminal> -> < Terminal >
2. <Non terminal> -> <Non terminal ><Non Terminal>

E.g

- A->a ; A->AB
- S->BA|a ; B->a ; A->b
- S->AB|BD ; A->CD|a|AC ; B->b ; C->DE ; D->d ; E->f

Algorithm to convert CFG to CNF

Step1:- If start symbol occurs on RHS of production rule, create a new start symbol s' & a new production as a S'->S

Step2:- Remove null production.

Step3:- Remove unit production.

Step4:- If R.H.S pf production contains more than 2 variables then two consecutive variables can be replaced by a new variable .

E.g = A->XYZ => A->XP or A->PZ

P->YZ P->XY

Step5:- If R.H.S of production contains combination of terminal and variables then each terminal symbols is replaced by new variable

E.g= A->aB => A->XB

X->a

Question:- Convert equivalent CNf for given CFG

S->aSa|bSb|a|b

Solution:- Since S apperas in RHS , we add new production , we have:-

S'->S (not in CNF) S->aSa (not in CNF)

S->bSb (not in CNF) S->a (CNF)

S->b (CNF)

No null|unit production

We find a production S->aSa & S->bSb where RHS contain combination of variables and terminals apply step 5

We have S->AX And S->BY

X->SA Y->SB

A->a B->b

Find production set we obtain is as follows :-

{S'->AX|BY ; S->AX|BY|a|b

X->SA Y->SB

A->a B->b }

Practice Question

Question1:- convert the given CFG to CNF

S->aAD A->aB|bAB

B->b D->a

Question2:- convert CFG to CNF

S->aAbB

A->aA|a

B->bB|b

Greiback Normal form (GNF)

A context free grammar is said GNF if all its production rules are of type

1. <NON- Terminal> -> <Terminal>
2. <Variable> -> <single terminal> <variable>

S-> ε

E.g:- A->a

B->aBB|a

S->aAABBB

A->a

B->b

Algorithm to convert CFG into GNF, the 2 important lemma's are used :-

1. LEMMA 1:-

Let $G=(V_N , \Sigma, P,S)$ be a given CFG and if there is production

$A->B\ y$ and $B=B_1|B_2|B_3|.....| B_n$

$A->=B_1\ y|B_2\ y|B_3\ y|.....| B_n\ y,$

Where $y \in V_N^*$

Question:- Convert given CFG into GNF

S->AA

A->aA|bA|aAS|b

Solution:- By

let S->AA

A->aA | bA | aAS | b

B1 B2 B3 B4

The new GNf are,

S->aAA | bAA | aASA | bA

A->aA | bA | aAS | b

#PRACTICE QUESTIONS

Question 1:- Convert given CFG into GNF

S->CA ; A->a ; C-> aB|b

Question2:- Convert given CFG into GNF

S-> AB | BC , A->aB | bA |a ; B->bB |Cc| b ; C->c

2. LEMMA 2:-

Let $G=(V_N , \Sigma, P, S)$ be a given CFG and if there is production

$A->A\alpha_1 | A\alpha_2 | A\alpha_3 |-----| A\alpha_n | B_1|B_2|B_3|.....| B_n$

Such that B_i do not start with A then equivalent grammar in GNF can be –

$A-> B_1|B_2|B_3|.....| B_n$

$A-> B_1Z|B_2Z|B_3Z|.....| B_nZ$

$Z-> \alpha_1 | \alpha_2 | \alpha_3 |-----| \alpha_n$

$Z-> \alpha_1Z | \alpha_2Z | \alpha_3Z |-----| \alpha_nZ$

Question:- Convert the following grammar in GNf

S->AB|BC

A->aB|bA|a

B->bB|cC|b

C->c

Solution:-As starting symbol is S then but values one by one of A and B

S->AB

S->BC

A->aB|bA|a

B->bB|cC|b

C->c

Step1:- Take S->AB
Now replace productions of A in S
We get, S->aBB| bAB|aB

Step2:- Take S->BC
Now replace production of b in S
We get, S->bBC|cCC|bC
Therefore ,

S-> aBB| bAB|aB| bBC|cCC|bC
A->aB|bA|a
B->bB|cC|b
C->c

#Regular Grammar

A grammar is regular if it has rules or the form A->a or A->aB
Or A-> ε

Linear Grammar

A grammar is said to be linear if non termianl generates a single Non terminal on R.H.S
e.g.:- A->B
There are 2 types of linear grammar

1. Left linear grammar
2. Right linear grammar

Left Linear Grammar

A grammar is said to be left linear if the production contain a single non terminal on right hand side of production, which occurs at the left most of the string.
E.g= S->Sa ; S->Pb ; P->a all 3 are examples of left linear grammar

Right Linear Grammar

A grammar is said to be left linear if the production contain a single non terminal on right hand side of production, which occurs at the reft most of the string.

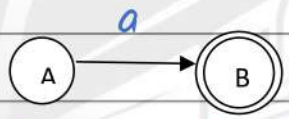
E.g= S->aS ; S->Pb ; P->a all 3 examples are right linear grammar

Conversion of finite automata to. Regular grammar or centext free grammar.

Rules are as follows:-

1. FA: 

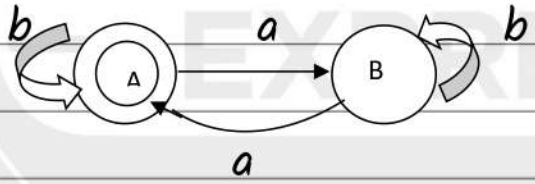
Production : A->aB

2. FA: 

Production: A->a ; A->aB

3. If initial state is final state , than add epsilon in a production

Question:- Convert the following Fa to RG



Solution:- A-> ε ; B->bB
A->bA ; B->aA
A->aB ; B->a
A->b

RG: A->bA|aB| ε |b
B->bB|aA|a

Regular Grammar: G=({A,B}, {a,b}, P,A)

Where, P: A->bA|aB| ε |b
B->bB|aA|a

Conversion of regular Grammar into Finite Automata:

Number of states in the automata will be equal to number of non-terminals +1 . Each state in automata represents each non-terminal in regular grammar. Additional state will be final state.

Rules are as follows:

1. For every production, A->aB make δ (A,a)=B
   [make a edge labelled 'a' from A to B]
2. For every production, A->a make δ (A,a)=final state
3. For every production, A->A or ε, make (δ, A,A)=A and A will be final state.

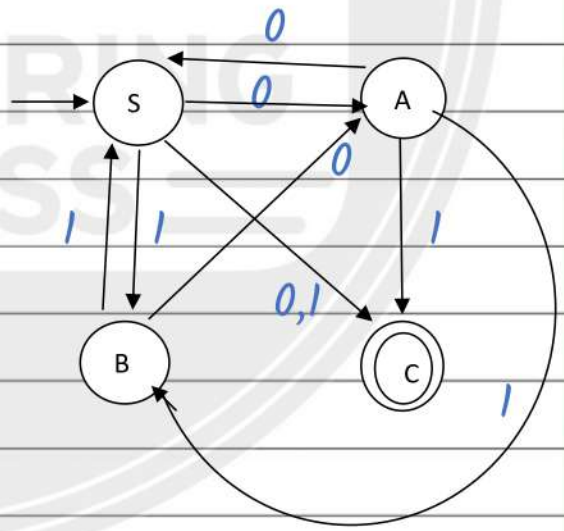
Question:- consider the following regular grammar

S->0A|1B|0|1 ;A->0S|1B|1 ;B->0A|1S design Finite Automata

Solution:- S->0A|1B|0|1

A->0S|1B|1

B->0A|1S



S->0 = δ (S,0)=A

S->1B= δ (S,1)=B

S->0= δ (S,0)= C

S->1= δ (S,1)=C

A->0S= δ (A,0)=S

A->1B= δ (A,1)=B

A->1= δ (A,1)=C

B->0A= δ (B,0)=A

B->1S= δ (B,1)=S

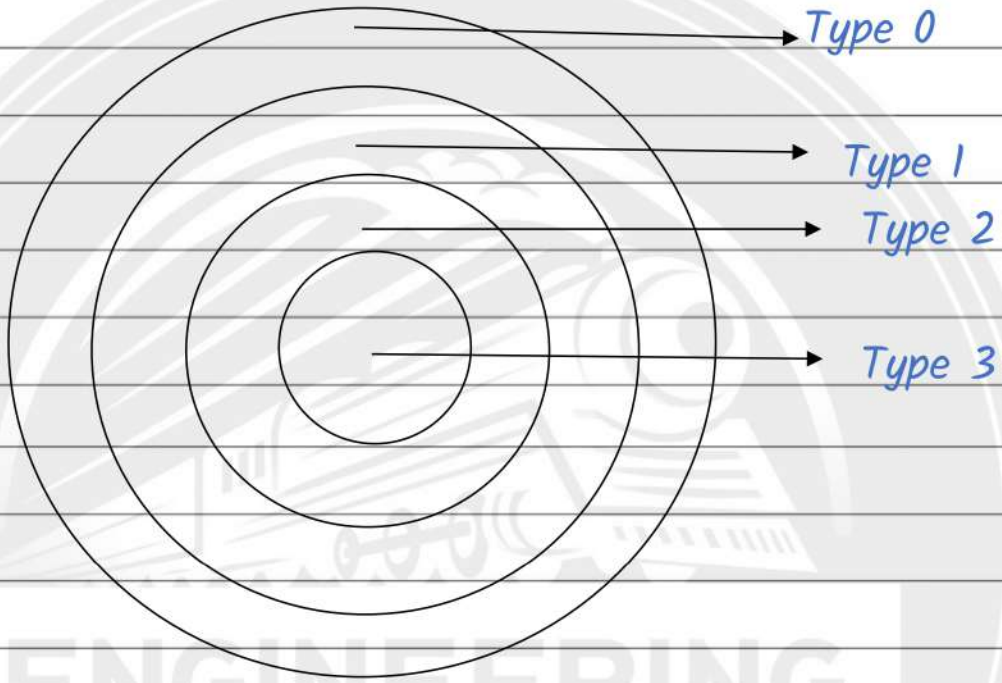
Practice Question

Question:- give the automata for the following RG

S->0S|1A|1 ; A->0A|1A|0|1

#Chomsky's Hierarchy Of Grammar

Noam Chomsky has classified grammar in 4 categories:- [type 0 , type 1 , type 2 and type 3] Based on the R.H.S forms of the productions . a grammar may be subset or super set of another Grammar as per its type . A higher type grammar is superst of all lower type grammar.



