DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

COURSE MATERIALS

CS 301 THEORY OF COMPUTATION

VISION OF THE INSTITUTION

To mould true citizens who are millennium leaders and catalysts of change through excellence in education.

MISSION OF THE INSTITUTION

NCERC is committed to transform itself into a center of excellence in Learning and Research in Engineering and Frontier Technology and to impart quality education to mould technically competent citizens with moral integrity, social commitment and ethical values.

We intend to facilitate our students to assimilate the latest technological know-how and to imbibe discipline, culture and spiritually, and to mould them in to technological giants, dedicated research scientists and intellectual leaders of the country who can spread the beams of light and happiness among the poor and the underprivileged.

ABOUT DEPARTMENT

- Established in: 2002
- Course offered : B.Tech in Computer Science and Engineering

  M.Tech in Computer Science and Engineering

  M.Tech in Cyber Security
- Approved by AICTE New Delhi and Accredited by NAAC
- Affiliated to the University of A P J Abdul Kalam Technological University.

DEPARTMENT VISION

Producing Highly Competent, Innovative and Ethical Computer Science and Engineering Professionals to facilitate continuous technological advancement.

DEPARTMENT MISSION

1. To Impart Quality Education by creative Teaching Learning Process
2. To Promote cutting-edge Research and Development Process to solve real world problems with emerging technologies.
3. To Inculcate Entrepreneurship Skills among Students.
4. To cultivate Moral and Ethical Values in their Profession.
5.

PROGRAMME EDUCATIONAL OBJECTIVES

PEO1: Graduates will be able to Work and Contribute in the domains of Computer Science and Engineering through lifelong learning.

PEO2: Graduates will be able to Analyse, design and development of novel Software Packages, Web Services, System Tools and Components as per needs and specifications.

PEO3: Graduates will be able to demonstrate their ability to adapt to a rapidly changing environment by learning and applying new technologies.

PEO4: Graduates will be able to adopt ethical attitudes, exhibit effective communication skills, Teamworkand leadership qualities.

PROGRAM OUTCOMES (POS)

Engineering Graduates will be able to:

1. **Engineering knowledge**: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
2. **Problem analysis**: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3. **Design/development of solutions**: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
4. **Conduct investigations of complex problems**: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
5. **Modern tool usage**: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
6. **The engineer and society**: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
7. **Environment and sustainability**: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8. **Ethics**: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9. **Individual and team work**: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10. **Communication**: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
11. **Project management and finance**: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
12. **Life-long learning**: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

PROGRAM SPECIFIC OUTCOMES (PSO)

PSO1: Ability to Formulate and Simulate Innovative Ideas to provide software solutions for Real-time Problems and to investigate for its future scope.

PSO2: Ability to learn and apply various methodologies for facilitating development of high quality System Software Tools and Efficient Web Design Models with a focus on performance optimization.

PSO3: Ability to inculcate the Knowledge for developing Codes and integrating hardware/software products in the domains of Big Data Analytics, Web Applications and Mobile Apps to create innovative career path and for the socially relevant issues.

COURSE OUTCOMES

SUBJECT CODE: C301	
COURSE OUTCOMES	
C301.1	To identify various formal languages such as regular, context-free, context sensitive and unrestricted languages.
C301.2	To acquire the knowledge about designing finite state automata, regular grammar, regular expression and Myhill- Nerode relation representations for regular languages
C301.3	To implement the concept of pumping lemma for regular language and to acquire Knowledge on the concept of context free languages
C301.4	To design push-down automata and context-free grammar representations for context-free languages.
C301.5	To designing Turing Machines for accepting recursively enumerable languages.
C301.6	To Acquire knowledge about the **usage** of various types of Turing machines and their working and to identify the notions of decidability and undecidability of problems, Halting problem

MAPPING OF COURSE OUTCOMES WITH PROGRAM OUTCOMES

CO'S	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12
CO1	3											3
CO2		3	3	2								3
CO3		3	3	3								3
CO4		3	3	2	2							3
CO5		3	3	3								3
CO6		3	3	2								3
CO1	3	3	3	2.4	2							3

Note: H-Highly correlated=3, M-Medium correlated=2, L-Less correlated=1

PSO MAPPINGS

CO'S	PSO1	PSO2	PSO3
CO1	3		
CO2	3	2	
CO3	3	2	
CO4	3	3	
CO5	3	3	
CO6	3	3	

SYLLABUS

Course code	Course Name	L-T-P Credits	Year of Introduction
CS301	THEORY OF COMPUTATION	3-1-0-4	2016

Prerequisite: Nil

Course Objectives

- To introduce the concept of formal languages.
- To discuss the Chomsky classification of formal languages with discussion on grammar and automata for regular, context-free, context sensitive and unrestricted languages.
- To discuss the notions of decidability and halting problem.

Syllabus

Introduction to Automata Theory, Structure of an automaton, classification of automata, grammar and automata for generating each class of formal languages in the Chomsky Hierarchy, decidability and Halting problem.

Expected Outcome

The Students will be able to

i. Classify formal languages into regular, context-free, context sensitive and unrestricted languages.
ii. Design finite state automata, regular grammar, regular expression and Myhill- Nerode relation representations for regular languages.
iii. Design push-down automata and context-free grammar representations for context-free languages.
iv. Design Turing Machines for accepting recursively enumerable languages.
v. Understand the notions of decidability and undecidability of problems, Halting problem.

Text Books

1. John E Hopcroft, Rajeev Motwani and Jeffrey D Ullman, Introduction to Automata Theory, Languages, and Computation, 3/e, Pearson Education, 2007
2. John C Martin, Introduction to Languages and the Theory of Computation, TMH, 2007
3. Michael Sipser, Introduction To Theory of Computation, Cengage Publishers, 2013

References

1. Dexter C. Kozen, Automata and Computability, Springer1999.

Course Plan

Module	Contents	Hours	End Sem. Exam Marks
I	Introduction to Automata Theory and its significance. **Type 3 Formalism:** Finite state automata – Properties of transition functions, Designing finite automata, NFA, Finite Automata with Epsilon Transitions, Equivalence of NFA and DFA, Conversion of NFA to DFA, Equivalence and Conversion of NFA with and without Epsilon Transitions.	10	15 %
II	Myhill-Nerode Theorem, Minimal State FA Computation. Finite State Machines with Output- Mealy and Moore machine (Design Only), Two- Way Finite Automata. Regular Grammar, Regular Expressions, Equivalence of regular expressions and NFA with epsilon transitions. Converting Regular Expressions to NFA with epsilon transitions Equivalence of DFA and regular expressions, converting DFA to Regular Expressions.	10	15 %

FIRST INTERNAL EXAM			
III	Pumping Lemma for Regular Languages, Applications of Pumping Lemma. Closure Properties of Regular sets (Proofs not required), Decision Problems related with Type 3 Formalism **Type 2 Formalism:-** Context-Free Languages (CFL), Context-Free Grammar (CFG), Derivation trees, Ambiguity, Simplification of CFG, Chomsky Normal Form, Greibach normal forms	**09**	**15 %**
IV	Non-Deterministic Pushdown Automata (NPDA), design. Equivalence of acceptance by final state and empty stack in PDA. Equivalence between NPDA and CFG, Deterministic Push Down Automata, Closure properties of CFLs (Proof not required), Decision Problems related with Type 3 Formalism.	**08**	**15 %**
SECOND INTERNAL EXAM			
V	Pumping Lemma for CFLs, Applications of Pumping Lemma. **Type 1 Formalism:** Context-sensitive Grammar. Linear Bounded Automata (Design not required) **Type 0 Formalism:** Turing Machine (TM) – Basics and formal definition, TMs as language acceptors, TMs as Transducers, Designing Turing Machines.	**09**	**20 %**
VI	Variants of TMs -Universal Turing Machine, Multi- tape TMs, Non Deterministic TMs, Enumeration Machine (Equivalence not required), Recursively Enumerable Languages, Recursive languages, Properties of Recursively Enumerable Languages and Recursive Languages, Decidability and Halting Problem. Chomsky Hierarchy	**08**	**20 %**
END SEMESTER EXAM			

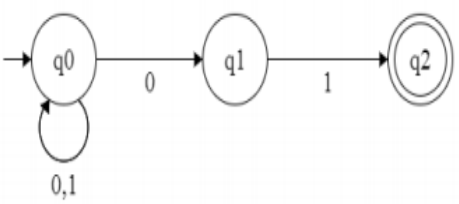
Question Paper Pattern

1. There will be *five* parts in the question paper – A, B, C, D, E
2. Part A
   a. Total marks : 12 b. *Four* questions each having *3* marks, uniformly covering modules I and II; All *four* questions have to be answered.
3. Part B
   a. Total marks : 18 b. *Three* questions each having *9* marks, uniformly covering modules I and II; T*wo* questions have to be answered. Each question can have a maximum of three subparts.
4. Part C
   a. Total marks : 12 b. *Four* questions each having *3* marks, uniformly covering modules III and IV; All *four* questions have to be answered.
5. Part D
   a. Total marks : 18 b. *Three* questions each having *9* marks, uniformly covering modules III and IV; *Two* questions have to be answered. Each question can have a maximum of three subparts
6. Part E
   a. Total Marks: 40 b. *Six* questions each carrying 10 marks, uniformly covering modules V and VI; *four* questions have to be answered. A question can have a maximum of three sub-parts.

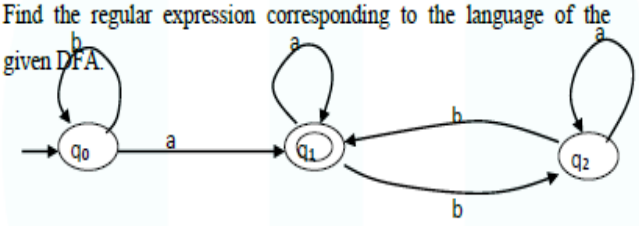
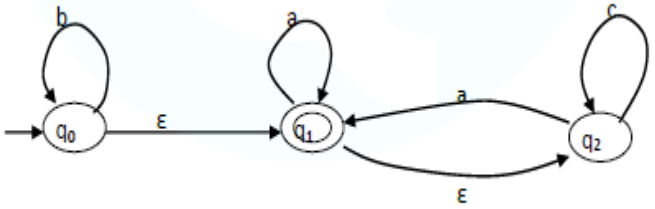
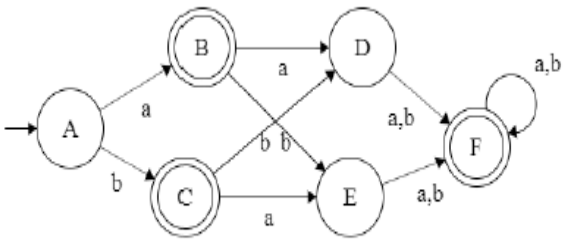
There should be at least 60% analytical/numerical questions.

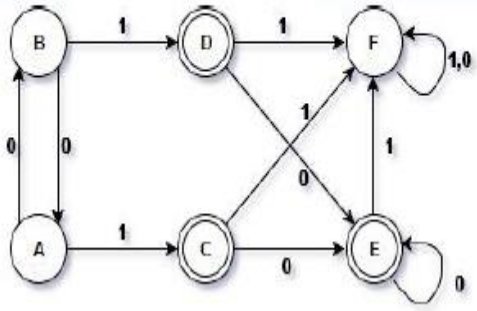
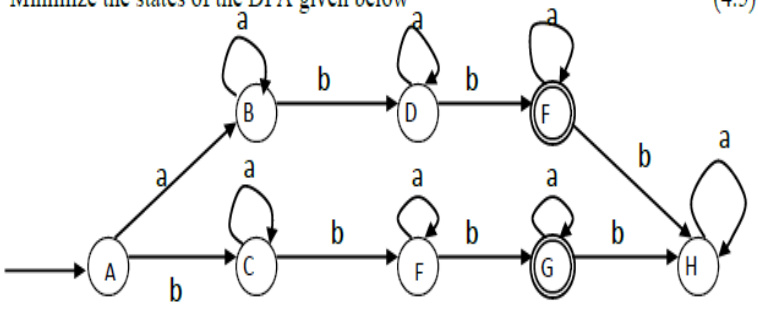
QUESTION BANK

MODULE I

Q:NO:	QUESTIONS	CO	KL	PAGE NO:
1	Formally define extended delta for an NFA. Show the processing of input $w = 0101$ for the following NFA. (3) 	CO1	K6	15
2	Differentiate between the transition function in DFA, NFA and ϵ-NFA	CO1	K3	19
3	a) Convert the following NFA to DFA and describe the language it accepts. (5) $M = (\{P, Q, R, S, T\}, \{0,1\}, \delta, P, \{S, T\})$ and δ is given as: b) Prove that " A language L is accepted by some ϵ-NFA if and only if L is accepted by some NFA" (4)	CO1	K6	23
4	What is a Finite state automata?	CO1	K1	05
5	Construct DFA for the language 101^*	CO1	K6	09

	0	1
P	{P,Q}	{P}
Q	{R,S}	{T}
R	{P,R}	{T}
S	-	-
T	-	-

6	What is a two-way finite automata?	CO1	K1	86
7	a) Find the regular expression corresponding to the language of the given DFA. (4.5)  b) Prove the equivalence of NFA and ε-NFA. (4.5)	CO1 CO1	K6 K1	59 34
8	a) Convert the ε-NFA to NFA. (4.5)  b) Prove the equivalence of regular expression and finite state automata (4.5)	CO1 CO1	K6 K1	27 34
	MODULE II			
1	Design a Moore machine to determine the residue of mod 2 of the input treated as a binary string. (3)	CO2	K6	40
2	a) State Myhill-Nerode theorem, Minimize the following DFA. (5)  b) Find an equivalent ε-NFA for the following regular expression (4) (0 + 1)*011	CO2 CO2	K6 K6	71 50
3	What is a two-way finite automata?	CO2	K1	86

4	Give a regular expression for the set of all strings not containing 101 as a substring (3	CO2	K6	45
5	a) Construct Regular grammar for the regular expression : (5) L = (a + b)*(aa + bb)(a + b)* b) List the closure properties of Regular sets. (4)	CO2	K6	90
6	State Myhill-Nerode theorem. Minimize the following DFA by table filling method using Myhill-Nerode theorem describing the steps in detail. (9) 	CO2	K6	71
7	Minimize the following DFA. 6 δ | a | b →P0 | P0 | P1 P1 | P2 | P1 P2 | P3 | P1 *P3 | P3 | P4 *P4 | P5 | P4	CO2	K6	67
8	b) Minimize the states of the DFA given below (4.5) 	CO2	K6	71
9	Can we use finite state automata to evaluate 1's complement of a binary number? Design a machine to perform the same.(3)	CO2	K6	39
10	State and prove equivalence of Regular expression and Finite automata	CO2	K1	54

11	Give the regular expression for the language: strings of 'a' and 'b' containing at least two 'b'. (3)	CO2	K6	45
	MODULE III			
1	Define context free grammar. Consider the following CFG (3) $S \rightarrow aS \mid Sb \mid a \mid b$ Prove by induction on the string length that no string in L(G) has ba as substring.	CO3	K6	111
2	a) State pumping lemma for regular languages. Prove that the language $L = \{a^{n^2} \mid n > 0\}$ is not regular. (5) b) Convert the following grammar into Chomsky normal form (4) $S \rightarrow ASB \mid \in,\quad A \rightarrow aAS \mid a,\quad B \rightarrow SbS \mid A \mid bb$	CO3 CO3	K4 K6	90 107
3	Construct CFG for $a^n b^m c^{m+n}$ where m,n>=1	CO3	K4	106
4	What do you mean by useless symbol in a grammar? Show the elimination of useless symbols with an example..	CO3	K1	107
5	Define CFG for the following languages over the alphabets {a,b} (9) i. $L = \{ a^{m+n}b^m c^n, m>0 \}$ ii. L contains all odd length strings only iii. $L = \{ 0^n 1^n 2^n \mid n>0 \}$	CO3	K6	111
6	Prove that the following languages are not regular (9) i. $L = \{0^{i^2} such\ that\ i \geq 1\}$ is not regular ii. $L = \{a^p\ such\ that\ p\ is\ a\ prime\ n\ umber$	CO3	K4	90
7	Convert the following grammar to Chomsky Normal Form. 3 S-> 0S0|1S1| ϵ	CO3	K6	107
8	Whether the following grammar is ambiguous? 3 E-> E+E| E*E|I I-> 0|1|a|b	CO3	K4	105
9	a) Verify that the following languages is not regular: 4.5 $\{a^n b^{2n} \mid n>0\}$ b) Which of the following operations are closed under regular sets. Justify your answer. 4.5 i) Complementation ii) Set difference iii) string reversal iv) Intersection	CO3 CO3	K4 K1	90 98

10	a) Give a CFG for the language N(M) where M = ({p,q,r}, {0, 1}, {Z, X_0}, δ, q_0, Z, r) and δ is given by δ(p , ϵ , X_0) = {(q, ZX_0)}, δ(q , ϵ , X_0) = {(r, ϵ)}, δ(q , 1, Z) = {(q, ZZ)}, δ(q , 0, Z) = {(q, ϵ)}. 4.5 b) Find the Greibach normal form grammar equivalent to the following CFG: 4.5 S → AB A → BS| 1 B → SA|0	CO3 CO3	K6 K6	107 116
11	State closure properties of regular sets	CO3	K1	98
12	a) Let G be the grammar (4) $S \rightarrow aB \mid bA, \quad A \rightarrow a \mid aS \mid bAA, \quad B \rightarrow b \mid bS \mid aBB$ For the string *aabbaabbba* find i) leftmost derivation, ii) parse tree, and iii) Is the grammar ambiguous?	CO3	K4	105
13	a) Convert to Greibach Normal form. {S→AB, A→SA|AA|a, B→SB|b} (4.5) b) Prove the equivalence of CFG and PDA. (4.5)	CO3	K6	116
	MODULE IV			
1	Design a PDA to accept the set of strings with twice as many 0's as 1's. (3)	CO4	K6	133
2	a) Prove the equivalence of acceptance of a PDA by final state and empty stack. (6)	CO4	K1	123
3	b) Construct PDA for the language **wcw**^{R} where w is string of zeroes and ones. (4.5)	CO4 CO4	K3 K6	162 133
4	Explain any two closure properties of CFLs	CO4	K2	135
5	Compare DPDA and NPDA	CO4	K2	132
6	Explain the different methods by which a PDA accepts a language	CO4	K2	122
7	Design a Push Down Automata for the language L= $\{a^n b^{2n} \mid n>0\}$ (9) Trace your PDA with n=3.	CO4	K6	133
8	Explain equivalance of acceptance of NPDA and CFG	CO4	K2	128
9	Explain decision properties related to CFL	CO4	K2	136
10	Design a PDA to accept the language $a^n b^n c^m$ where m,n>=1	CO4	K6	134
	MODULE V			

1	a) Show that the language $L = \{ww \mid w \in \{a,b\}^*\}$ is not a CFL. (5) b) Design a TM to compute the 2's complement of a binary string. (5)	CO5 CO5	K6 K6	144 161
2	a) State and prove pumping lemma for context free languages. Mention the application of pumping lemma. (6) b) Design a Turing machine to accept , (4) $L = \{ w \in \{0,1\}^* \mid w$ *has equal number of* $0's$ *and* $1's\}$.	CO5 CO5	K1 K6	144 161
3	Design a TM to find the sum of two numbers m and n. Assume that initially the tape contains m number of 0s followed by # followed by n number of 0s (5)	CO5	K6	161
4	a) Give the context sensitive grammar for the language $a^n b^n c^n$ where n>0. (5)	CO5	K6	150
6	a) What is Pumping lemma for CFL?	CO5	K1	144
8	a) What is a Context sensitive grammar(CSG). Design a CSG to accept the language $L = \{ 0^n 1^n 2^n \mid n>0 \}$ (6) b) Define Linear Bound Automata (4)	CO5 CO5	K3 K1	150 151
9	a) State Pumping lemma for CFLs. Write the applications of pumping lemma for CFL s. 4 b) Check whether $L=\{a^i b^i c^i \mid i > 0\}$ belong to CFL or not. 6	CO5 CO5	K1 K6	144 147
10	a) Design Turing machine to compute addition of two numbers. Assume unary notation for number representation. (6) b) Describe all instantaneous descriptions (ID) from initial ID: q_0010 to Final ID: 00 with respect to constructed Turing Machine.(assume q_0 as initial state.) (4)	CO5 CO5	K6 K6	160 159
	MODULE VI			
1	b) Define formally Type 0, Type 1, Type 2 and Type 3 grammar. Show the corresponding automata for each class (5)	CO6	K2	176
2	a) List the closure properties of Recursive Languages (4)	CO6	K1	169
3	b) Define a Universal Turing Machine (UTM). With the help of suitable arguments show the simulation of other Turing machines by a UTM. (6)	CO6	K1	163
4	b) What is a non-deterministic Turing Machine? Give an example.	CO6	K1	167
5	b) What is recursive and recursively enumerable languages (5)	CO6	K2	169

6	Discuss the Undecidable Problems About Turing Machines (10)	CO6	K4	176
7	Explain multitape turing machine in detail	CO6	K2	165
8	Prove that intersection of two recursive languages are recursive	Co6	K4	173

APPENDIX 1

CONTENT BEYOND THE SYLLABUS

S:NO;	TOPIC	PAGE NO:
1	Classes of automata	180
2	Applications of automata	281

MODULE NOTES

MOD - I

Introduction to Automata Theory & Significance

Automata theory is the study of machines or computing devices.

The main goal of this abstract machines was to describe the boundary between what a computing machine could do and what it could not do.

Theory of computation (TOC) is related to something about computation. Computation means some task that can be performed by computer or any calculating device.

Need for the Automata Theory :-

* Automata Theory is essential for the study of the limits of the computation
* Designing and checking the behaviour of digital circuits
* Pattern searching in websites
* Verifying systems of all types that have a finite number of distinct states, such as communication protocol or protocols for secure exchange of information.

Basic Concepts of Automata Theory

i) Symbol

Symbol is the basic building block of TOC. It can be of the form a, b, 0, 1, ...  ...

ii) Alphabets

An alphabet is a finite, nonempty set of symbols. Alphabet is Represented by "Σ". The common alphabets are

*) $\Sigma = \{0, 1\}$, the binary alphabet

*) $\Sigma = \{a, b \ldots z\}$, the set of all lower-case letters

*) $\Sigma = \{A, B \ldots Z\}$, the set of all upper-case letters

*) The set of all Ascii characters.

iii) Strings

A String or word is a finite sequence of symbols chosen from some alphabet.

Eg:- 01101 is a string from the binary alphabet $\Sigma = \{0, 1\}$

aabb is a string from the alphabet $\Sigma = \{a, b\}$

Empty String is a string with zero occurences of the symbol. An empty string is denoted by 'ϵ'. Empty string can be chosen from any alphabet.

Length of the string is the number of positions for symbols in strings eg: The string 01101 has length 5.

The standard notation for the length of a string w is $|w|$

eg: $|011| = 3$ and $|\epsilon| = 0$.

Powers of an alphabet :-

If Σ is an alphabet, then set of all strings of certain length from that alphabet can be expressed by an exponential notation

Σ^k be the set of all strings of length k over symbols in Σ

Note:-

$\Sigma^0 = \{\epsilon\}$ Regardless of what the alphabet Σ is. i.e ϵ is the only string whose length is zero.

Eg:.

Let $\Sigma = \{a, b\}$ contain 2 symbols

$\Sigma^0 = \{\epsilon\}$ - set of all strings of length zero. i.e 2^0

$\Sigma^1 = \{a, b\}$ - set of all strings of length 1 i.e 2^1

$\Sigma^2 = \Sigma\Sigma = \{a, b\}\{a, b\}$

$= \{aa, ab, ba, bb\}$ = set of all strings of length 2

$= 2^2 = 4$

$\Sigma^3 = \Sigma\Sigma\Sigma = \{aa, ab, ba, bb\}\{a, b\}$

$= \{aaa, aba, baa, bba, aab, abb, bab, bbb\}$

: set of all strings of length 3

i.e equal to $2^3 = 8$

In general we can represent Σ^n where 'n' is the length strings over Σ

Kleen closure

The set of all strings over alphabet Σ is denoted by Σ^*. We can substitute any number in place of * starting from zero.

In general Σ^* : set of all strings possible over $\Sigma = \{a, b\}$

i.e, $\boxed{\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \ldots \cup \Sigma^n}$

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

So Σ^* is infinite

We can say Σ^* is the mother set or universal set of all languages

The set of non empty strings from alphabet Σ is denoted by Σ^+.

ie $\boxed{\Sigma^+ = \Sigma^1 \cup \Sigma^2 \cup \cdots \cup \Sigma^n}$

NB:-
Alphabet is always finite

iv) Languages
A language is a set of strings, all of which are chosen from some Σ^*

If Σ is an alphabet, and $L \subseteq \Sigma^*$, then L is a language over Σ.

A language can be a subset of the universal set Σ^*

Number of languages possible over Σ^* is infinite

eg:- Let $\Sigma = \{a, b\}$

→ Define language L_1, which is the set of all strings of length 2.

So $L_1 = \{aa, ab, ba, bb\}$.

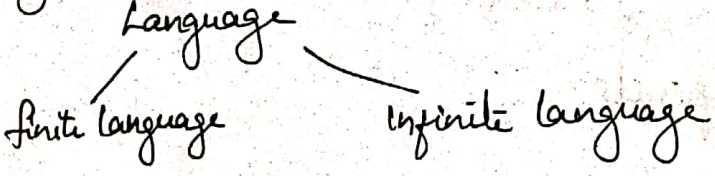
→ Define language L_2, which is the set of all strings of length 3

$L_2 = \{aaa, aab, aba, abb, baa, bab, bba, bbb\}$

→ Define language L_3, which is the set of all strings starts with a

$L_3 = \{a, aa, ab, aaa, aba, aab, abb, aaaa \cdots\}$

A language can be of two types -



In the above given examples language L_1 and L_2 are finite but the language L_3 is infinite.

Finite state Automata

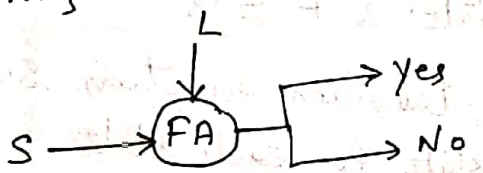
A Finite Automaton or Finite state automaton consists of a finite set of states and the set of transitions from state to state that occur on input symbol which is chosen from an alphabet Σ.

For the same input symbol there may be 0, 1 or more transition from each state.

One state usually denoted by q_0 which is the initial state in which the automaton starts. some states are designated as final or accepting states.

A Finite automaton provides a function from Σ^* into {yes, No}

Eg:



The above finite automata will tells whether the given string is present in the language or not.

Formal definition of a finite Automaton:-

An finite automaton can be Represented by a 5-tuple as $(Q, \Sigma, \delta, q_0, F)$ where

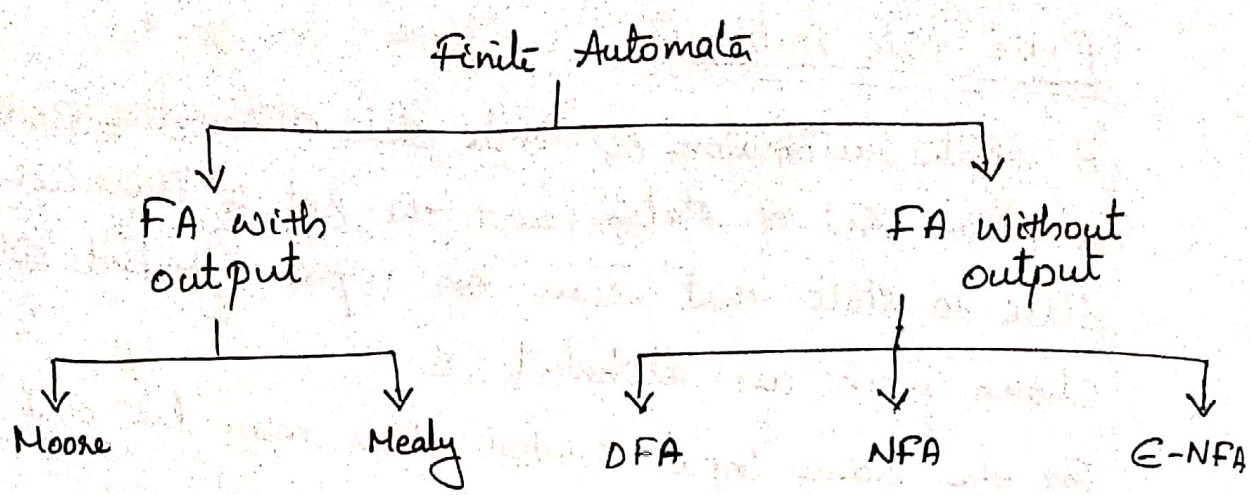
Q - Finite set of states

Σ - is a finite set of symbols called alphabet of the automaton

δ - transition function

q_0 - initial state from where any input is processed

F - set of final states



Deterministic Finite state Automaton (DFA)

A deterministic finite state automaton is specified by 5-tuple $(Q, \Sigma, \delta, q_0, F)$ where

Q - finite set of states

Σ - a finite set of input alphabet

q_0 - initial state

F - set of final state & $F \subseteq Q$

$\delta = Q \times \Sigma \rightarrow Q$, the transition function such that $\delta(q, a)$ is a state for each state q and for each input symbol a

DFA can be represented by two ways

i) Transition Diagram

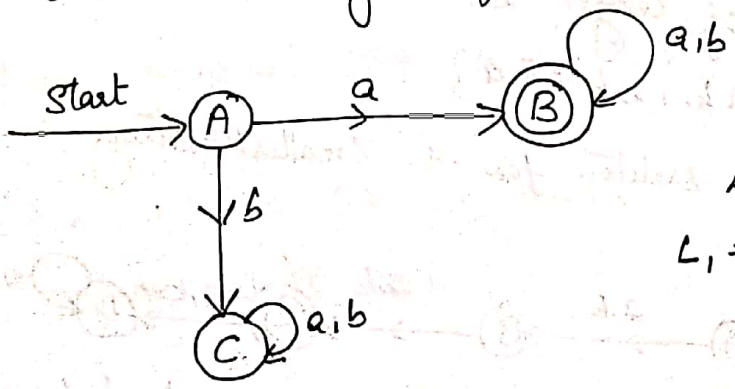
ii) Transition Table

Transition Diagram is the common way of representing finite automata. It consists of circles and arcs. Each circle is called the state of the finite automata and arc represent the transition from one state to

another state. The input symbols are represent on top of the arc.
One of the state is called initial state which is indicated by the arrow labeled 'start'.
One is called final state which is indicated by double circle. A FA can have more than one final state.