TYPE 0:-

A type 0 Grmmar is any phase structure grammar without any restrictions. All grammar are type 0 grammar . and all language are type 0 language . A production without any restriction is called a type 0 production. Type 0 language is accepted by turning machine.

$$\emptyset \cup \emptyset = \emptyset \propto \Psi$$

Where, 'A' is variable , $\emptyset$ is called left contect and Ψ is right context and [$\emptyset \propto \Psi$] the replacement string .

E.g= abAbcd->abABbcd	E.g= AC ε -> A ε ε
Solution:- ab->left context	Solution:- A->left context
Bcd->right context	ε ->right context
∝ ->AB	∝ -> ε

TYPE 1:-

Restricted type zero grammar are known as type 1 or context sensitive grammar. CSG if all productions of a grammar are of type one production, then the grammar is known as type 1 grammar. The language generated by a context sensitive grammar is called CSL. Type one language is accepted. By linear bounded automaton. LB, A, a production of the form.

$$\emptyset \cap \emptyset = \emptyset \propto \Psi$$

Is called type 1 production. If $\propto \neq \varepsilon$

1. aABCD->abcDbcD is a type 1 production where a , bcD are the left context and right context respectively. A is replaced by bcD$\neq \varepsilon$
2. AB ε -> AbBc ε

   $\propto$ = bBC$\neq \varepsilon$

   $\emptyset$ =A

   $\Psi = \varepsilon$

E.g:- S->aA| ε is allowed

S->aAS| ε is not allowed

TYPE 2:-

A restricted Type 1 grammar are known as type 2 grammar. A type2 production is a production of the form A-> $\propto$ where A $\in V_N$ And $\propto \in (V_N \cup \Sigma)^*$ in other words , the LHS of production has no left or right context

E.g= S->Aa ; A->a ; B->abc ;A-> ε are type 2 production

A grammar is called type 2 grammar if it contains only type 2 productions it is also called the context free grammar . push down automata can be used to represent these language.

TYPE 3:-

This is the most restricted type. A grammar is called a type 3 or regular grammar . if all its production are type 3productions. A production S-> ε is allowed in type 3 grammar. But in this case S does not appear on the RHS of any production. A production of the form A->a or A->aB;

Where, $A,B \in V_N$ (variable) and a,b are terminals are called type 3 production

Thank you




Engineering Express: Simplifying Engineering Education

Are you drowning in complex engineering concepts? Struggling to decipher cryptic notes? Fear not! Welcome to Engineering Express, where we decode the mysteries of engineering subjects for you. 🚀

What We Offer:

Clear Explanations: Our notes break down intricate topics into bite-sized, easy-to-understand language. No jargon, no headaches!

Real-Life Connections: Ever wondered how those abstract formulas apply in the real world? We've got you covered with relatable examples.

Maximize Your Marks: Our mission? Boost your exam scores. With our straightforward notes, you'll be well-prepared to conquer any engineering challenge.

Follow us on social media:

YouTube - https://www.youtube.com/@EngineeringExpress.?sub_confirmation=1

WhatsApp - https://chat.whatsapp.com/H16tpUl2SmQ3o6vlcD2D1u

Instagram - https://www.instagram.com/engineeringexpress2312/

Telegram - https://t.me/engineeringexpressofficial

Join the Engineering Express community today and unlock the power of simplified learning! 🎓🔥

THEORY OF AUTOMATA AND FORMAL LANGUAGE (BCS-402)

AKTU New Session (2024-25)

SIMPLE AND EASY EXPLANATION + HANDWRITTEN NOTES PDF

ALL UNIT-4

TOPIC

Push Down Automata and Properties of Context Free Languages: Nondeterministic Pushdown Automata (NPDA)- Definition, Moves, A Language Accepted by NPDA, Deterministic Pushdown Automata (DPDA) and Deterministic Context free Languages (DCFL), Pushdown Automata for Context Free Languages, Context Free grammars for Pushdown Automata, Two stack Pushdown Automata, Pumping Lemma for CFL, Closure properties of CFL, Decision Problems of CFL, Programming problems based on the properties of CFLs.

Engineering Express

Unit	Theory of Automata and Formal Languages
I	**Basic Concepts and Automata Theory:** Introduction to Theory of Computation- Automata, Computability and Complexity, Alphabet, Symbol, String, Formal Languages, Deterministic Finite Automaton (DFA)- Definition, Representation, Acceptability of a String and Language, Non Deterministic Finite Automaton (NFA), Equivalence of DFA and NFA, NFA with ε-Transition, Equivalence of NFA's with and without ε-Transition, Finite Automata with output- Moore Machine, Mealy Machine, Equivalence of Moore and Mealy Machine, Minimization of Finite Automata.
II	**Regular Expressions and Languages:** Regular Expressions, Transition Graph, Kleen's Theorem, Finite Automata and Regular Expression- Arden's theorem, Algebraic Method Using Arden's Theorem, Regular and Non-Regular Languages- Closure properties of Regular Languages, Pigeonhole Principle, Pumping Lemma, Application of Pumping Lemma, Decidability- Decision properties, Finite Automata and Regular Languages
III	**Regular and Non-Regular Grammars**: Context Free Grammar(CFG)-Definition, Derivations, Languages, Derivation Trees and Ambiguity, Regular Grammars-Right Linear and Left Linear grammars, Conversion of FA into CFG and Regular grammar into FA, Simplification of CFG, Normal Forms- Chomsky Normal Form(CNF), Greibach Normal Form (GNF), Chomsky Hierarchy, Programming problems based on the properties of CFGs.
IV	**Push Down Automata and Properties of Context Free Languages**: Nondeterministic Pushdown Automata (NPDA)- Definition, Moves, A Language Accepted by NPDA, Deterministic Pushdown Automata(DPDA) and Deterministic Context free Languages(DCFL), Pushdown Automata for Context Free Languages, Context Free grammars for Pushdown Automata, Two stack Pushdown Automata, Pumping Lemma for CFL, Closure properties of CFL, Decision Problems of CFL, Programming problems based on the properties of CFLs.
V	**Turing Machines and Recursive Function Theory** : Basic Turing Machine Model, Representation of Turing Machines, Language Acceptability of Turing Machines, Techniques for Turing Machine Construction, Modifications of Turing Machine, Turing Machine as Computer of Integer Functions, Universal Turing machine, Linear Bounded Automata, Church's Thesis, Recursive and Recursively Enumerable language, Halting Problem, Post's Correspondance Problem, Introduction to Recursive Function Theory.

UNIT - 4

#PUSH DOWN AUTOMATA AND PROPERTIES OF CFL

#Push Down Automata(PDA)

1. PDA are the enhanced version of finite automata.
2. It has Additional stack that can be used to store and retrive information,which makes it more powerful than a regular finite automata.
3. PDA reads the input string and changes the stack based on the current input and state.PDA accept the CFL.
4. The langauge recognized by PDA is the set of string that can be accepted by the PDA.
5. PDA can push symbols On to the stack. Pop symbol of the stack and read the top symbol of the stack without removing it.
6. In PDA, the stack uses three basic operations, push, pop and skip.
7. The push operation adds a symbol onto the top of the stack while. Pop operation removes the top symbol from the stack.
8. The skip operation does not modify the stack and allows the PDA to move to the next symbol in the input without performing any stack operation.
9. PDA is defined by seven tuples.-[Q, Σ, δ, T, q_0, Z_0, F]

Where, Q:- is a set of state.

Σ: - Is a set of input symbols.