Transition Table represent the list of transition Rules (functions) of a finite automaton, with the help of transition table one can easily observe the different transitions of finite state automaton The rows of the table corresponds to the states and columns to the input.

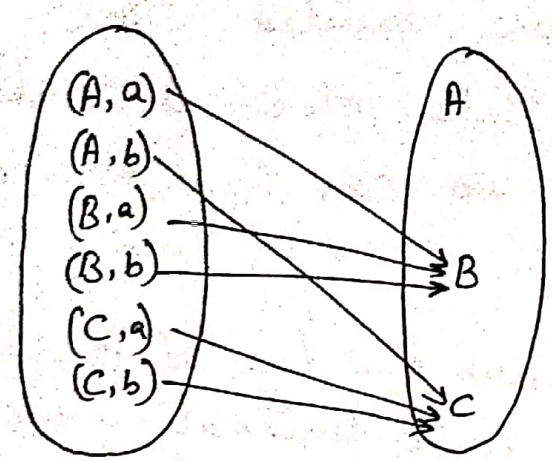
Q. Construct a FA for the language which is the set of all strings which starts with a over $\Sigma=\{a,b\}$
The transition diagram for the above FA is as follows



Let the language L_1 be
$L_1 = \{a, aa, ab, aaa, aab \ldots\}$

The state A is the start state or initial state
State B is called final state and is represented in double circle
State C is called dead state or error state.

For the above problem the transition function $\delta: Q \times \Sigma \rightarrow Q$ is as follows where $Q = \{A, B, C\}$ and $\Sigma = \{a, b\}$



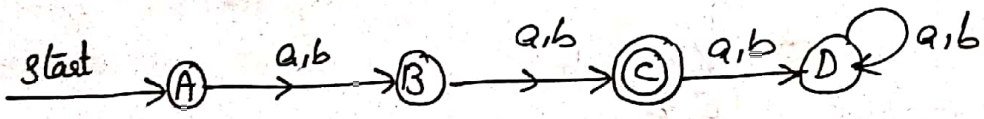
The transition table is as follows.

	a	b
→ A	B	C
B*	B	B
C	C	C

Q Construct DFA that accepts set of all strings over $\Sigma = \{a, b\}$ of length 2.

$L_1 = \{aa, ab, bb, ba\}$

Step 1: Draw skeleton for the smallest string in the language

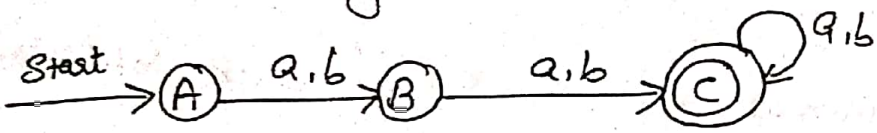


On state A & B we have two inputs 'a' and 'b' But on 'c' there is no transition. It moves to D state called dead state. On reaching dead state we cannot go back.

Q Construct a DFA , $w \in \{a,b\}$ and $|w| \geq 2$.

$L = \{aa, ab, ba, bb, aaa \ldots bbb \ldots\}$

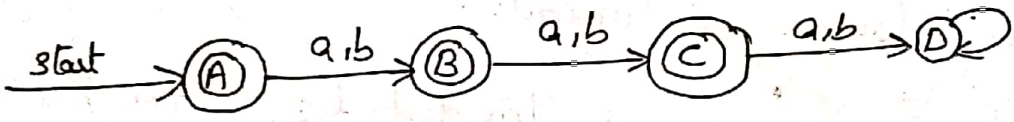
The language is infinite so ~~we cannot give finite automata~~ draw the skeleton of smaller string



Q Construct a DFA , $w \in \{a,b\}$, $|w| \leq 2$.

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

The language is finite. So draw the skeleton of the biggest string in language.



When we want to accept $\in$, then make the initial state equal to accept state.

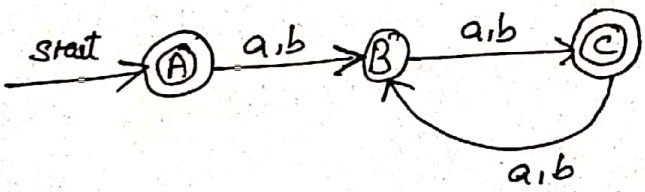
Q Construct minimal DFA , $w \in \{a,b\}$ such that

$|w| \bmod 2 = 0$

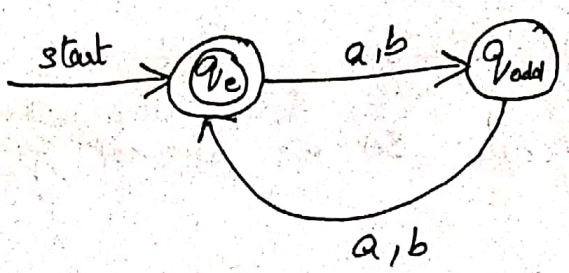
$L = \{\in, aa, ab, ba, bb, aaaa \ldots bbbb \ldots\}$

The language is infinite

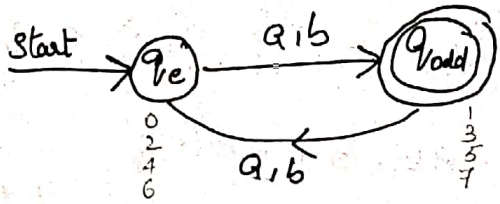
⇓ Minimal DFA



⇓ minimal DFA



Q Construct minimal DFA, $w = \{a, b\}$ such that $|w| \bmod 2 = 1$ or $|w| \cong 1 \bmod 2$

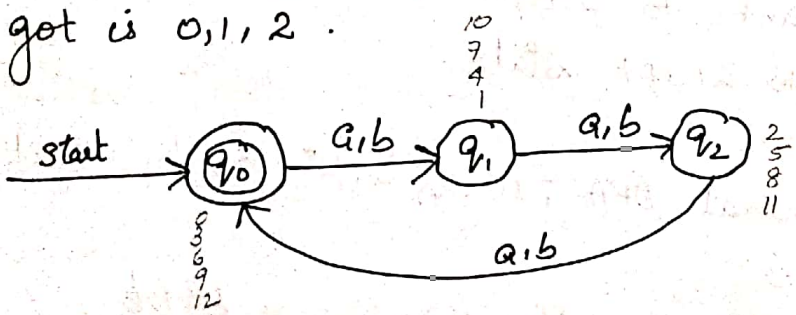


Q Construct minimal DFA $w \in \{a,b\}^*$ $L = |w| \bmod 3 = 0$

Let the language $L = \{\epsilon, aaa, aab, aba, bb \ldots aaaaaa \ldots\}$

The language is infinite.

When a number is divided by 3 the Remainder we get is 0, 1, 2.



In state q_0 we get Remainder zero
In state q_1 we get Remainder one
In state q_2 we get Remainder two

NB:-

In general $|w| \bmod n = 0$ then minimal DFA Contain 'n' no: of states.

Extending Transition functions in DFA

Extended transition function describes what happens when we start in any state and follow any sequence of input. If δ is our transition function, then the extended transition function constructed from δ will be called $\hat{\delta}$.

The extended transition function is a function that takes a state 'q' and a string 'w' and returns a state p. the where the automaton reaches when starting in q and processing the sequence of inputs w. $\hat{\delta}$ can be defined by induction on the length of input string as follows.

BASIS: $\hat{\delta}(q, \epsilon) = q$. i.e if we are in state q and read no input, then we are still in state q.

INDUCTION: Suppose w is a string of the form xa, that is 'a' is the last symbol of input 'w' and x is the string consists of rest of the symbols in input. for eg: $w = 1101$ where $x = 110$ and $a = 1$. Then

$$\hat{\delta}(q, w) = \delta(\hat{\delta}(q, x), a)$$

To compute $\hat{\delta}(q, w)$, first compute $\hat{\delta}(q, x)$, the state that the automaton is in after processing all but the last symbol of w. Suppose this state is p, that is $\hat{\delta}(q, x) = p$. Then $\hat{\delta}(q, w) = \delta(p, a)$

Nondeterministic Finite Automata

A nondeterministic finite Automata (NFA) has the power to be in several states at once.

In NFA,

i) one input symbol causes to move more than one state or a state may contain more than one transition from same input symbol.

ii) It is not compulsory that all the states have to consume all input symbols in Σ.

Definition of Nondeterministic Finite Automata

An NFA can be Represented by 5 tuple notation

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

where

Q is the finite set of states.

Σ is the finite set of input symbols.

q_0 is the member of Q, the start state

F is the final state $F \subseteq Q$

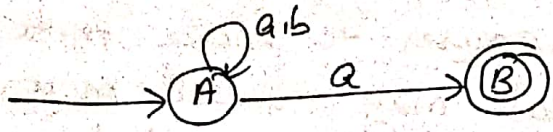
δ is the transition function that takes a state in Q and an input symbol in Σ as arguments and returns a subset of Q.

The only difference between an NFA and a DFA is in the type of values that δ returns, a set of state in case of NFA and a single state in case of DFA.

$$\delta : Q \times \Sigma = 2^Q$$

Q Construct NFA which accepts language L where every string ends with 'a' over $\Sigma = \{a, b\}$

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



Consider the input 'a'

$A \xrightarrow{a} A$

$A \xrightarrow{a} B$

The string is accepted since B is the final state

Consider another input ab

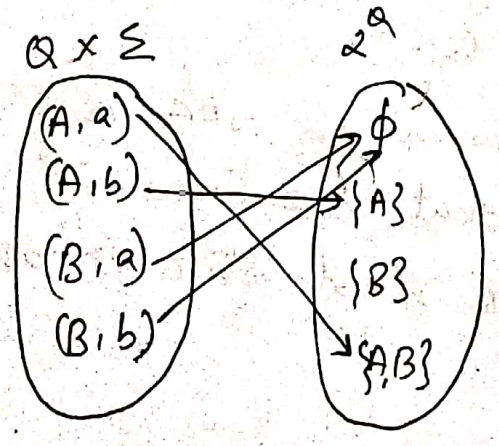
$A \xrightarrow{a} A \xrightarrow{b} A$

$A \xrightarrow{a} B \xrightarrow{b} \phi$

The end state do not contain final or accepting state so the input is rejected.

The ~~transf~~ transition function of the NFA is as given below

$Q = \{A, B\}$ $\Sigma = \{a, b\}$ $2^Q = \{\phi, \{A\}, \{B\}, \{A, B\}\}$



The transition table can be represented by

	a	b
→ A	{A,B}	{A}
*B	{ϕ}	{ϕ}

The transition tables can be used to specify the transition function for an NFA as well as for a DFA. The only difference is the entry of NFA's and DFA's transition table is that, each entry in the table for the NFA is a set, even if the set is a singleton (has one member). Also when there is no transition at all from a given state on a given input symbol, the proper entry is ϕ, the empty set.

Extended Transition Function

As for DFA, we need to extend transition function δ of an NFA to a function $\hat{\delta}$ that takes a state q and a string of input symbol w, and returns the set of states that the NFA is in if it starts with the state q and processes the string w.

BASIS: $\hat{\delta}(q, \epsilon) = \{q\}$. That is without reading any input symbols, we are only in the state we began in.

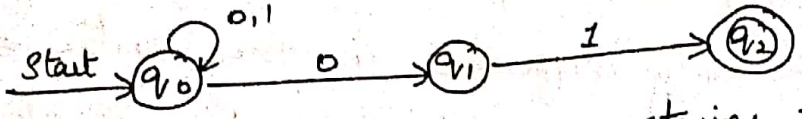
INDUCTION: Suppose w is of the form $w = xa$, where a is the final symbol of w and x is the rest symbol of w

Suppose that $\hat{\delta}(q, x) = \{p_1, p_2, \ldots p_k\}$. Let

$$\bigcup_{i=1}^{k} \delta(p_i, a) = \{r_1, r_2 \ldots r_m\}$$

Then $\hat{\delta}(q, w) = \{r_1, r_2 \ldots r_m\}$. We compute $\hat{\delta}(q, w)$ by first computing $\hat{\delta}(q, x)$ and by then following any transitions from any of these states that is labeled 'a'. we can also have $\hat{\delta}(q, xa) = \hat{\delta}(\delta(q, x), a)$

Consider the example:



The above NFA accepts all strings that end in 01

Let us use $\hat{\delta}$ to describe the processing of input 00101 for the above NFA.

Summary of the steps are as given below.

1. $\hat{\delta}(q_0, \epsilon) = \{q_0\}$
2. $\hat{\delta}(q_0, 0) = \delta(q_0, 0) = \{q_0, q_1\}$
3. $\hat{\delta}(q_0, 00) = \delta(\hat{\delta}(q_0, 0), 0)$
   $= \delta(q_0, q_1), 0)$
   $= \delta(q_0, 0) \cup \delta(q_1, 0)$
   $= \{q_0, q_1\} \cup \phi = \{q_0, q_1\}$
4. $\hat{\delta}(q_0, 001) = \delta(\hat{\delta}(q_0, 00), 1)$
   $= \delta(q_0, q_1), 1)$
   $= \delta\{q_0, 1) \cup \delta(q_1, 1)$
   $= \{q_0, q_2\}$

5. $\hat{\delta}(q_0, 0010) = \delta(\hat{\delta}(q_0, 001), 0)$

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

$= \delta(q_0, 0) \cup \delta(q_2, 0)$

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

6. $\hat{\delta}(q_0, 00101) = \delta(\hat{\delta}(q_0, 0010, 1)$

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

$= \delta(q_0, 1) \cup \delta(q_1, 1)$

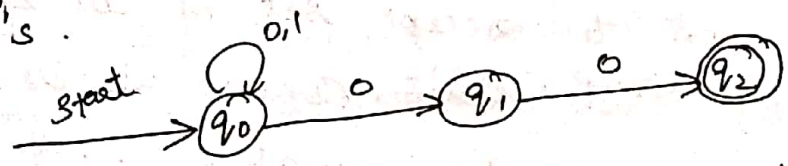
$= \{q_0, q_2\} = \{q_0, (q_2)\}$

Acceptance of string by NFA:-

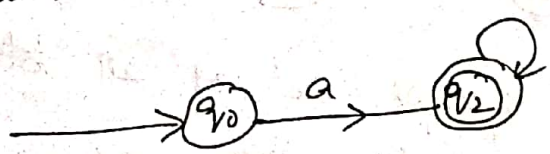
Acceptance of a string in NFA is defined as if any completed path ends with final state, then we can say it is accepted otherwise it is not accepted.

Designing of NFA's :-

Q Design an NFA to accept set of strings over alphabet set $\{0, 1\}$ and ending with two consecutive 0's.



Q Construct NFA to accept set of strings over $\{a, b\}$ which starts with a.



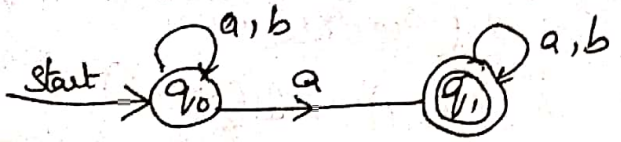
Consider input ab the transitions are $q_0 \xrightarrow{a} q_2 \xrightarrow{b} q_2$

Since the transitions of input reaches final state it is accepted

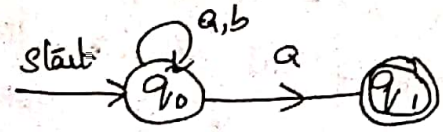
Consider another input ba the transition are $q_0 \xrightarrow{b} \phi$ since the transition of input do not end with any final or accepting state it is Rejected.

Q Construct NFA to accept set of all strings over {a,b} which contain a

$L = \{a, aa, ab, aaa, abb \ldots\}$

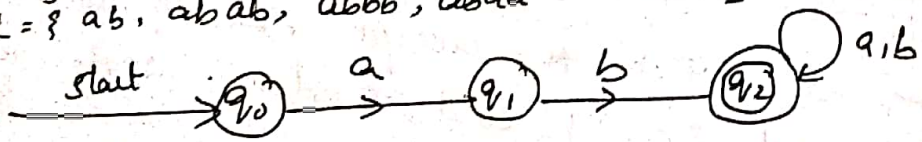


Q Construct NFA to accept set of all strings over {a,b} which ends with a



Q Construct NFA to accept set of all strings over {a,b} which starts with 'ab'

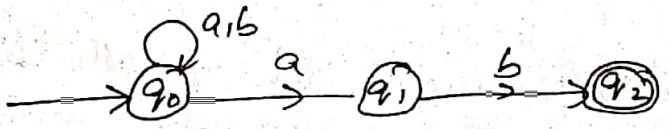
$L = \{ab, abab, abbb, abaa \ldots\}$



Q Construct NFA to accept set of all strings over {a,b} which ~~starts~~ contains ab as substring

$L = \{ab, aaba, baba, abab \ldots\}$

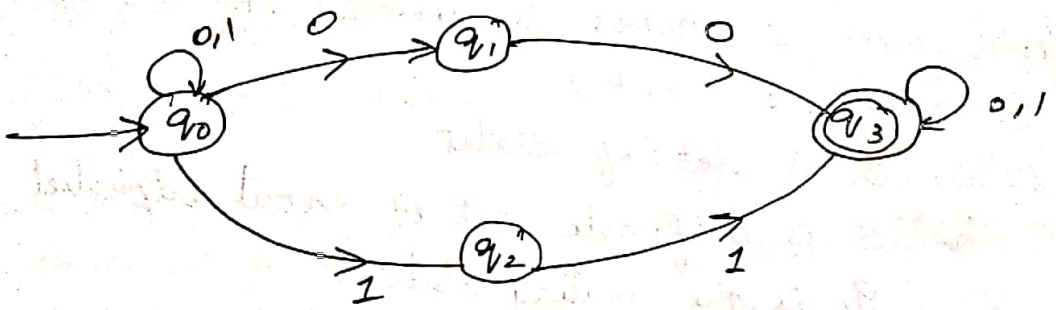
Q Construct NFA to accept set of all strings over {a,b} which ends with ab



Q Design an NFA which accept string 1100 only

$L = \{1100\}$

Q Design a NFA to accept the string with 0's and 1's such that string contains either two consecutive 0's or two consecutive 1's



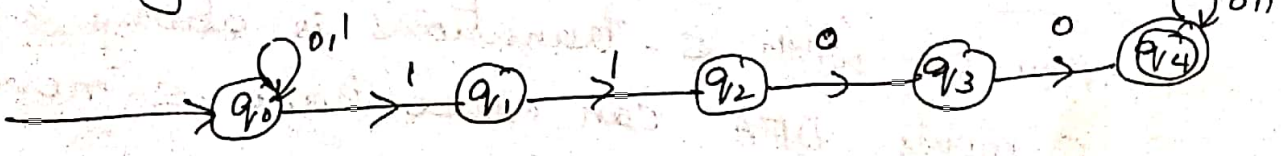
Consider the input 10100

the transition of input is

$q_0 \xrightarrow{1} q_0 \xrightarrow{0} q_0 \xrightarrow{1} q_0 \xrightarrow{0} q_0 \xrightarrow{0} q_0$, $q_0 \xrightarrow{1} q_2 \xrightarrow{0} q_1 \xrightarrow{0} q_3$

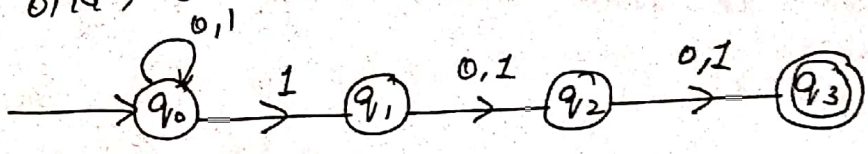
Since the transition ends with final state the given input is accepted. ie the final path for the input 10100 is $q_0 \xrightarrow{1} q_0 \xrightarrow{0} q_0 \xrightarrow{1} q_0 \xrightarrow{0} q_1 \xrightarrow{0} q_3$

Q Design NFA which accepts set of all strings containing 1100 as substring



Q Design NFA which accepts set of all strings containing third symbol from right side is 1.

$L = \{ 01\mathbb{1}, 0100, 0101, 0110 - - - \}$



4/8/17

Finite Automata with Epsilon-Transitions

Finite Automata with ϵ-transitions is possible only in NFA. So FA with ϵ-transitions is an NFA including transitions on the empty input that is ϵ

Formal definition :-

NFA with ϵ-moves is denoted by 5-tuple notation $(M = Q, \Sigma, \delta, q_0, F)$

where Q is set of states

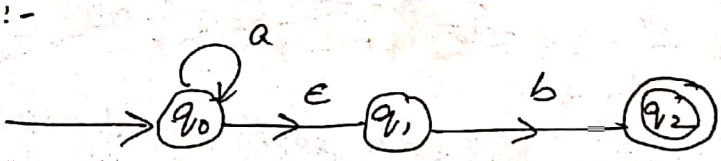
Σ is a finite set of input alphabet

q_0 is the initial state

δ is the transition function which maps

$$Q \times (\Sigma \cup \{\epsilon\}) \rightarrow 2^Q$$

eg :-



The meaning of epsilon transition is a transition on empty input of length 0. In the above figure without reading any input symbol the transition moves from state q_0 to state q_1

NB :-

Any FA with ϵ-transitions is always a NFA with ϵ-moves. DFA can never have ϵ-moves.

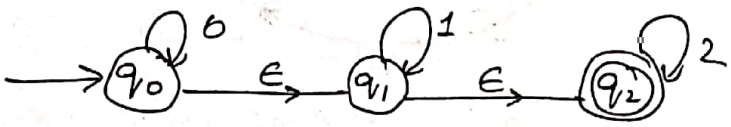
ϵ-closure

ϵ-closure is the property of ϵ-NFA. ϵ-closure(q_0) is defined as the set of all states p such that there is a path from q_0 to p with label ϵ.

or,

ϵ-closure(q_0) is the set of all states which can be reachable by ϵ from the state q_0.

eg:



ϵ-closure(q_0) = $\{q_0, q_1, q_2\}$

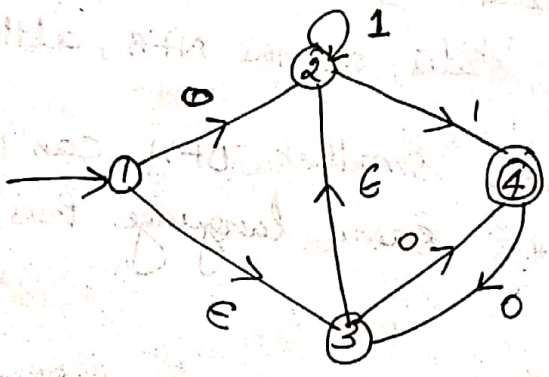
ϵ-closure(q_1) = $\{q_1, q_2\}$

ϵ-closure(q_2) = $\{q_2\}$

NB:-

ϵ-closure of a state is that state itself and the set of states reachable on ϵ transition alone.

Q Find the ϵ-closure for all states.



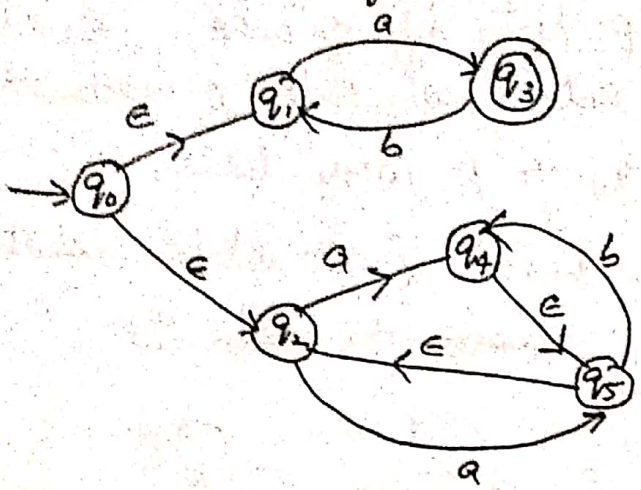
ϵ-closure(1) = {1, 3, 2}

ϵ-closure(2) = {2}

ϵ-closure(3) = {3, 2}

ϵ-closure(4) = {4}

Q Find the ε- closure of all states.



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 $(q_4) = \{q_4, q_5, q_2\}$

ε-closure $(q_5) = \{q_5, q_2\}$

Equivalence of NFA and DFA

Every language that can be described by some NFA can also be described by some DFA.

DFA has as many states as the NFA, although it has more transitions

In the worst case the smallest DFA can have 2^n states if the NFA for the same language has only n states.

Equivalence of NFA and DFA can be proved by the subset construction method because the DFA involves contains the subset of the set of states of NFA.

Theorem:-

Let L be a language accepted by a NFA. Then there exists a DFA that accepts L.

proof:-

Let $N = (Q_N, \Sigma, \delta_N, q_0, F_N)$ be the NFA that accepts the language L.

And $D = (Q_D, \Sigma, \delta_D, \{q_0\}, F_D)$ be the DFA that accepts the same language L.

The states of D are all possible non-empty subsets of the states of N.

ie $Q_D = 2^{Q_N}$

An element in Q_D is denoted by $[q_1, q_2 \dots q_i]$ where $q_1, q_2 \dots q_i$ are in Q

Define $\delta_D([q_1, q_2, \dots q_i], a) = [p_1, p_2 \dots p_j]$

iff $\delta_N(\{q_1, q_2 \dots q_i\}, a) = \{p_1, p_2 \dots p_j\}$

ie δ_D is computed by applying δ_N to each state of Q_D which is represented by $[q_1, q_2 \dots q_i]$

On applying δ_N to each of $q_1, q_2 \dots q_i$ and taking the union, we get some new states $p_1, p_2 \dots p_j$. The new states can be represented by $[p_1, p_2 \dots p_j] \in Q_D$

Now we have to show $\delta_D(q_0, x) = [p_1, p_2 \dots p_j]$ is equivalent to $\delta_N(q_0, x) = \{p_1, p_2 \dots p_j\}$. This must be prove by induction on the length of the input string x.

BASIS: If w is of length zero. then $\delta_D(q_0, w=\epsilon) = [q_0]$
iff $\delta_N(q_0, \epsilon) = q_0$

INDUCTION: Suppose that the above result is true for the input of length n

Now we prove the results for the string of length $n+1$ with $a \in \Sigma$. Then by induction assume

$$\delta_D(\{q_0\}, xa) = \delta_D(\delta_D(\{q_0\}, x), a)$$
$$= \delta_D[p_1, p_2 \cdots p_j], a) \quad \text{iff } \delta_D(\{q_0\}, x) = p_1, p_2 \cdots p_j$$
$$= [r_1, r_2 \cdots r_k]$$

If and only if $\delta_N(\{p_1, p_2 \cdots p_j\}, a) = \{r_1, r_2 \cdots r_k\}$

Thus $\delta_D(\{q_0\}, xa) = [r_1, r_2 \cdots r_k]$

iff $\delta_N(q_0, xa) = \{r_1, r_2 \cdots r_k\}$

Since the result is true for $n+1$. It is also true for the input of length n. Thus N & D both accepts the same string w iff $\delta_D(\{q_0\}, x)$ or $\delta_N(q_0, x)$ contains a state in F

Conversion of NFA to DFA (Subset Construction method)

Algorithm to convert NFA to DFA :-

Let the NFA is $N = (Q, \Sigma, \delta, q_0, F)$ and the equivalent DFA is $D = (Q', \Sigma, \delta', q_0', F)$ and $Q' = 2^Q$.

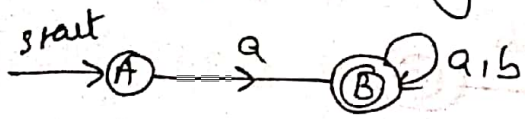
F' is set of all subsets of Q which are having Final state of given NFA.