T:- Is finite and non empty set of PD symbol

δ: - Is the transition function, which maps

For DPDA:-

$$\delta: Q \times \Sigma \cup \times T \rightarrow Q \times T^*$$

Or

$$\delta: Q \times (\Sigma \cup \{\varepsilon\}) \times T \rightarrow Q \times T^*$$

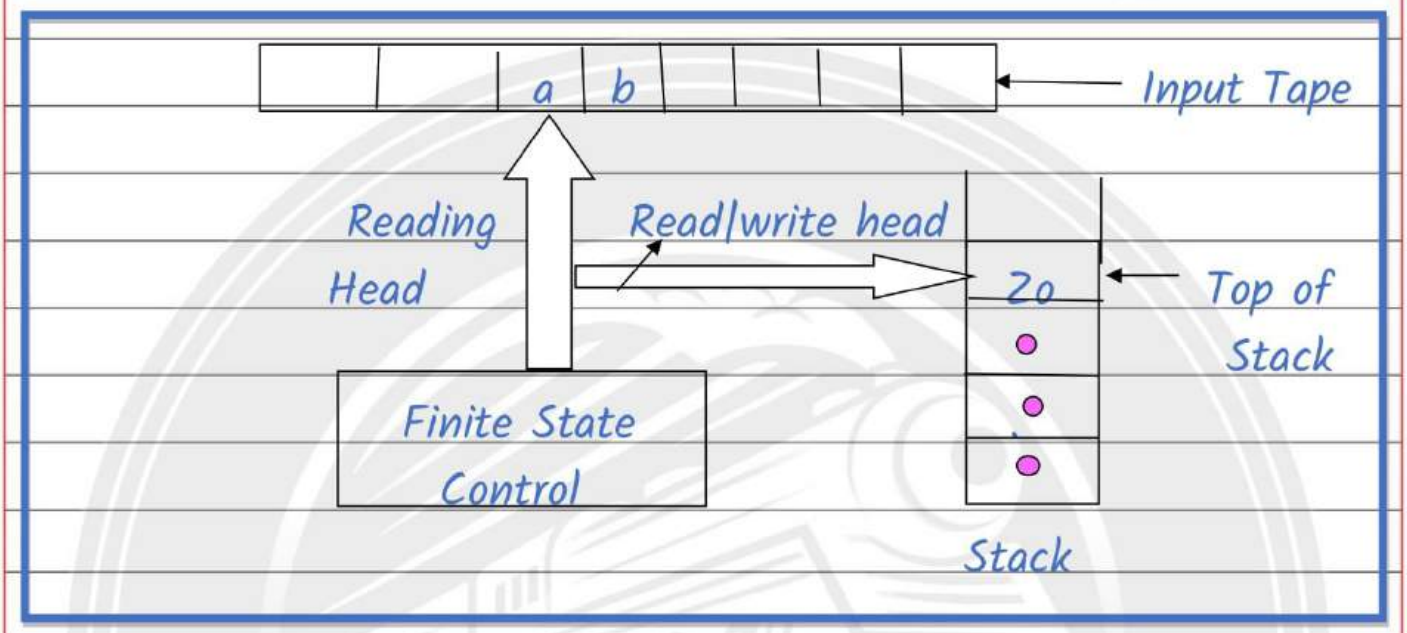
For NDPDA:-

$\delta: Q \times \Sigma \times T \rightarrow 2^{Q \times T^*}$

q_0:- Is the starting state

$Z_0 \in T$ Is the starting (topmost or initial stack symbol)

$F \subseteq Q$ is the set of final states and sometimes $F = \emptyset$



#Comparison b\w PDA and FA

PDA	FA
The PDA can be defined as a collection of 7 tuples -[Q, Σ, δ, T, q_0, Z_0, F].	The FA can be defined as a collection of 5 tuples [Q, Σ, δ, q_0, F].
PDA can accept both Regular Language and CFL.	FA can not accept CFL. FA can accept only Regular language.
PDA has a Stack which is used To remember the I/P.	FA has no stack Storage.
PDA is much more superior than FA.	FA is less powerful than PDA.

Deterministic Push down automata[DPDA]

The model should be deterministic in the sense that it should not show more path moving to different state from the same. Input. In fact, for each specific input, there is only one specific part, like as. $L=a^nb^n$

The concept of DPDA is similar to DFA.

Nondeterministic pushdown automata.[NDPDA]

The non DPDA is very similar to NFA.

The CFG, which accepts DPDA, accepts NDPDA as well.

Similarly, there are some CFG which can be accepted by NDPDA.

#Instantaneous Description[ID]

It is an informal notation of how a pda computes a input stream and make a decision that string is accepted or rejected That is, tells the condition of a machine at particular situation.

An ID is a triplet[q,w, α]

Where,

q:- it is a current state[$q \in Q$]

w:- it is the remaing input $W \in \Sigma^*$,

α: - is the stack contents $\alpha \in T^*$

Turnstile Notation

$\vdash$ Sign is called turnstile notation and represent one move

$\vdash^*$ Sign represents a sequence of moves

Basic Operations on stack

1. PUSH OPERATION

$$\delta(q_0, a, Z_0) \vdash (q_0, aZ_0)$$

This Implies that while taking the transition from state q_0 to state q_0, the input symbol a is consumed, and the top of the stack Z_0 is replaced or pushed by a new string. aZ_0

2. POP OPERATION

$\delta (q_0, a, c) \vdash (q, \varepsilon)$

This implies that we have popped a from the stack.

3. SKIP OPERATION

$\delta (q_1, a, Z_0) \vdash (q_1, Z_0)$

This implies read a from the input tape is skiped from the stack

Question1:-design PDA for language $L=\{a^n b^n \mid n>=1\}$, $\Sigma = \{a,b\}$

Solution:- language L =(ab, aabb, aaabbb.....)

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

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

$\delta (q_0, b, a)=(q_1, \varepsilon)$ {it means top element is poped}

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

$\delta (q_1, \varepsilon, z_0)=(q_f, \varepsilon)$

Language of PDA

There are 2 kinds of possible acceptance by PDA for any string

a) Acceptance by final state

b) Acceptance by empty stack

1. Acceptance by final state.

The pta is said to be except its input by the final state, if it enters any final state in zero or more moves after reading the entire input.

Let P:-[Q, Σ, δ, T, q_0, Z_0, F] be a PDA. The language acceptable by the final state can be defined as:-

$L(PDA)=\{W \mid (q_0,W,Z) \vdash^* (q, \varepsilon, Z\}, q \in F$

2. Acceptance by empty stack

On reading the input stream from the initial configuration for some PDA, the stack of PDA gets empty.

Let P:-[Q, Σ, δ, T, q_0, Z_0, F] be a PDA. The language acceptable by the empty stack can be defined as:-

$N(PDA)=\{W|(q_0,W,Z) \vdash^* (p, \varepsilon, \varepsilon\},q \in Q$

Equivalence of acceptance by final state and empty stack

1. If L=N(PI) for some PDA PI , Then there is the PDA, P2 such that L=L(P2). That means the language accepted by empty stack pda will also be accepted by final state PD.

2. If there is a language L=L(PI)for some PDA PI then there is a PDA P2 such that L=N(P2). That means Language accepted by final State PDA is also accepted by empty stack PDA

Question1:-Design a PDA that accepts equal number of a's and b's in a string over Σ ={a,b}

Solution:- language L =(ab, aabb, aaabbb.....)

$\delta (q_o, a, Z_o)=(q_0, aZ_o)$

$\delta (q_o, a, a)=(q_0, aa)$

$\delta (q_o, b, a)=(q_1, \varepsilon)$ {it means top element is poped}

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

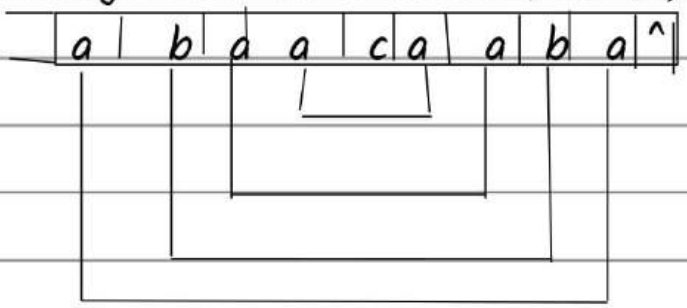
Acceptance :-

$\delta (q_1, \varepsilon, z0)=(q_f, \varepsilon)$ for using final state

$\delta (q_1, \varepsilon, z0)=(q_0, \varepsilon)$ for using null stack

Question2:-Design PDA for $L=\{WcW^T|W \in (a,b)^*\}$

Solution:-



$\delta(q_0, a, Z_0) = (q_0, aZ_0)$
$\delta(q_0, a, Z_0) = (q_0, aZ_0)$
$\delta(q_0, a, Z_0) = (q_0, aZ_0)$
$\delta(q_0, a, a) = (q_0, aa)$
$\delta(q_0, a, b) = (q_0, ab)$
$\delta(q_0, b, Z_0) = (q_0, bZ_0)$
$\delta(q_0, b, b) = (q_0, bb)$
$\delta(q_0, b, a) = (q_0, ba)$
$\delta(q_0, c, Z_0) = (q_1, Z_0)$
$\delta(q_0, c, a) = (q_1, a)$
$\delta(q_0, c, b) = (q_1, b)$

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

$\delta(q_0, b, b) = (q_1, \wedge)$
{using null stack}

$\delta(q_0, b, b) = (q_1, \wedge)$
{using final state}

Convert CFG to PDA

Step1:- Convert the given CFG into Greibach normal form. GNF.

Step2:- Push the start symbol S of the CFG into the stack using transition ID

$$\delta(q_0, a, Z_0) \vdash (q_0, aZ_0)$$

Step3:- For a production rule in the form of.

$(NT_1) \rightarrow$ (single T)(String of NT)

ID will be : $\delta(q_1, T, NT) \rightarrow (q_1,$ string of NT)

Step4:- for a production rule in the form of

$(NT_1) \rightarrow$ (single T)

ID will be : $\delta(q_1, T, NT_1) \rightarrow (q_1, \wedge)$

Step5:- Lastly, add the transition id for accepting the string, either by empty stack or final step.