δ' is calculated as

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

Then $\delta'(q,a) = \delta(q_0,a) \cup \delta(q_1,a)$

Q Convert the following NFA To DFA



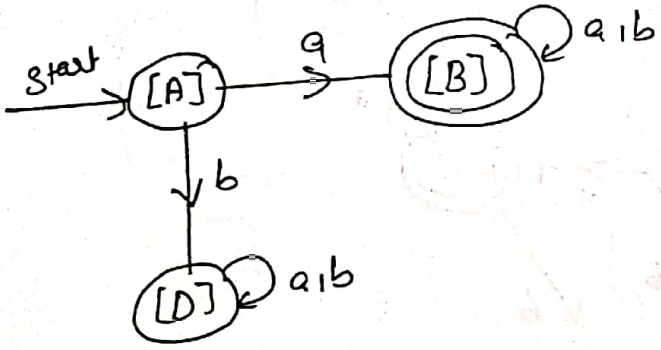
STT of NFA

	a	b
→ A	{B}	{}
B*	{B}	{B}

STT of DFA

	a	b
→ [A]	[B]	[D]
[B]	[B]	[B]
[D]	[D]	[D]

The equivalent DFA is



Q Convert NFA to DFA

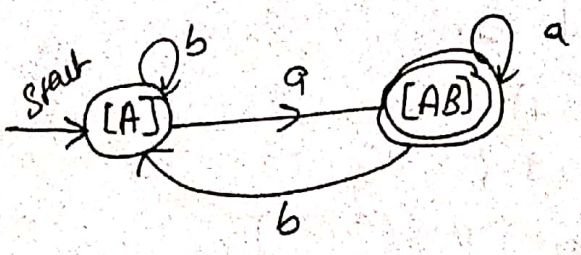
L_1 = { ends with a }

STT of NFA

	a	b
→ A	{A,B}	{A}
B*	{}	{}

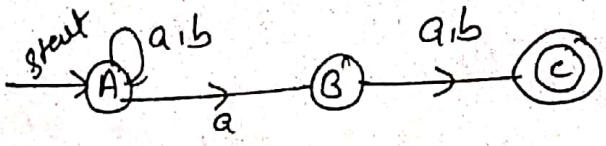
STT of DFA

	a	b
→ [A]	[AB]	[A]
[AB]*	[AB]	[A]



Q Convert NFA to DFA

$L = \{aa, ab, aaa, bab \ldots\}$

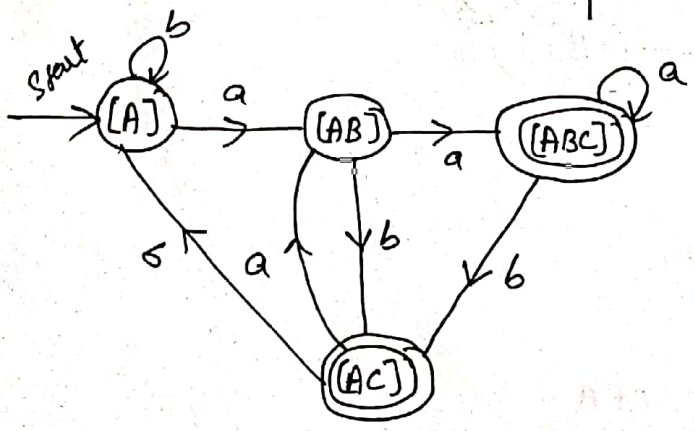


STT of NFA

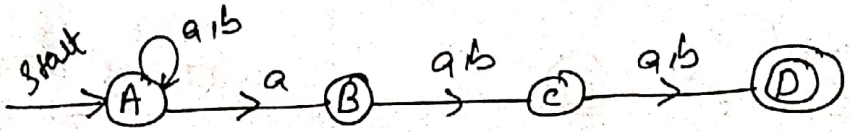
	a	b
→ A	{A,B}	{A}
B	{C}	{C}
C*	{ }	{ }

STT of DFA

	a	b
→ A	[AB]	[A]
[AB]	[ABC]	[AC]
[ABC]*	[ABC]	[AC]
[AC]*	[AB]	[A]



Q Convert NFA to DFA

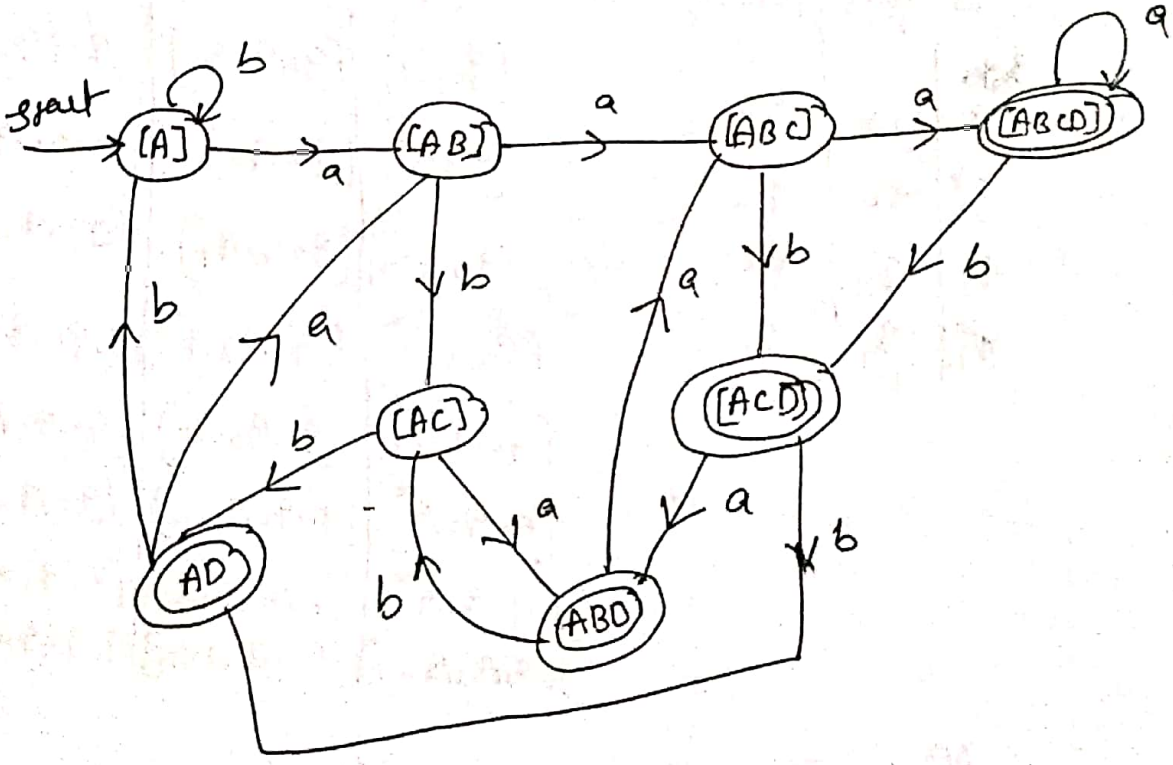


STT of NFA

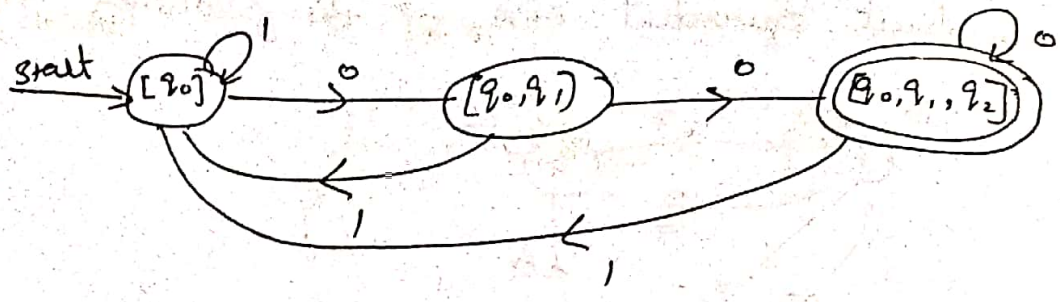
	a	b
→ A	{A,B}	{A}
B	{C}	{C}
C	{D}	{D}
D*	{ }	{ }

STT of DFA.

	a	b
→ A	[AB]	[A]
[AB]	[ABC]	[AC]
[ABC]	[ABCD]	[ACD]
[AC]	[ABD]	[AD]
[ABCD]*	[ABCD]	[ACD]
[ACD]*	[ABD]	[AD]
[ABD]*	[ABC]	[AC]
[AD]*	[AB]	[A]



NB:
If a minimal NFA contain 'n' states then the minimal DFA contain 2^n states.



Equivalence and Conversion of NFA With and Without ϵ-Transitions

Conversion of ϵ-NFA to NFA:-

Let $M_1 = (Q, \Sigma, \delta, q_0, F)$ be the NFA with ϵ-transition and $M_2 = (Q, \Sigma, \delta', q_0, F')$ be the NFA without ϵ-transition

We have to convert M_1 to M_2

Following are the step to convert M_1 to M_2

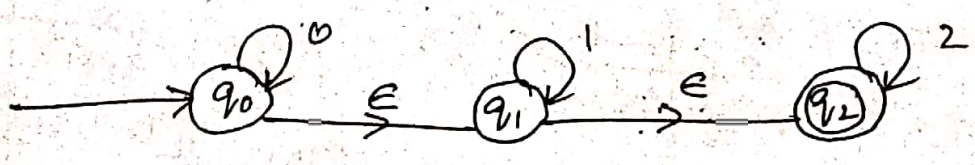
1. Find the ϵ-closure of each state q in M_1. If it contains any final state in M_1, then the final state of NFA without ϵ-transition is $F' = \{F \cup \{q\}\}$. otherwise $F' = F$

2. For all state $q \in Q$, and a input $a \in \Sigma$. find

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

where $\hat{\delta}(q,a) = \epsilon\text{-closure}(\delta(\epsilon\text{-closure}(q), a))$

Q. Convert the following ϵ-NFA to NFA



ϵ-closure $(q_0) = \{q_0, q_1, q_2\}$

ϵ-closure $(q_1) = \{q_1, q_2\}$

ϵ-closure $(q_2) = \{q_2\}$

So the new final state $f' = \{q_0, q_1, q_2\}$

$\delta'(q_0, 0) = \epsilon$-closure $(\delta(\epsilon$-closure$(q_0), 0)$

$= \epsilon$-closure $(\delta(q_0, q_1, q_2), 0)$

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

$= \epsilon$-closure (q_0)

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

$\delta'(q_0, 1) = \epsilon$-closure $(\delta(\epsilon$-closure $(q_0), 1))$

$= \epsilon$-closure $(\delta(q_0, q_1, q_2), 1))$

$= \epsilon$-closure $(\delta(q_0, 1) \cup \delta(q_1, 1) \cup \delta(q_2, 1))$

$= \epsilon$-closure $(\phi \cup q_1 \cup \phi\}$

$= \epsilon$-closure (q_1)

$= \{q_1, q_2\}$

$\delta'(q_0, 2) = \epsilon$-closure $(\delta(\epsilon$-closure $(q_0), 2))$

$= \epsilon$-closure $(\delta(q_0, q_1, q_2), 2))$

$= \epsilon$-closure $(\delta(q_0, 2) \cup \delta(q_1, 2) \cup \delta(q_2, 2))$

$= \epsilon$-closure (q_2)

$= \{q_2\}$

$\delta'(q_1, 0) = \epsilon$-closure $(\delta(\epsilon$-closure $(q_1), 0))$

$= \epsilon$-closure $(\delta(q_1, q_2), 0))$

$= \epsilon$-closure $(\delta(q_1, 0) \cup \delta(q_2, 0))$

$= \epsilon$-closure $(\phi \cup \phi)$

$= \phi$

$\delta'(q_1,1) = \epsilon\text{-closure}(\delta(\epsilon\text{-closure}(q_1),1))$

$= \epsilon\text{-closure}(\delta(q_1,q_2),1)$

$= \epsilon\text{-closure}(\delta(q_1,1) \cup \delta(q_2,1)$

$= \epsilon\text{-closure}(q_1 \cup \phi)$

$= \{q_1,q_2\}$

$\delta'(q_1,2) = \epsilon\text{-closure}(\delta(\epsilon\text{-closure}(q_1),2))$

$= \epsilon\text{-closure}(\delta(q_1,q_2),2))$

$= \epsilon\text{-closure}(\delta(q_1,2) \cup \delta(q_2,2)$

$= \epsilon\text{-closure}(\phi \cup q_2)$

$= \{q_2\}$

$\delta'(q_2,0) = \epsilon\text{-closure}(\delta(\epsilon\text{-closure}(q_2),0)$

$= \epsilon\text{-closure}(\delta(q_2,0))$

$= \phi$

$\delta'(q_2,1) = \epsilon\text{-closure}(\delta(\epsilon\text{-closure}(q_2),1)$

$= \epsilon\text{-closure}(\delta(q_2,1))$

$= \phi$

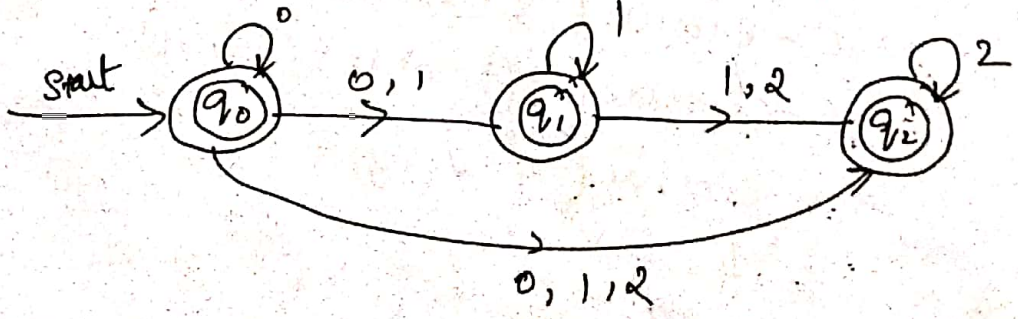
$\delta(q_2,2) = \epsilon\text{-closure}(\delta(\epsilon\text{-closure}(q_2),2)$

$= \epsilon\text{-closure}(\delta(q_2,2))$

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

$= \{q_2\}$

NFA Without ϵ-transition

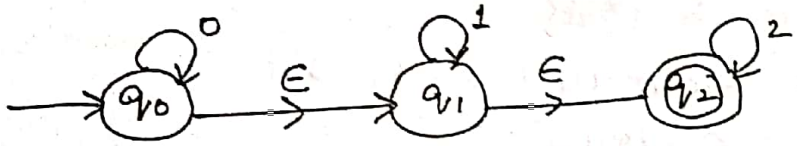


* Q_D is the set of subsets of Q_E.
* $q_D = \epsilon$-closure(q_0), that is the start state of D is the ϵ-closure of the start state of E.

NB: This rule differs from the original subset construction, where the start state of the constructed automaton was just the set containing the start state of the given NFA.

* F_D is those set of states that contain at least one accepting state of E.
* $\delta_D(S, a) = \epsilon\text{-closure}(\delta(S, a))$ where S is the subset of E.

Q Convert the following ϵ-NFA to DFA



The start state of the DFA can be found out by ϵ-closure of the start state of given ϵ-NFA.

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

The transition of the input is as follows

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

$\delta_D(\{q_0, q_1, q_2\}, 1) = \epsilon\text{-closure}(\delta(q_0, 1) \cup \delta(q_1, 1) \cup \delta(q_2, 1))$
$= \epsilon\text{-closure}(\phi \cup q_1 \cup \phi)$
$= \epsilon\text{-closure}(q_1)$.
$= \{q_1, q_2\}$

$\delta_D(\{q_0,q_1,q_2\},2) = \epsilon\text{-closure}(\delta(q_0,2) \cup \delta(q_1,2) \cup \delta(q_2,2))$

$= \epsilon\text{-closure}(\phi \cup \phi \cup q_2)$

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

$= \{q_2\}$

$\delta_D(\{q_1,q_2\},0) = \epsilon\text{-closure}(\delta(q_1,0) \cup \delta(q_2,0))$

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

$= \phi$

$\delta_D(\{q_1,q_2\},1) = \epsilon\text{-closure}(\delta(q_1,1) \cup \delta(q_2,1))$

$= \epsilon\text{-closure}(q_1 \cup \phi)$

$= \epsilon\text{-closure}(q_1)$

$= \{q_1,q_2\}$

$\delta_D(\{q_1,q_2\},2) = \epsilon\text{-closure}(\delta(q_1,2) \cup \delta(q_2,2))$

$= \epsilon\text{-closure}(\phi \cup q_2)$

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

$= \{q_2\}$

$\delta(\{q_2\},0) = \epsilon\text{-closure}(\delta(q_2,0))$

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

$= \phi$

$\delta(\{q_2\},1) = \epsilon\text{-closure}\,\delta(q_2,1))$

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

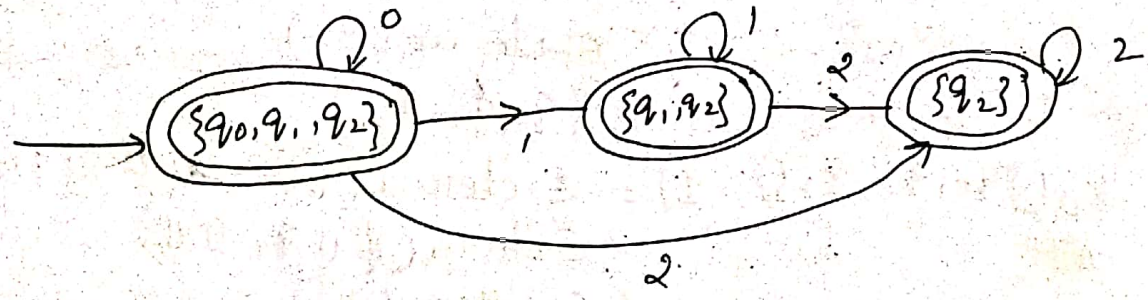
$= \phi$

$\delta(\{q_2\},2) = \epsilon\text{-closure}\,\delta(q_2,2))$

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

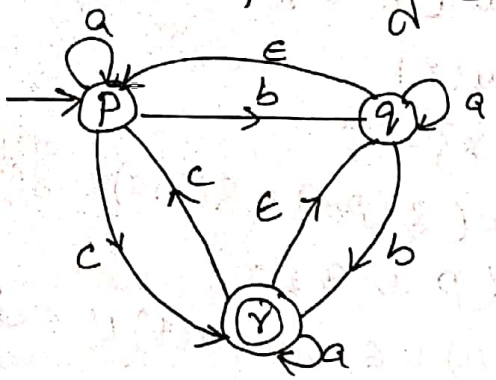
$= \{q_2\}$

The equivalent DFA for the given ϵ-NFA is



The final states of the above DFA is equal to $\{q_0, q_1, q_2\}$, $\{q_1, q_2\}$, $\{q_2\}$ because the final state of the ϵ-NFA $\{q_2\}$ state is present in all the three subsets of DFA.

Q Convert the following ϵ-NFA to DFA



or

	ε	a	b	c
→p	ϕ	{p}	{q}	{r}
q	{p}	{q}	{r}	ϕ
r*	{q}	{r}	ϕ	{p}

The start state of DFA is

ϵ-closure start state in ϵNFA, ie

ϵ-closure(p) = {p} ⇒ start state of DFA

$\delta(p, a) = \epsilon\text{-closure}(\delta(p, a))$

$= \epsilon\text{-closure}(p)$

$= \{p\}$ ✓

$\delta(p, b) = \epsilon\text{-closure}(\delta(p, b))$

$= \epsilon\text{-closure}(q)$

$= \{q, p\}$ ✓

$\delta(p, c) = \epsilon\text{-closure}(\delta(p, c))$

$= \epsilon\text{-closure}(r)$

$= \{r, q, p\}$ ✓

$\delta(\{p, q\}, a) = \epsilon\text{-closure}(\delta(p, a) \cup \delta(q, a))$

$= \epsilon\text{-closure}(p \cup q)$

$= \epsilon\text{-closure } p \cup \epsilon\text{-closure } q$

$= \{p, q\}$ ✓

$\delta(\{p,q\},b) = \epsilon\text{-closure}(\delta(p,b) \cup \delta(q,b))$

$= \epsilon\text{-closure}(q \cup r)$

$= \epsilon\text{-closure}(q) \cup \epsilon\text{-closure}(r)$

$= \{p,q,r\}$ ✓

$\delta(\{p,q\},c) = \epsilon\text{-closure}(\delta(p,c) \cup \delta(q,c))$

$= \epsilon\text{-closure}(r \cup \phi)$

$= \epsilon\text{-closure}(r)$

$= \{p,q,r\}$ ✓

$\delta(\{p,q,r\},a) = \epsilon\text{-closure}(\delta(p,a) \cup \delta(q,a) \cup \delta(r,a))$

$= \epsilon\text{-closure}(p \cup q \cup r)$

$= \epsilon\text{-closure}(p) \cup \epsilon\text{-closure}(q) \cup \epsilon\text{-closure}(r)$

$= \{p,q,r\}$ ✓

$\delta(\{p,q,r\},b) = \epsilon\text{-closure}(\delta(p,b) \cup \delta(q,b) \cup \delta(r,b))$

$= \epsilon\text{-closure}(q \cup r \cup \phi)$

$= \epsilon\text{-closure}(q) \cup \epsilon\text{-closure}(r)$

$= \{p,q,r\}$ ✓

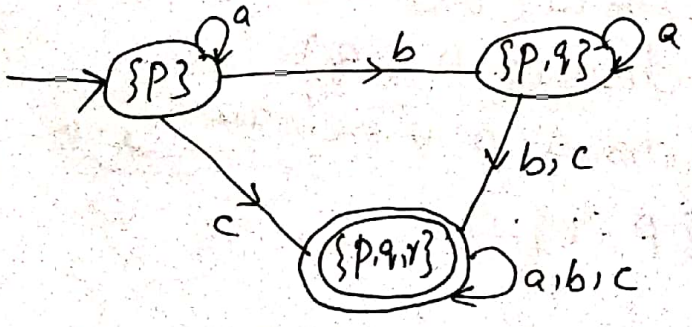
$\delta(\{p,q,r\},c) = \epsilon\text{-closure}(\delta(p,c) \cup \delta(q,c) \cup \delta(r,c))$

$= \epsilon\text{-closure}(r \cup \phi)$

$= \epsilon\text{-closure}(r) \cup \epsilon\text{-closure}(\phi)$

$= \{p,q,r\}$ ✓

The states in DFA are $\{p\}$, $\{p,q\}$ and $\{p,q,r\}$ of which final state is $\{p,q,r\}$ because the final state of ϵ-NFA 'r' is present in the subset $\{p,q,r\}$



Equivalent DFA

Equivalence of NFA With and Without Epsilon Transition

Theorem:-

If the language L is accepted by a NFA with ϵ-transitions, then L is accepted by an NFA without ϵ-transitions

Proof:-

Let $M = (Q, \Sigma, \delta, q_0, F)$ be an NFA with ϵ-transitions and $M' = (Q, \Sigma, \delta', q_0, F')$ be the NFA without ϵ-transitions.

where

$$F' = \begin{cases} F \cup \{q_0\} & \text{if } \epsilon\text{-closure}(q_0) \text{ contains a state in } F \\ F & \text{otherwise} \end{cases}$$

$\delta'(q, a) = \hat{\delta}(q, a)$; $q \in Q$ & $a \in \Sigma$

By induction $\delta'[q_0, w] = \hat{\delta}(q_0, w)$ for any string w.

For $w = \epsilon$, $\delta'(q_0, \epsilon) = \{q_0\}$ and $\hat{\delta}(q_0, \epsilon) = \epsilon\text{-closure}\{q_0\}$

So $\delta'(q_0, \epsilon) \neq \hat{\delta}(q_0, \epsilon)$

So the induction is not true for the string having length 0.

Basis:-

Let a be the string of length one. That is $a \in \Sigma$.

$\delta'[q_0, a] = \hat{\delta}(q_0, a)$ by definition of δ'

Induction:-

Assume that $\delta'(q_0, x) = \hat{\delta}(q_0, x)$ for any string x of length $\leq n$. Assume w is a string of length $n+1$.

Let $w = xa$ for some symbol $a \in \Sigma$. Then

$$\delta'(q_0, w) = \delta'(q_0, xa)$$
$$= \delta'(\delta'\{q_0, x\}, a)$$
$$= \delta'(\hat{\delta}(q_0, x), a)$$
$$= \delta'(p, a) \quad \text{where } p = \hat{\delta}(q_0, x)$$

We have to show that $\delta'(p, a) = \hat{\delta}(q_0, xa)$

$$\text{i.e. } \delta'(p, a) = \hat{\delta}(\hat{\delta}(q_0, x), a)$$
$$= \hat{\delta}(q_0, xa)$$
$$= \hat{\delta}(q_0, w)$$
$$\delta'(p, a) = \hat{\delta}(q_0, w)$$

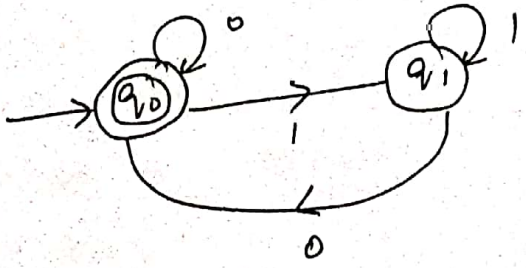
Q Construct a minimal DFA, which accepts set of all string over $\{0, 1\}$ which when interpretted as binary number divisible by 2.

$\Sigma = \{0, 1\}$ $w \in \{0, 1\}^*$

eg: 110
$1 \times 2 + 1$
$= 3 \times 2 + 0 = 6$

101
$1 \times 2 + 0$
$= 2 \times 2 + 1 = 5$

When a number divide by 2. we get two remainder 0 and 1



	0	1
→ q_0^*	q_0	q_1
q_1	q_0	q_1

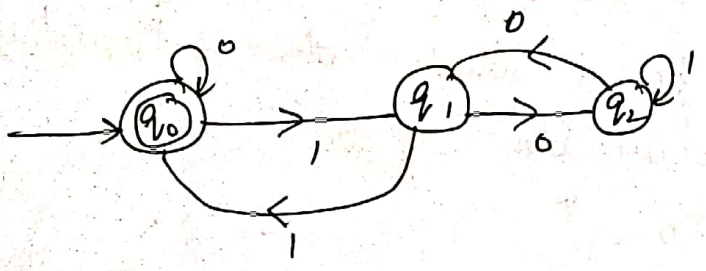
NB:-
If the question is DFA is divisible by n in case of binary no: then no: of states in DFA is n states

Shortcut method:-

* Write down the states in the transition table equal to the number divisible in the given question
* Write all inputs
* Write all states in order and fill transitions table.

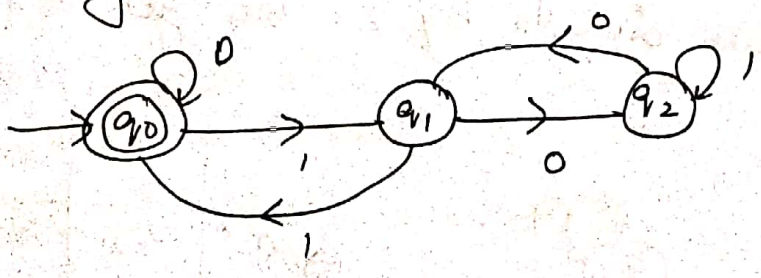
Q Construct a DFA over $\Sigma = \{0,1\}$ of binary numbers which is divisible by 3.

$\Sigma = \{0,1\}$ Remainder = 0,1,2



	0	1
→ q_0*	q_0	q_1
q_1	q_2	q_0
q_2	q_1	q_2

Q Construct a DFA over $\Sigma = \{0,1\}$ in which binary number $\cong$ 0 mod 3



MOD-II

Finite state Machines with output

Machines which can be formulated as FA with out is classified into two types

i) Moore Machine

ii) Mealy Machine

Moore Machine

It is the machine with finite number of state an for which, the output symbol depends only upon the present state of the machine.

The moore machine consists of 6-tuple

$M = (Q, \Sigma, \Delta, \delta, q_0, \lambda)$ where

Q - Finite set of states.

Σ - input symbols or alphabet

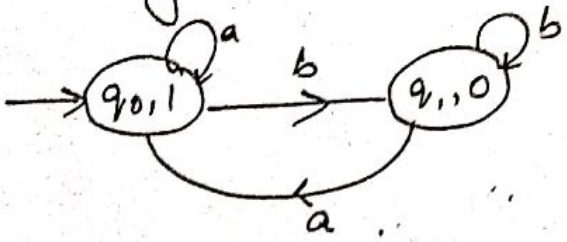
Δ - an output alphabet

λ - output function $(Q \to \Delta)$

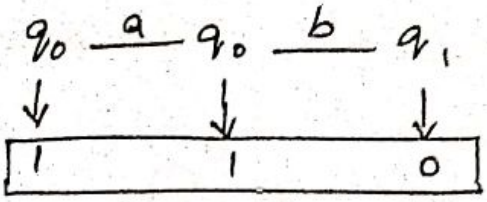
δ - transition function $(Q \times \Sigma \to Q)$

q_0 - starting state.

eg:



In the above moore machine, the state q_0 provides the output 1 and the state q_1 provides the output 0. For the input string ab in the machine, the traversal takes place as follows.



The output for the string ab is '110'

Note:-

The Moore machines produces n+1 output symbols for 'n' input symbols.