Question1:- S->aA1 A2 | a , A1->bA2|b , A2->bA2A3|b , A3->c
Convert the given CFG into its equivalent pda. Also show the transion IDS for the string 'abbbcb' for PDA

Solution:-

Sol" Understand the grammar by the following IDs

$\delta(q,\wedge,S) = \{(q,a)(q,aA_1,A_2)\}$
$\delta(q,\wedge,A_1) = \{(q,bA_2)(q,b)\}$
$\delta(q,\wedge,A_2) = \{(q,bA_2A_3)(q,b)\}$
$\delta(q,\wedge,A_3) = \{(q,c)\}$
$\delta(q,a,a) = (q,\wedge)$
$\delta(q,b,b) = (q,\wedge)$
$\delta(q,c,c) = (q,\wedge)$

Note:- we always use leftmost derivation

Now, we seperate terminal and non-terminal symbol. we take terminal symbol as input symbol & non-terminal symbol as stack symbols

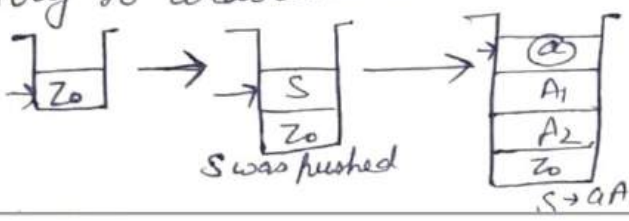
Thus, the transition IDS are:

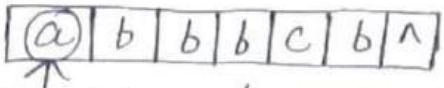
$\delta(q_0,\wedge,z_0) = (q_1, S z_0)$
$\delta(q_1,\wedge,S) = (q_1, A_1A_2)$
$\delta(q_1,a,S) = (q_1,\wedge)$
$\delta(q_1,b,A_1) = (q_1,A_2)$
$\delta(q_1,b,A_1) = (q_1,\wedge)$
$\delta(q_1,b,A_2) = (q_1,A_2A_3)$
$\delta(q_1,b,A_2) = (q_1,\wedge)$
$\delta(q_1,c,A_3) = (q_1,\wedge)$
$\delta(q_1,\wedge,z_0) = (q_0,\wedge)$

LMD for I/P "abbbcb"

$S \to aA_1A_2$
$\to abA_2A_2$
$\to abbA_2A_3A_2$
$\to abbbA_3A_2$
$\to abbbcA_2$
$\to abbbcb$

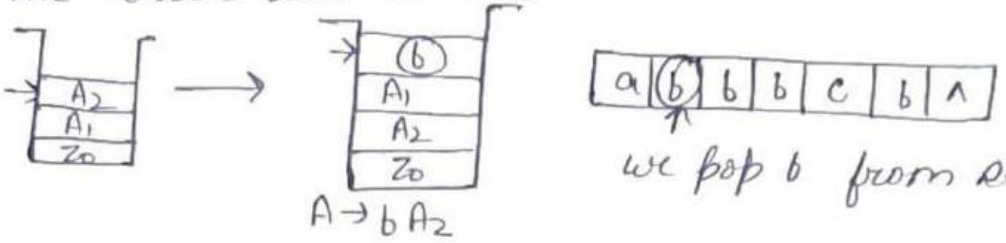
Try to understand the mechanism by:



at this part 

I/P string a is matched with top value of stack, so we will pop it.

Thus the stack will become



And this procedure will go on, until the input symbol from the input string are read & the stack becomes empty which indicates the string has been accepted

IDs are :-

$\delta(q_0, \wedge, Z_0) = (q_1, SZ_0)$

$\delta(q_1, abbbcb, S) = (q_1, \overset{\downarrow}{a}bbbcb, \overset{\downarrow}{a}A_1A_3)$

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

$\delta(q_1, bbbcb, A_1A_2) = (q_1, \overset{\downarrow}{b}bbCb, \overset{\downarrow}{b}A_2A_2)$

$\delta(q_1, b, b) = (q_1, \wedge)$

$\delta(q_1, bbcb, A_2A_2) = (q_1, \overset{\downarrow}{b}bcb, \overset{\downarrow}{b}A_2A_3A_2)$

$(q_1, \overset{\downarrow}{b}cb, A_2A_3A_2) = (q_1, bcb, \overset{\downarrow}{b}A_3A_2)$

$(q_1, \overset{\downarrow}{c}b, A_3A_2) = (q_1, cb, \overset{\downarrow}{c}A_2)$

$(q_1, c, c) = (q_1, \wedge)$

$(q_1, b, A_2) = (q_1, b, b)$

$(q_1, \wedge, Z_0) = (q_1, \wedge)$

Convert PDA to CFG

The production in PER induced by transition Ids of the PDA as follows.

1. S production are given by

   $S \rightarrow [q0, Z0, q]$ for every $q \in Q$

2. For every poppinf move

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

Induced a production $[q\ Z\ q'] \rightarrow a$

3. For every push move $\delta(q, a, Z) = (q1, Z_1, Z_2, \ldots Z_n)$

Induced production

$$[q\ Z\ q'] \rightarrow a\ [q_1\ Z_1\ q_2]\ [q_2\ Z_1\ q_3]\ [q_3\ Z_1\ q_4] \ldots\ldots [q_n\ Z_n\ q']$$

Where each state $q', q_1, q_2, \ldots q_m$ can be any state in PDA

Question1:-Generate a PDA for the PDA M, defined as

$M = (\{q0, q1\}\ \{0,1\}\ \{a, Z0\}\ q0, \delta, Z0, q1)$

Where is defined as :-

1. $\delta(q0, 1, Z0) = (q0, xZ0)$
2. $\delta(q0, 1, x) = (q0, xx)$
3. $\delta(q0, 0, x) = (q0, x)$
4. $\delta(q0, \hat{}, x) = (q1, \hat{})$
5. $\delta(q0, \hat{}, x) = (q1, \hat{})$
6. $\delta(q1, 0, x) = (q1, xx)$
7. $\delta(q1, 0, Z0) = (q1, \hat{})$

Solution:-

Soln:- firstly induced the S production for every $q \in Q$

$\Rightarrow S \to [q_0 z_0 q_0] / [q_0 z_0 q_1]$

* Now for 1st transition ID (Push move)

$\delta(q_0, 1, z_0) = (q_0, \underbrace{x, z_0}_{\text{4 production}})$

$[q_0 z_0 q_0] \to 1 [q_0 x q_0] [q_0 z_0 q_0]$
$[q_0 z_0 q_0] \to 1 [q_0 x q_1] [q_1 z_0 q_0]$
$[q_0 z_0 q_1] \to 1 [q_0 x q_0] [q_0 z_0 q_1]$
$[q_0 z_0 q_1] \to 1 [q_0 x q_1] [q_1 z_0 q_1]$

* for 2nd transition ID (Push move)

$\delta(q_0, 1, x) = (q_0, \underbrace{xx}_{\text{4 production}})$

$[q_0 x q_0] \to 1 [q_0 x q_0] [q_0 x q_0]$
$[q_0 x q_0] \to 1 [q_0 x q_1] [q_1 x q_0]$
$[q_0 x q_1] \to 1 [q_0 x q_0] [q_0 x q_1]$
$[q_0 x q_1] \to 1 [q_0 x q_1] [q_1 x q_1]$

* for 3rd transition ID (Push move)

$\delta(q_0, 0, x) = (q_0, \underbrace{x}_{2^1 = 2 \text{ production}})$

$[q_0 x q_0] \to 0 [q_0 x q_0]$
$[q_0 x q_1] \to 0 [q_0 x q_1]$

* for 4th transition ID (Pop move)

$\delta(q_0, \wedge, x) = (q_1, \wedge)$

$[q_0 x q_1] \to \wedge$

* for 5th transition ID (POP move)

$\delta(q_1, \wedge, x) \Rightarrow (q_1, \wedge)$

$[q_1 x q_1] \to \wedge$

* for 6th transitionID(push move)

$$\delta(q_1, 0, x) = (q_1, \underbrace{x\ x}_{2})$$

$2^2 = 4$ production

$$[q_1\ x\ q_0] \to 0\,[q_1\ x\ q_0]\,[q_0\ x\ q_0]$$
$$[q_1\ x\ q_0] \to 0\,[q_1\ x\ q_1]\,[q_1\ x\ q_0]$$
$$[q_1\ x\ q_1] \to 0\,[q_1\ x\ q_0]\,[q_0\ x\ q_1]$$
$$[q_1\ x\ q_1] \to 0\,[q_1\ x\ q_1]\,[q_1\ x\ q_1]$$

* for 7th transitionID(pop move)

$$\delta(q_1, 0, z_0) = (q_1, \wedge)$$

$$[q_1\ z_0\ q_1] \to 0$$

Further, the production can be simplified by removing null production, useless production, or unit production.

Difference btw DPDA and NDPDA

DPDA	NDPDA
1. It is less powerful than NDPDA. Example we can only construct dpda for odd length palindromes and not for even length paladrops.	1. It is more powerful than dpta example, ndpda can be constructed for both even length and odd length. Palindromes,
2. It is possible to convert every DPDA to a corresponding NDPDA.	2. It is not possible to convert every NDPDA to a corresponding DPDA.
3. The language accepted by DPDA is a subset of the language accepted by NDPDA	3. The language accepted by NDPDA is not a subset of the language accepted by DPDA
4. The language accepted by DPDA is called DCFL[Deterministic context free language], which is a subset of NCFL[Nondeterministic context free language] accepted by NDPDA.	4.The language accepted by NPDA is called NCFL, nondeterministic context free language.
5. There is only one state transition from one state to another state for an input symbol.	4. There may or may not be more than one state transition from one state to another state for same input symbol.

Pumping Lemma for CFLs

It is used to prove that a language is not context free.

If a is a context free language, then a has a pumping length P such that any string S where |s|>=P may Be divided into five pieces.

S=uvxyz such that the following conditions must be true:

1. $u\ v^i\ x\ y^i\ z$ is in A for every $i>=0$
2. $|v\ y|>0$
3. $|v\ x\ y|<=P$

Ques:- Show that $L=\{a^n b^n c^n / n \geq 0\}$ is not context free

Soln:- Assume that L is context free. So, it must be having a pumping length (P) such that $|S| \geq P$ & $S=a^P b^P c^P$. Now we divide S into 5 parts u v x y z

Ex:- P=4 So, $S=a^4 b^4 c^4$

Case 1:- v & y each contain only one type of symbol S = aaaa bbbb cccc (u v x y z)

Now if $i=2 \Rightarrow u v^i x y^i z$

$\Rightarrow$ a aaaa abbbbc ccccc $\notin L$

$|a|=6$, $|b|=4$, $|c|=5$

$\Rightarrow |a| \neq |b| \neq |c|$

Thus this case does not belong to given language.

Case 2:- Either v or y has more than one kind of symbol

aa aa bbbb cccc (u v x y z)

if $i=2 \Rightarrow u v^i x y^i z$

$\Rightarrow$ aa aabb aabbbbbb cccc $\notin L$

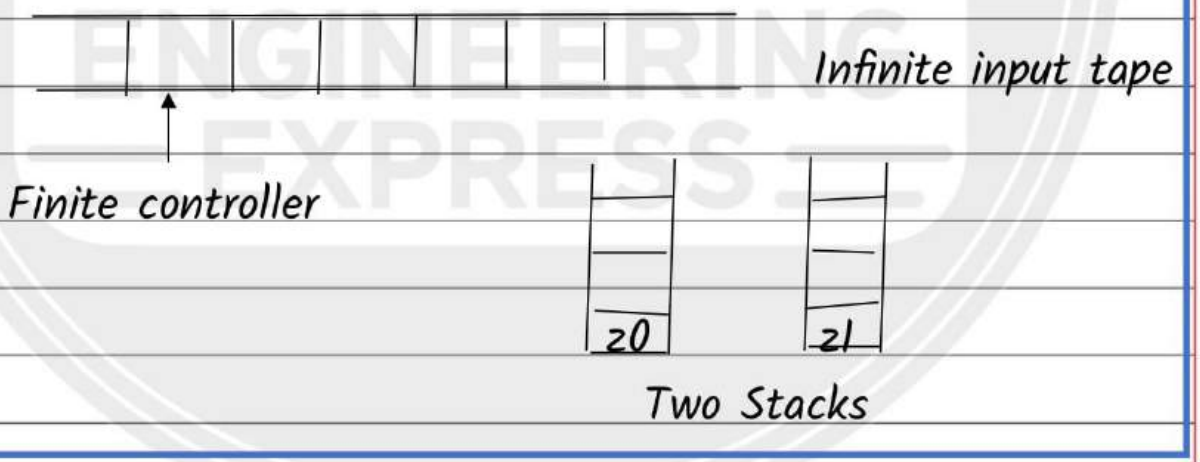
Thus our assumption that L is a context free language does not hold. Thus, the given language is not context free language.

Two Stack PDA:-

Definition:- A Two stack PDA is a 6 tuples($Q, \Sigma', \delta, T, q_0, F$), where Q, Σ', T, q_0, F are the same as one stack PDA but the transition function is different, which is as follows: -

$$\delta: Q \times (\Sigma \cup \{\varepsilon\}) \times T \times T \rightarrow Q \times T^* \times T^*$$

1. It is similar to a PDA, but it has two stacks instead of one.
2. A machine using two stacks, then push down automata, except the recursively ennumerable language.|Turing machine
3. Two stackpad is a variant for turing machine. It is a computational model which is based on the generalization of PDA
4. The moves of the two stack pda are based on:-
   - ➔ The state of final control.
   - ➔ The input symbol that read.
   - ➔ The top of stack symbol on each of its stack.



Examples :- $L=\{a^n b^n c^n | n>=1\}$

Early early proof that this language L is not context-free language, but this language can be solved through two stack pdf.

$L=\{abc, aabbcc, aaabbbccc.....\}$

Logic:-

1. If A is read push a in stack one. No change in step two.

2. If B is read, pop a from stack one and push B into stack 2. This ensure number of a equals to number of b.
3. If she is read, pop be from stack 2. This ensures number of b equals to c

$\delta(q0, a, Z0,Z1)=(q0, aZ0, Z1)$

$\delta(q1, a, a,Z1)=(q1, aa, Z1)$

$\delta(q1, b, a,Z1)=(q2, ^, bZ1)$

$\delta(q2, b, a,b)=(q2, ^, bb)$

$\delta(q2, c, Z0,b)=(q3, Z0, ^)$

$\delta(q3, ^, Z0,Z1)=(qf, ^, ^)$

Thank you

Join us and subscribe our Chenal

THEORY OF AUTOMATA AND FORMAL LANGUAGE (BCS-402)

AKTU New Session (2024-25)

SIMPLE AND EASY EXPLANATION + HANDWRITTEN NOTES PDF

ALL UNIT-5

TOPIC

Turing Machines and Recursive Function Theory : Basic Turing Machine Model, Representation of Turing Machines, Language Acceptability of Turing Machines, Techniques for Turing Machine Construction, Modifications of Turing Machine, Turing Machine as Computer of Integer Functions, Universal Turing machine, Linear Bounded Automata, Church's Thesis, Recursive and Recursively Enumerable language, Halting Problem, Post's Correspondance Problem, Introduction to Recursive Function Theory.

Engineering Express

Unit	Theory of Automata and Formal Languages
I	**Basic Concepts and Automata Theory:** Introduction to Theory of Computation- Automata, Computability and Complexity, Alphabet, Symbol, String, Formal Languages, Deterministic Finite Automaton (DFA)- Definition, Representation, Acceptability of a String and Language, Non Deterministic Finite Automaton (NFA), Equivalence of DFA and NFA, NFA with ε-Transition, Equivalence of NFA's with and without ε-Transition, Finite Automata with output- Moore Machine, Mealy Machine, Equivalence of Moore and Mealy Machine, Minimization of Finite Automata.
II	**Regular Expressions and Languages:** Regular Expressions, Transition Graph, Kleen's Theorem, Finite Automata and Regular Expression- Arden's theorem, Algebraic Method Using Arden's Theorem, Regular and Non-Regular Languages- Closure properties of Regular Languages, Pigeonhole Principle, Pumping Lemma, Application of Pumping Lemma, Decidability- Decision properties, Finite Automata and Regular Languages
III	**Regular and Non-Regular Grammars**: Context Free Grammar(CFG)-Definition, Derivations, Languages, Derivation Trees and Ambiguity, Regular Grammars-Right Linear and Left Linear grammars, Conversion of FA into CFG and Regular grammar into FA, Simplification of CFG, Normal Forms- Chomsky Normal Form(CNF), Greibach Normal Form (GNF), Chomsky Hierarchy, Programming problems based on the properties of CFGs.
IV	**Push Down Automata and Properties of Context Free Languages**: Nondeterministic Pushdown Automata (NPDA)- Definition, Moves, A Language Accepted by NPDA, Deterministic Pushdown Automata(DPDA) and Deterministic Context free Languages(DCFL), Pushdown Automata for Context Free Languages, Context Free grammars for Pushdown Automata, Two stack Pushdown Automata, Pumping Lemma for CFL, Closure properties of CFL, Decision Problems of CFL, Programming problems based on the properties of CFLs.
V	**Turing Machines and Recursive Function Theory** : Basic Turing Machine Model, Representation of Turing Machines, Language Acceptability of Turing Machines, Techniques for Turing Machine Construction, Modifications of Turing Machine, Turing Machine as Computer of Integer Functions, Universal Turing machine, Linear Bounded Automata, Church's Thesis, Recursive and Recursively Enumerable language, Halting Problem, Post's Correspondance Problem, Introduction to Recursive Function Theory.

UNIT – 5

TURING MACHINE AND RECURSIVE FUNCTION THEORY

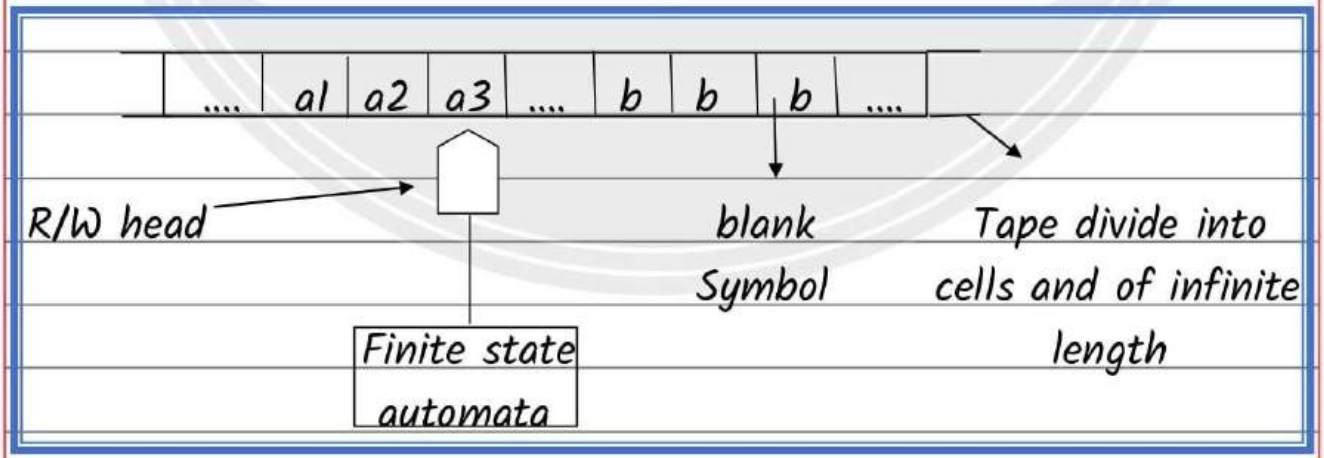
Turing machine

The turing machine is a generalized machine which can recognize all type of language. Regular language, contextfree language and context sensitive language. Apart from these languages, the turing machine also accept the language generated from unrestricted grammar, thus turing machine can accept any generalized language. This chapter mainly concentrates on building a turing machine for any language.