Mealy Machine

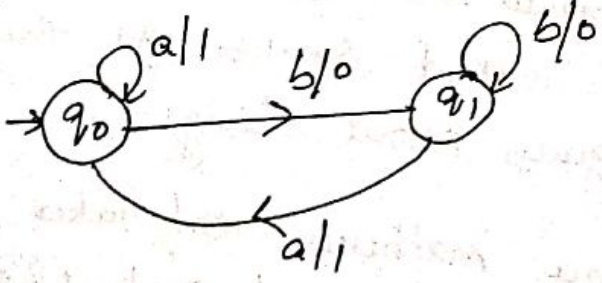
It is the machine with finite number of states and for which the output symbol is the function of the present input symbol as well as the present state of the machine.

A mealy machine is denoted by 6-tuple of the form

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

Q - set of states
Σ - input alphabet
Δ - output alphabet
δ - transition function $(Q \times \Sigma \to Q)$
λ - output function $(Q \times \Sigma \to \Delta)$
q_0 - initial state

eg:-



$$\lambda : Q \times \Sigma \rightarrow \Delta$$

$(q_0, a) \rightarrow 1$

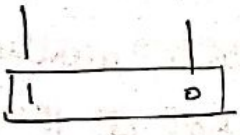
$(q_0, b) \rightarrow 0$

$(q_1, a) \rightarrow 1$

$(q_1, b) \rightarrow 0$

for the i/p 'ab' the processing is carried out in the above mealy machine as follows.

$q_0 \xrightarrow{a} q_0 \xrightarrow{b} q_1$



The output for the string ab is '10'

Note:-

The mealy machine produces 'n' output symbols for the 'n' input symbols.

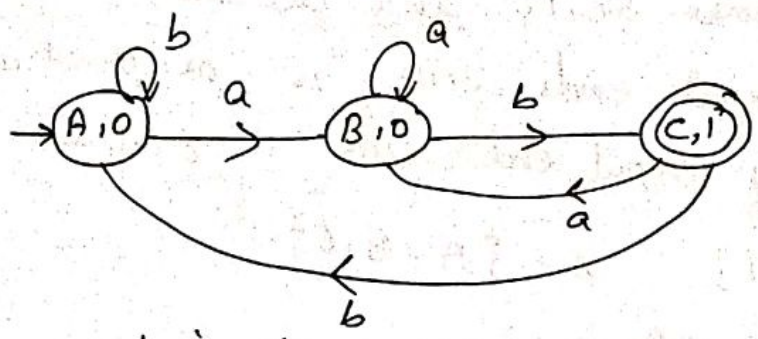
Difference between Moore and Mealy Machines:-

- Moore machines print character when in state.
- Mealy machines print character when traversing an arc.

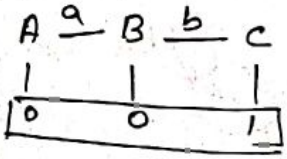
Both moore and mealy machines has no final states. These machines provide some output only. In both moore and mealy machines the process halts after producing the output symbol for the last input symbol of given input strings

Q Construct Moore machines that takes set of all strings over $\{a, b\}$ as input and print 1 as output for every occurence of ab as a substring

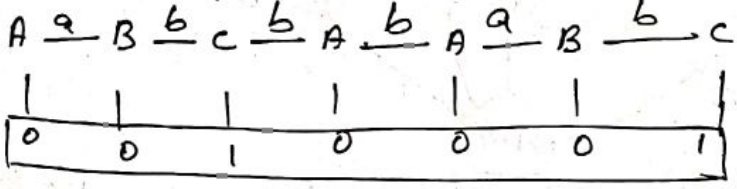
$\Sigma = \{a, b\}$ $\Delta = \{0, 1\}$



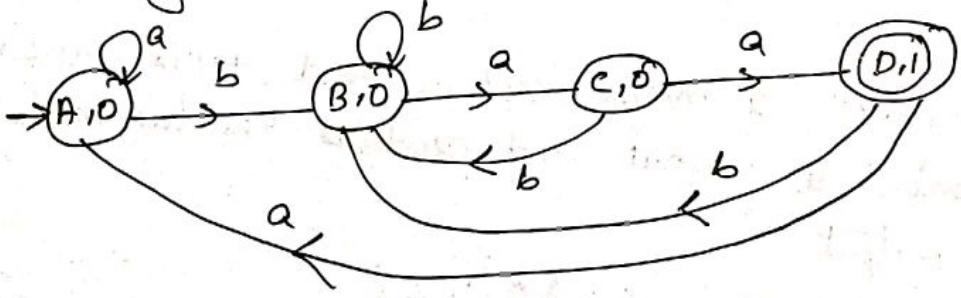
for the input 'a b'



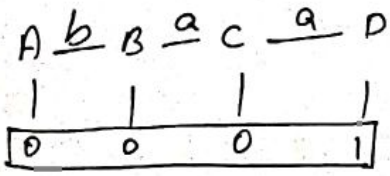
for the input abbbab



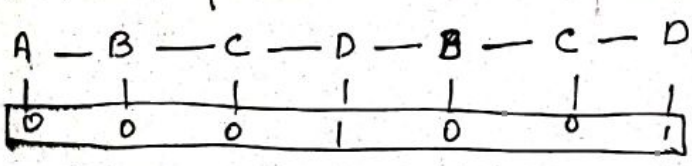
Q Construct a Moore machine that takes set of all strings over $\{a, b\}$ as input and print 1 as output for every occurence of baa as a substring



for the input baa the o/p is as follows



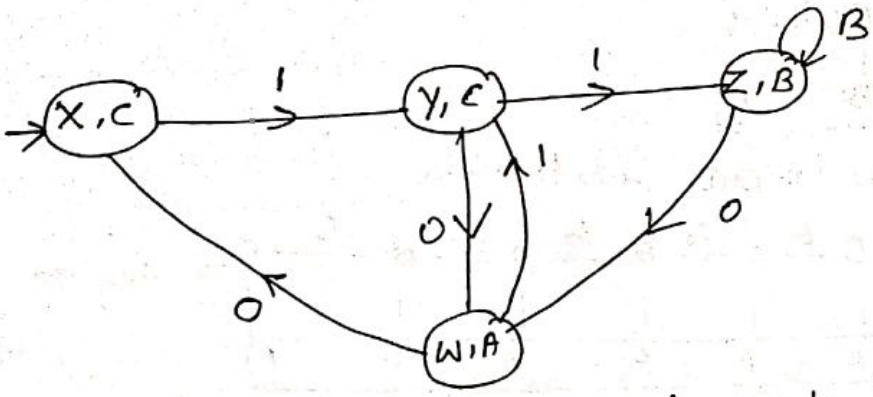
for the input baabaa



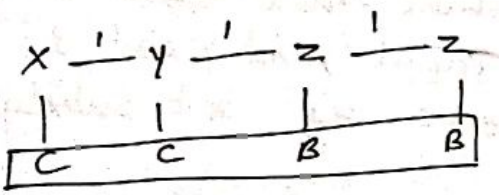
Q Construct moore machine that takes set of all strings over {0,1} and produces 'A' as output if the input ends with '10' or produces 'B' as output if input ends with '11' otherwise produces 'C'

$\Sigma = \{0, 1\}$ $\Delta = \{A, B, C\}$

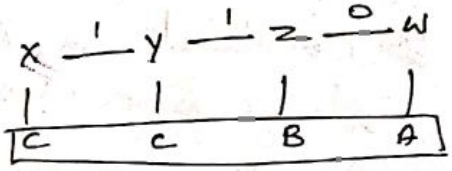
10 – A
11 – B



for input '111'



for input '110'



Q Construct a moore machine that takes binary number as input and produces Residue modulo 3 as output.

$\Sigma = \{0, 1\}$ $\Delta = \{0, 1, 2\}$

	0	1	Δ
q_0	q_0	q_1	0
q_1	q_2	q_0	1
q_2	q_1	q_2	2

Regular Expressions

Regular Expression is another type of language defining noration. The language accepted by deterministic and non-deterministic finite automata can easily be described by Regular expression

Regular expression is used as an input language in many systems that process strings as input.

Operations of Regular expression

There are three operations on languages that the operators of Regular expression represent. These operations are

(i) Union
(ii) Concatenation
(iii) closure

(i) Union :-

The union of two language L and M is denoted by L∪M which is the set of string that are in either L or M or both

eg: L = {001, 10, 111}
M = {∈, 001}
L∪M = {∈, 10, 001, 111}

(ii) Concatenation :-

The concatenation of language L and M is the set of strings that can be formed by taking any string in L and concatenating it with any

String in M. We can denote concatenation of languages either with a dot or with no operator at all.

eg:- $L = \{001, 10, 111\}$

$M = \{\epsilon, 001\}$

$L \cdot M$ or $LM = \{001, 10, 111, 001001, 10001, 111001\}$

The first 3 strings in LM are the strings in L concatenated with ϵ. Since ϵ is the identity for concatenation, the resulting strings are the same as the strings of L. The last 3 strings in LM are formed by taking each string in L and concatenate it with the second string in M, which is 001

iii) Closure

The closure or kleene closure of a language L is denoted by L^* and represents the set of those strings that can be formed by taking any number of strings from L possibly with repetition.

$$L^* = \bigcup_{i=0}^{\infty} L^i$$

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

$$L^i = L L^{i-1} \text{ for } i \geq 1$$

eg: $L = \{0, 11\}$

$L^* = \{0, 00, \epsilon, 0110, 011111 \cdots\}$

Construction of Regular Expression

1. The constants ϵ and ϕ are regular expressions denoting the language $\{\epsilon\}$ and ϕ respectively.

That is $L(\epsilon) = \{\epsilon\}$ and $L(\phi) = \phi$

2. If a is any symbol, then a is a Regular expression. This expression denotes the language $\{a\}$. That is $L(a) = \{a\}$

3. A variable represented in upper case like 'L' is any language.

4. If E and F are Regular expressions, then E+F is a Regular expression denoting the Union of L(E) and L(F). That is $L(E+F) = L(E) \cup L(F)$

5. If E and F are Regular expressions, then EF is a Regular expression denoting the concatenation of L(E) and L(F). That is $L(EF) = L(E)\,L(F)$.

6. If E is a Regular expression then E^* is a Regular expression denoting the closure of L(E). That is $L(E^*) = (L(E))^*$

7. If E is a Regular expression, then (E), a parenthesized E, is also a Regular expression, denoting the same language as E. Formally $L((E)) = L(E)$

Precedence of Regular Expression

Like other algebra, Regular expressions operators have an assumed order of precedence which means that operators are associated with their

operands in a particular order

i) The star operator is of highest precedence.

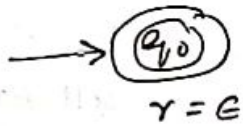
ii) Next precedence comes the concatenation or 'dot' operator. After grouping star to their operands. concatenation operator with their operands are grouped together.

iii) Third precedence goes to Union operator and is grouped with their operands.

eg: The expression $01^* + 1$ is grouped as $(0(1^*)) + 1$

Following are the some examples of Representing RE

(i): ϵ - denotes the empty string

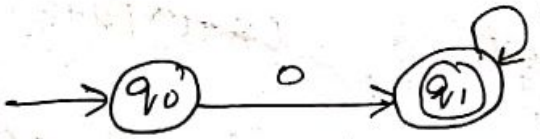


ii) 0^* - denotes the language of string of any number of 0's.

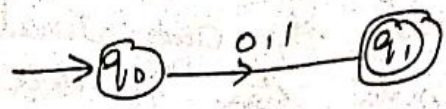
$L(0^*) = \{\epsilon, 0, 00, 000 \cdots\}$

iii) 01^* - denotes the language of strings of 0's and 1's and it should begin with zero followed by any number of 1's

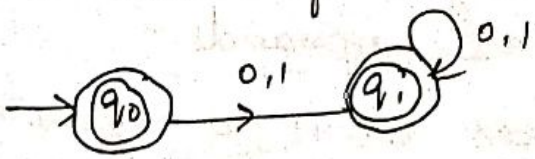
$L = \{0, 01, 011, 0111 \cdots\}$



iv) $(0+1)$ - denotes the language of strings of either 0 or 1



v) $(0+1)^*$ - denotes the language of strings of 0's and 1's where the expression consists of any number of 0's or any number of 1's.



Examples of RE:-

Q Construct RE of all strings over $\{a,b\}$ whose length is exactly 2.

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

L is finite. So take union of all possible string

i.e $aa+ab+ba+bb$

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

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

So $RE = (a+b)(a+b)$

Q Construct RE of all strings over $\{a,b\}$ whose length is exactly 3

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

RE for string of length 4 $= (a+b)(a+b)(a+b)(a+b)$

Q Construct RE of string of length atleast 2

$L = \{aa, ab, ba, bb, aaa \ldots\}$ L is infinite

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

Equivalence of RE and NFA with ε-transitions

Theorem: Every language defined by a regular expression is also defined by a finite automaton.

Proof: Suppose $L = L(R)$ for a regular expression R. We have to show that $L = L(E)$ for some ε-NFA E with

i) Exactly one accepting state
ii) No arcs into the initial state
iii) No arcs out of the accepting state

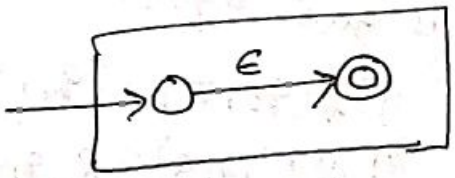
BASIS:

There are three parts to the basis

a) Construction of automaton for the RE ε
b) Construction of automaton for the RE ϕ
c) Construction of automaton for the RE 'a'

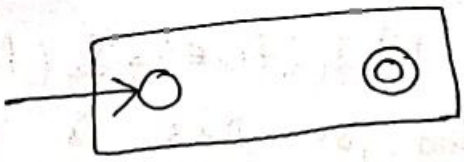
The automaton of the above three parts is as given below

for ε



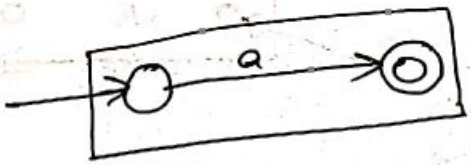
ε is a RE denotes the set {ε}

for ϕ



ϕ is a RE denotes an empty set

for 'a'



a is a RE denotes the set {a}

All the above 3 automatons in the basis part consists of only one accepting state, no edges into the initial state and no arcs out of the accepting state.