Turing machine model

The turing machine model is shown in below figure. It is a finite automaton connected to read, write head with the following components.

1. Tape
2. Read- write head
3. Control unit



Tape:- It is a temporary storage, and it is divided into cells. Each cell can store the information of only one symbol.

Read- Write head:- The read-write head can read a symbol from where it is pointing to, and it can write into the tape to where the read-write head points do.

Control unit:- The reading /writing form/ to the tape is determined by the control unit. The different move performed by the machine depends on the current scan symbol and the current state.

Mathematically ,

Definition:- Turing machine is defined as 7 tuples

$$[Q, \Sigma, \delta, T, q_0, b, F]$$

Where ,

- Qis a finite , non empty set of states
- Σ Is a non empty set of input symbols and is a subset of T And $b \in \Sigma$
- T is a finite Non empty set of tape symbols.
- $b \in T$ Is the blank
- δ Is the transition function,which maps

$$Q \times T \rightarrow Q \times T \times \{L, R\}$$

- q_0 Is thr initial state
- $F \in Q$Is the final state

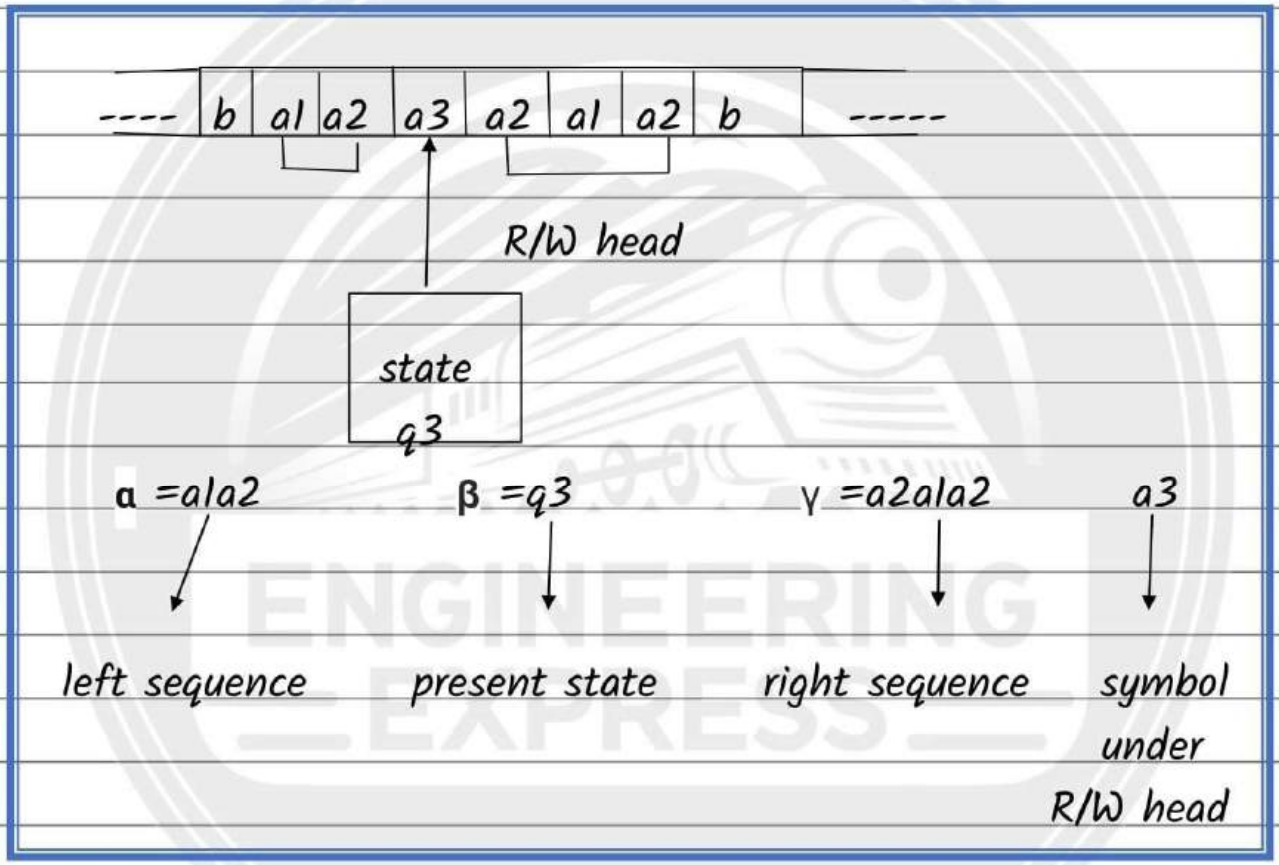
Representation od turing machine

Generally, we have three techniques to represent turing machine.

1. Representation by instantaneous description.
2. Representation by a transition table.
3. Representation by transition diagram.

Representation by Instantaneous Description.

An instantaneous description of turing machine is defined in terms of the entity input strain and the current state. And instantaneous description of tuning machine is. defined as a string $\alpha\beta\gamma$
Where, β is the present state of turing machine and The entire input strain is split as $\alpha\gamma$ in which α is the processed string,while γ is the remaining string to be scanned.



Moves in turing machine

$$\delta(q1, x_3) = (q2, y, R)$$

Representation by Transition Table

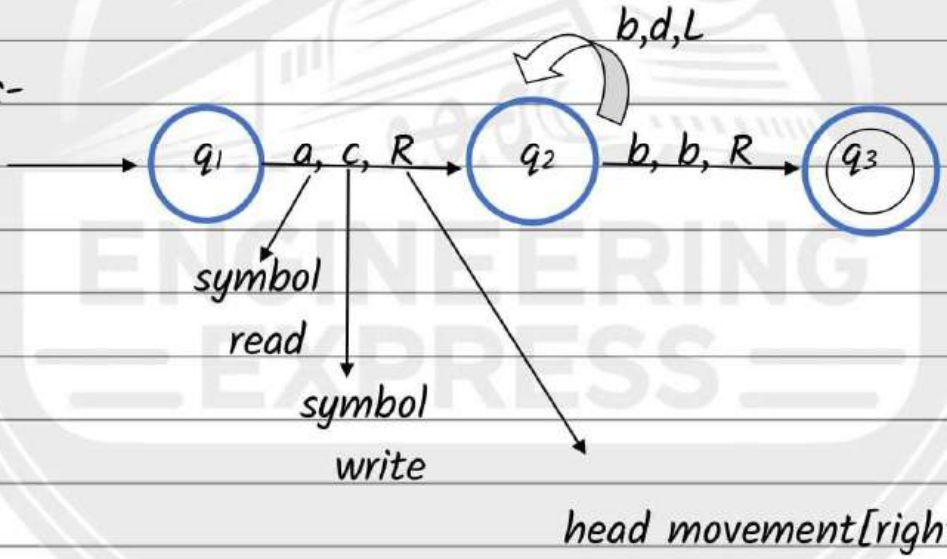
If $\delta(q,a) \vdash (\alpha, \beta, \gamma)$, We write $\alpha\beta\gamma$ under a column and q-row. So if we get $\alpha\beta\gamma$ in the table, it means that α is written in the current cell,β gives the movement of the head(L or R) and γ denotes the new state into which the turing machine enters

Present State	type symbol		
	b	0	1
→ q1	1Lq2	0Rq1	-
q2	bRq3	oLq2	1Lq2
q3	-	bRq4	bRq5
q4	0Rq5	0Rq4	1Rq4
*q5	0Lq2	-	1Rq4

Representation by Transition Diagram

1. Turing machine can also be represented using directed graph
2. Vertices are used to represent states
3. 3. edges are used to represent transitions

Example:-



language Acceptability by Turing machine(TM)

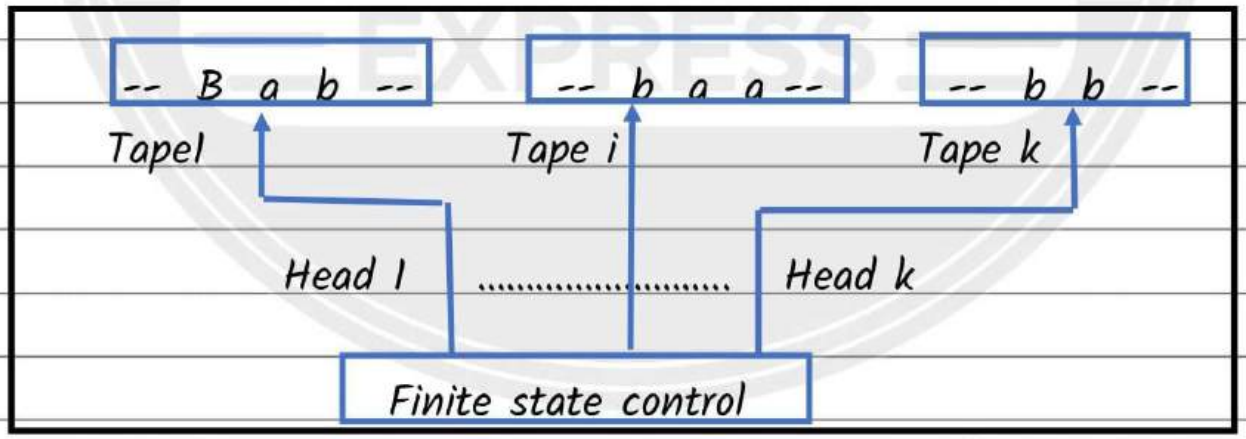
1. A string w of a language L is said to be accepted by turning machine If it halfs in final state
2. a String w of a language L is said to be rejected by turing machine, If either it halts in non final state or does not half

Difference B/w TM and PDA

Feature	Pushdown Automata	Turing Machine
Tape	Has only one tape	Has infinite tape
Memory	Has a stack	Has infinite memory
Computational Power	Less powerful	More powerful
Acceptance of languages	Accepts context-free languages	Accepts all languages
Type of Automata	Non-deterministic	Deterministic

Variants of Turing machine

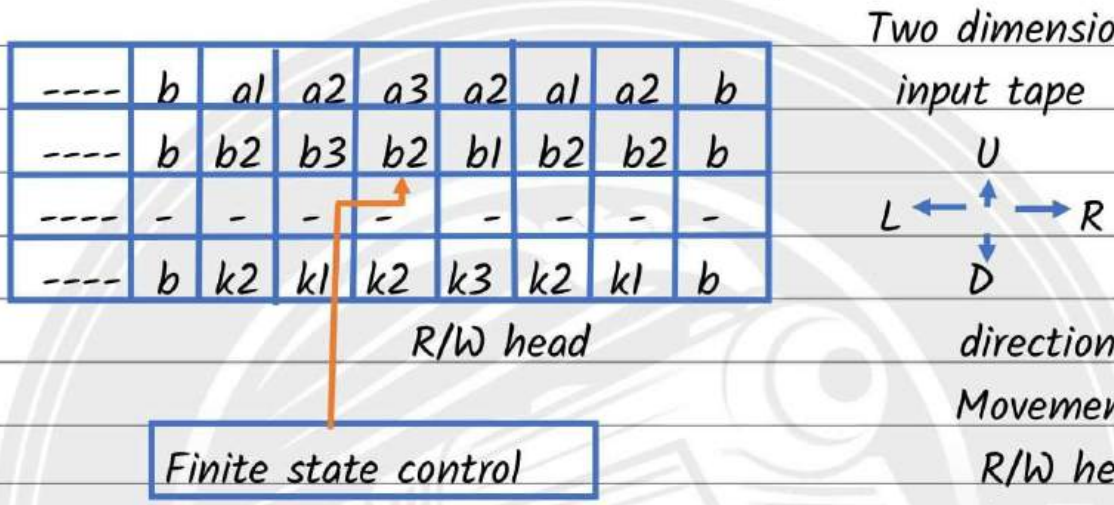
- Multitape Turning machine:- A turing machine with several input tapes is said to be a multiple turing machine. In a multiole TM each tape is completetd by its own independent R/W head .A multiple TM with k tapes is shown in figure below:-



Typically an n-type TM can be defined as M=(Q, Σ, δ, T, q o, b, F) where all the parameters have the same meaning as in the basic definition of TM but the transition function δ is defined as: -

$$Q \times T^k \rightarrow Q \times T^k \times [L, R, stop]^k$$

- **Multidimensions Turing machines:-** A TM is said to be a multidimensional TM its input tape can be viewed as ectending infinitely move than one dimension . The formal definition of two dimensional TM involves a transition function of the form:- $Q \times T \rightarrow Q \times T \times \{L, R, U, D, stop\}$



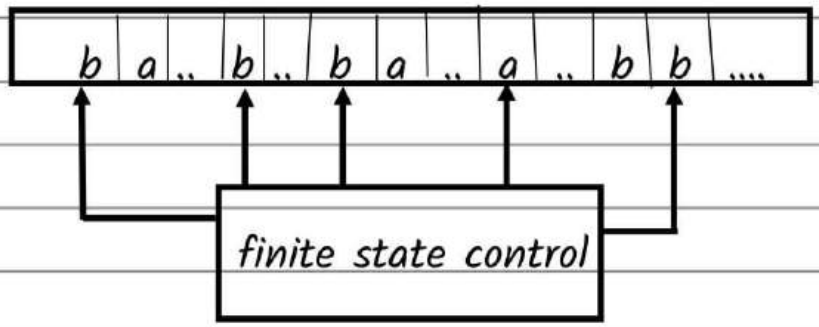
----	b	a1	a2	a3	a2	a1	a2	b
----	b	b2	b3	b2	b1	b2	b2	b
----	-	-	-	-	-	-	-	-
----	b	k2	k1	k2	k3	k2	k1	b

- **Multihead Turing machines:-** A multihead TM $M=(Q, \Sigma, \delta, q\ 0, b, F)$ can be viewed as TM eith a single tape and a single finite state control(also called unit)but the several independent R/Wheads. The transition function of a multihead TM can be defined as

$$Q \times T \rightarrow Q \times T \times \{L, R, stop\}^n$$

Where n= no of R/Wheads

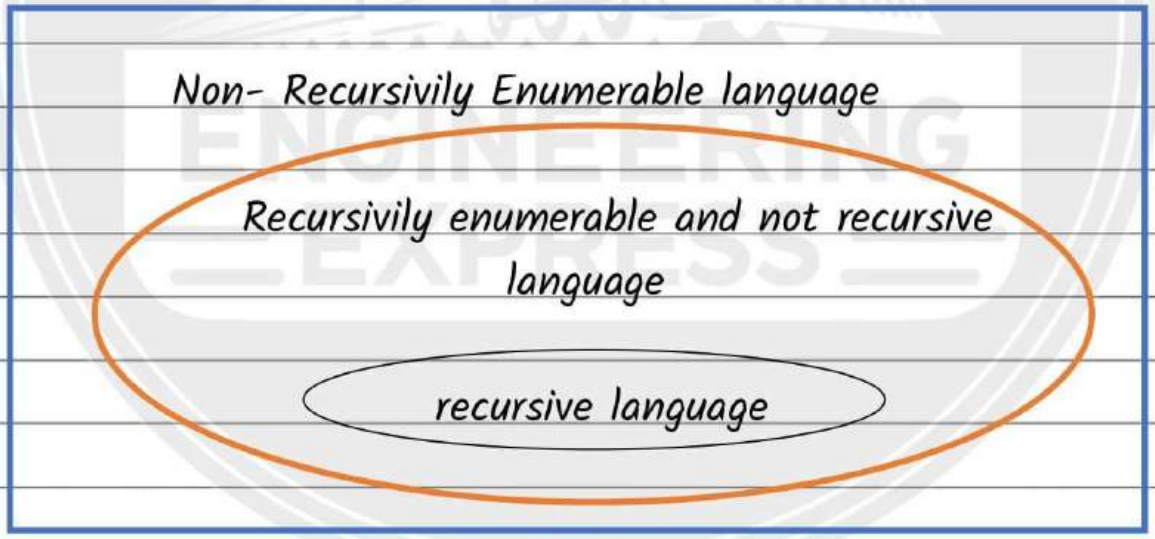
A move of the multihead TM depends upon the state and the symbols scanned each head . In one move , the R/W hed may take move independently L, R, or may remain stationary . A five head TM is shown in fig:-



Church's Thesis

According to Alanzo Church, "there is an effective procedure to solve a problem if and only if there is a turing machine, which stops that problem. This statement is called a thesis. Churche's thesis, rather than a theorem, because it is fundamentally unprovable Unprovable not, church's thesis is generally accepted to be true. In fact, many computers scientists define an effective procedure and something for which all. tuning machine exist, according to church's thesis, a turing machine can be treated as the most general computing system. Therefore, finding an algorithm for yes, no problem is equivalent to constructing a TM which can execute the algorithum

Recursive and Recursively Enumerable language



1. Recursively Enumerable language:- A language is Recursively Enumerable if some TM accept it . Let L be a recursively Enumerable language and M the TM that accept it.

For string W:-

a) If W belongs to L , then M halts in a final state

b) If W doesn't belongs to L, then M halts in a non final state loops forever

2. Recursive Language:- A language if some TM accept it and halts on any input string. A language Is recursive if there is a membership algorithm for it. Let L be recursive language and M turing machine that accepted for string W:-

a) If W belongs to L, then M halts in a final state.

b) If W does not belongs to L, then M halts in a non-final state.

From these two languages following observations are made:-

1. Every recursive language is recursively enumerable language.
2. Not all recursively language are recursive that means every TM which accepts the enumerable language must have some words for which it loops forever.

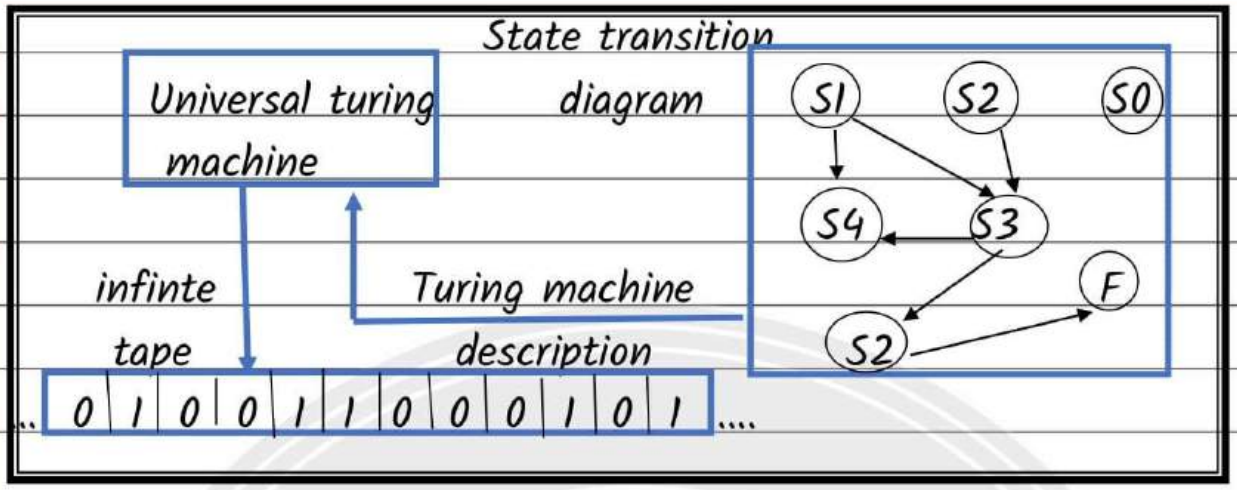
- Universal Turing machine(UTM):- As the name suggests, universal turing machine is a kind of turing machine which is capable of doing anything that any other turing machine can do. That means universal turing machine should have capability of iritating any turing machine T for.

A TM said to be universal TM if it is capable to accept all string.

When following information is given: -

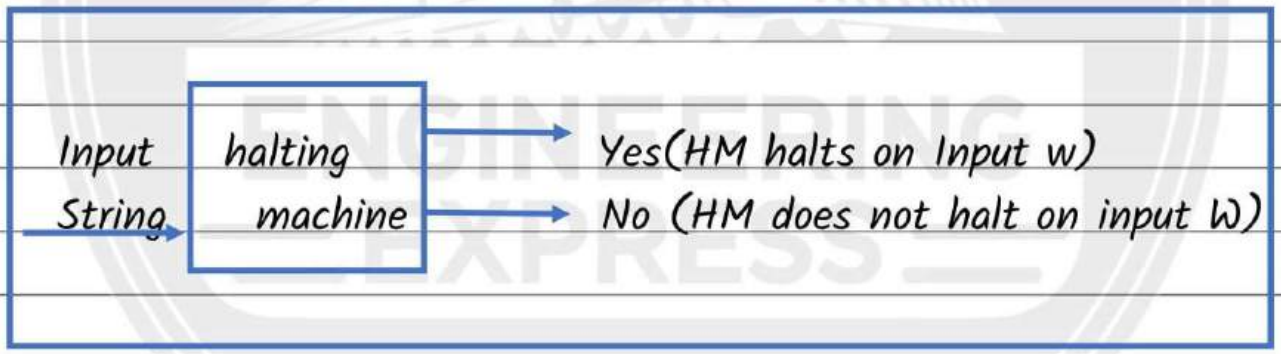
1. The description of T is term of its operation or program area of tape.
2. The initial configuration of T, which includes starting state and symbol. (State area of tape)
3. The input data to be given to TM, T (that is data area of the date)

We introduce the notation of a universal TM(UTM), which, along with the input on the tape, takes in the description of a machine M. The UTM can go on them to similar M on the rest of the content of the input table. A universal TM can thus simulate any other machine.



Halting Problem of a TM:-

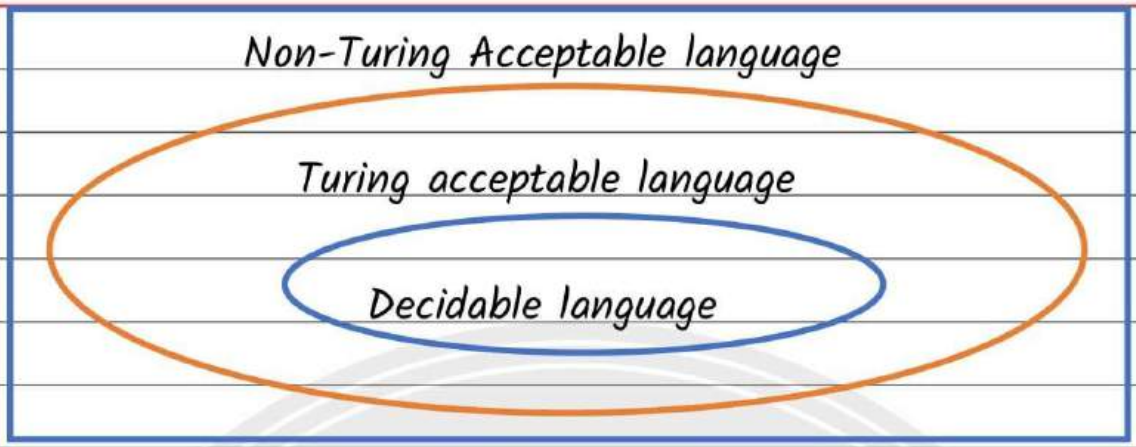
We know The input to a TM is a string. TM themselves can be written as a string. And these strings can be used as an input to other machine. While reading input a TM may halts or sometimes may enter in a infinite loop. In that case, machine is unable to decide what to do for a particular input.



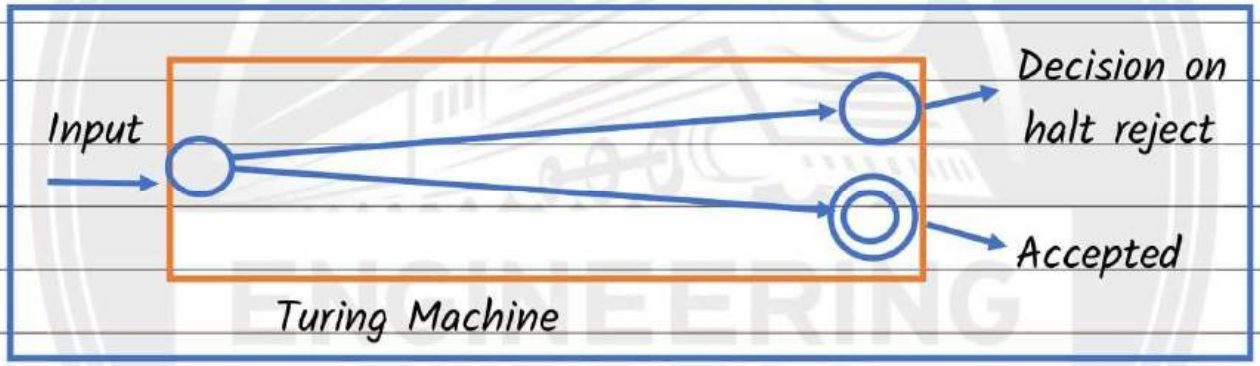
Hence, the halting problem is undecidable.

Decidable and undecidable problem

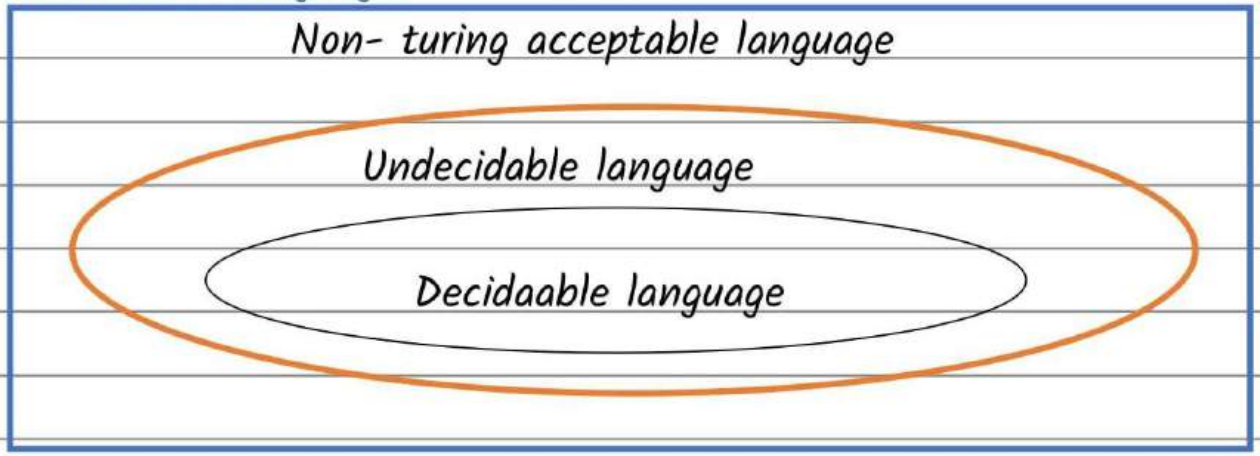
If an algorithm exists for a solving a problem, then this is called decidable problem. Whereas when there exists no algorithm to solve a problem, then the problem is called undecidable. A language is called decidable or recursive if there is a turing machine which accept and halts on every input string. Every decidable language is turing- acceptable.



A decision problem 'p' is decidable if the language 'L' of all yes instance to 'P' is decidable for a decidable language for each input string. The turing machine halts either at the accept or the reject state as. Depicted in the following diagram.



For an undecidable language, there is no turing machine which accepts the language and makes a decision forever. Input string w. A decision Problem 'P' is called undecidable if the language L of all yes instances to P is not decidable. Undecidable language are not recursive language, but sometimes they may be recursively ennumerable language.



PCP[Post correspondence problem]

The PCP was first introduced by Emil Post in 1946. The problem ever and alphabet set Sigma belongs to a class of Yes, no problems. The PCP consists of two strings that are of equivalent over the input sigma. The two list are A=w1, w2, w3.....wn and B= x1, x2, x3.....xn Then there exists a non empty set of integers i1, i2, i3, ...in Such that to solve the PCP, we try all the combinations of i1, i2, i3, ...in To find the Wi = Xi Then we say that PCP has a solution.

Modified PCP.

Is the first substream used in PCP is always x1 and y1 or(w1, x1) Then the PCP is known as Modified PCP. [MPCP]

Given a set of pairs of strings {(w1,x1), (w2, x2).....(wn, xn)} is there then the solution is an instance such that W1.

Wi1, wi2,.. win= x1, xi1, xi2 .. xin

That means this pair is forced. To be at the beginning of the string. For example, consider to list A and B as

	List A	List B
i	wi	xi
1	11	111
2	100	001
3	111	11

111	111
001	11
11	001

Then the solution is.

W1, w2, w3= x1,x2,x3

11100111=11100111

That means it is essential to have W1 and X1 at the beginning of the list.

Thank you

Join us and subscribe our Chenal