INDUCTION

Assume that the statement of theorem is true for the immediate subexpression of the given Regular expression. There are three cases

Case 1:

Consider the expression $R+S$

and consider the NFA for the expression R as M_1 and

$M_1 = (Q_1, \Sigma_1, \delta_1, q_1, \{f_1\})$

and consider the NFA for the expression S as M_2 and

$M_2 = (Q_2, \Sigma_2, \delta_2, q_2, \{f_2\})$

The final ϵ-NFA be M and is given by

$$M = (Q_1 \cup Q_2 \cup \{q_0, f_0\}, \Sigma_1 \cup \Sigma_2, \delta, q_0, \{f_0\})$$

where δ is given by

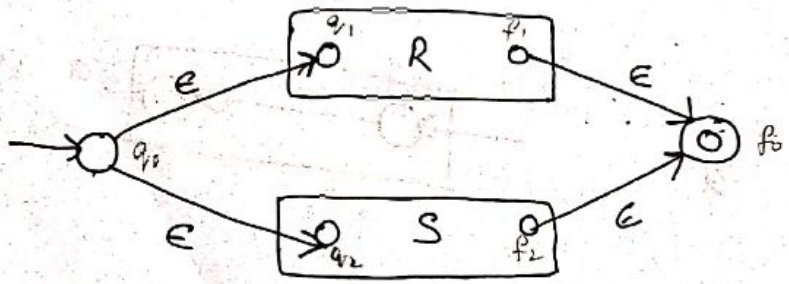
i) $\delta(q_0, \epsilon) = \{q_1, q_2\}$

ii) $\delta(q_0, a) = \delta_1(q, a)$ if $q \in Q_1 - \{f_1\}, a \in \Sigma_1 \cup \{\epsilon\}$

$= \delta_2(q, a)$ if $q \in Q_2 - \{f_2\}, a \in \Sigma_2 \cup \{\epsilon\}$

iii) $\delta_1(f_1, \epsilon) = \delta_2(f_2, \epsilon) = \{f_0\}$

The automaton for $R+S$ is as given below



Case 2:

Consider the expression RS and Assume the NFA for R as $M_1 = (Q_1, \Sigma_1, \delta_1, q_1, \{f_1\})$ and the NFA for the RE S as $M_2 = (Q_2, \Sigma_2, \delta_2, q_2, \{f_2\})$ and the final ϵ-NFA be M and is given by

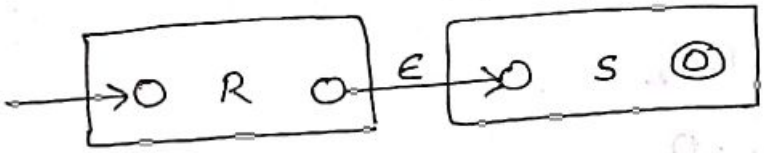
$M = (Q_1 \cup Q_2, \Sigma_1 \cup \Sigma_2, \delta, \{q_1\}, \{f_2\})$ where δ is given by

i) $\delta(q, a) = \delta_1(q, a)$ for $q \in Q_1 - \{f_1\}, a \in \Sigma_1 - \{\epsilon\}$

ii) $\delta(f_1, \epsilon) = \{q_2\}$

iii) $\delta(q, a) = \delta_2(q, a)$ for $q \in Q_2 - \{f_2\}, a \in \Sigma_2 - \{\epsilon\}$

The automaton for RS is given by



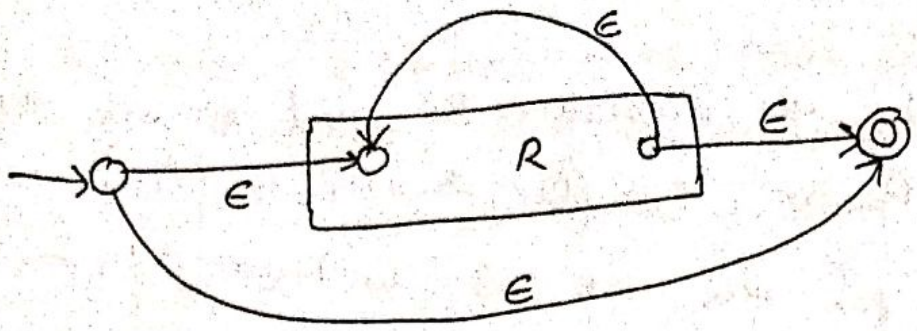
Case 3:

Consider the expression R^* and assume the NFA for R as $M_1 = (Q_1, \Sigma_1, \delta_1, q_1, \{f_1\})$ then the final ϵ-NFA $M = (Q \cup \{q_0, f_0\}, \Sigma, \delta, q_0, \{f_0\})$ where δ is given by

$\delta(q_0, \epsilon) = \delta(f_1, \epsilon) = \{q_1, f_0\}$

$\delta(q, a) = \delta_1(q, a)$ for $q \in Q_1 - \{f_1\}, a \in \Sigma_1 \cup \{\epsilon\}$

The automaton for R^* is given by



The above automaton allows to go either.

i) Directly from the start state to the accepting state a path labeled ϵ. ϵ also belongs to R^*

ii) To the start state of the automaton for R, through that we can accept one or more times the R.

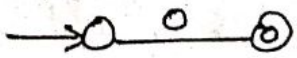
Q Convert the RE 01^* to the ϵ-NFA

$r = r_1 r_2$

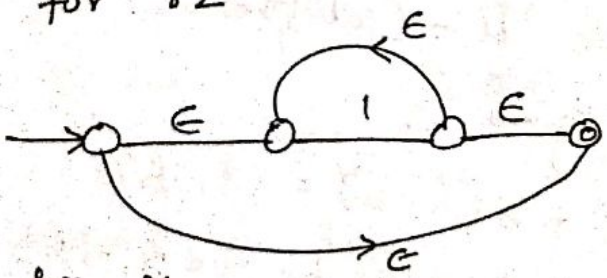
$r_1 = 0$

$r_2 = 1^*$

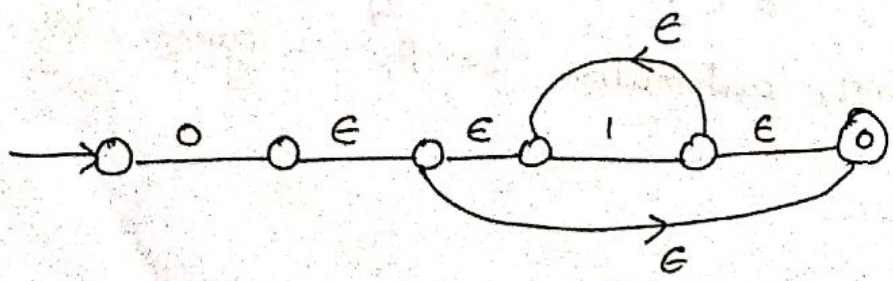
ϵ-NFA for r_1



ϵ-NFA for r_2



ϵ-NFA for r



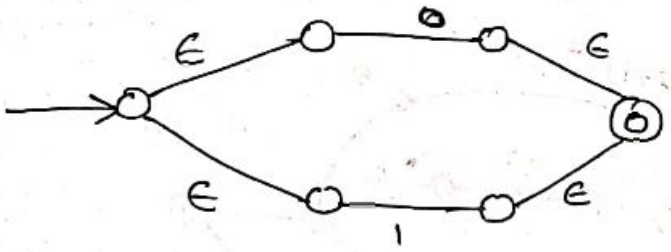
Q Convert the RE $(0+1)01$ to the ϵ-NFA.

$r = r_1 * r_2$

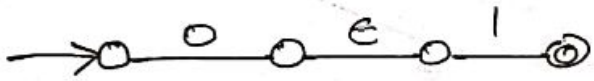
$r_1 = 0+1$

$r_2 = 01$

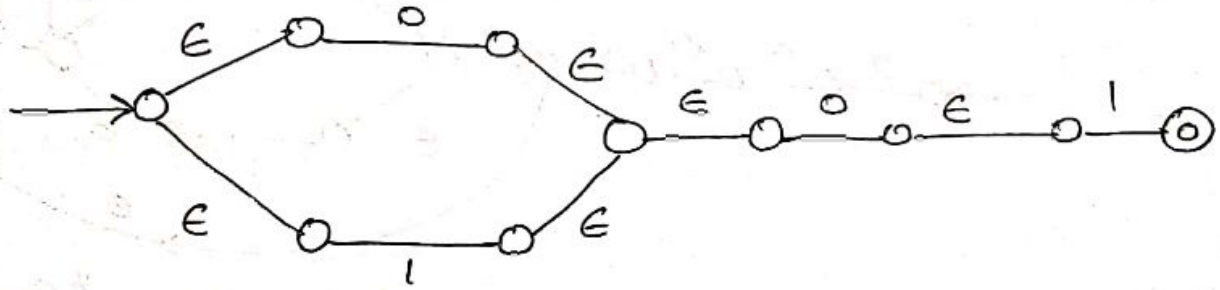
ϵ-NFA for r_1



ϵ-NFA for r_2



ϵ-NFA for r is



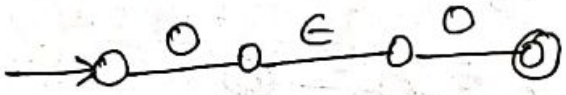
Q Convert the RE $00(0+1)^*$ to the ϵ-NFA

$r = r_1 r_2$

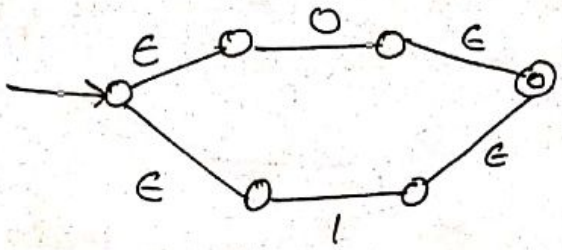
$r_1 = 00$

$r_2 = (0+1)^*$

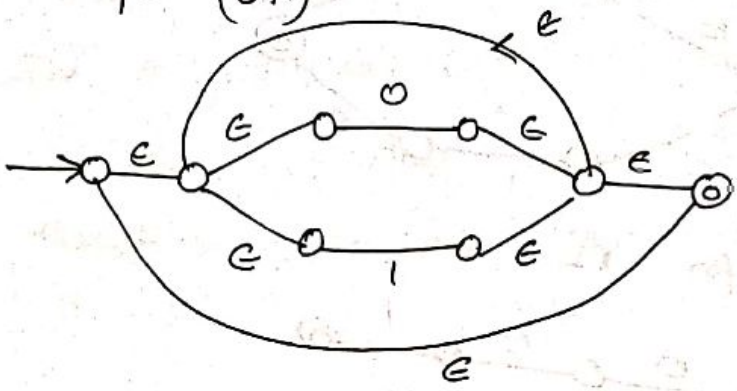
ϵ-NFA for r_1 is



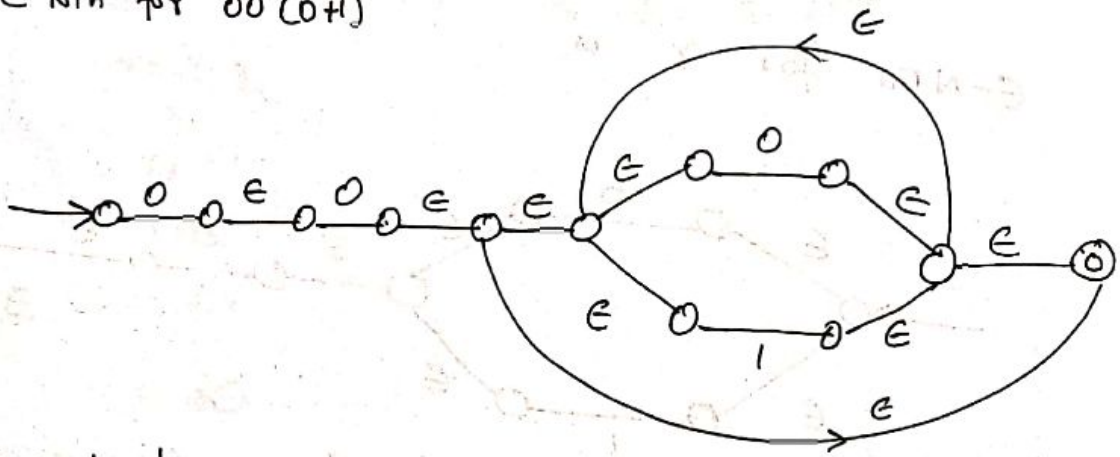
ϵ- NFA for 0+1



ϵ-NFA for $(0+1)^*$



ϵ-NFA for $00(0+1)^*$



Q Construct an ϵ-NFA equivalent $(0+1)^*(00+11)$

$r = r_1 r_2$

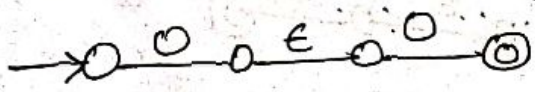
$r_1 = (0+1)^*$

$r_2 = \underset{r_3}{00} + \underset{r_4}{11}$

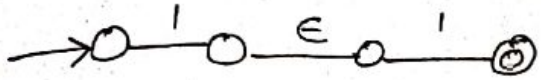
$r_3 = 00$

$r_4 = 11$

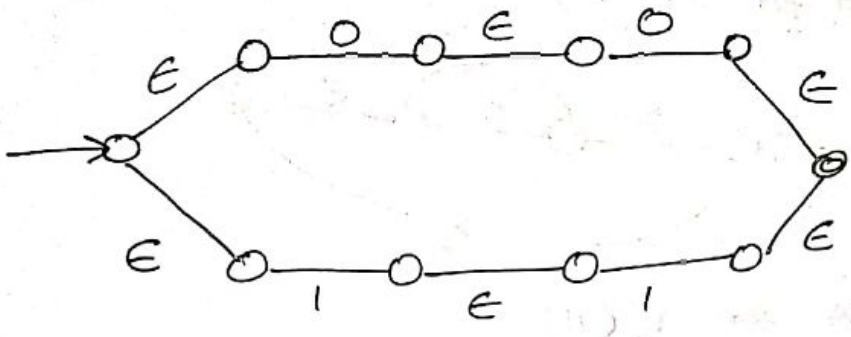
ϵ- NFA for r_3



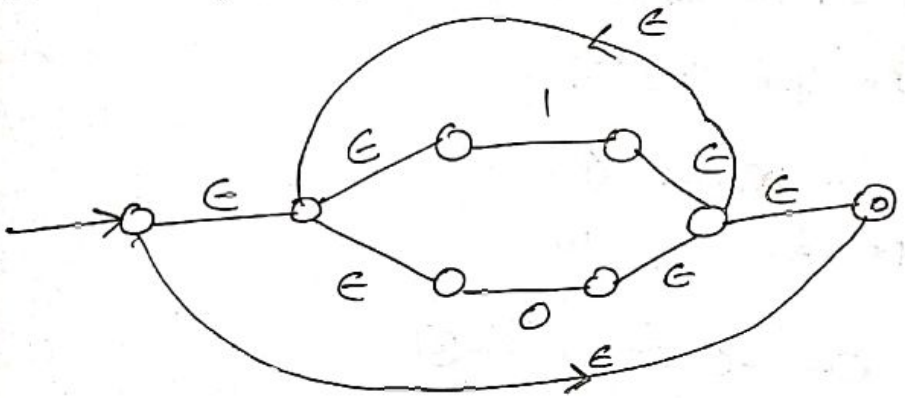
∈-NFA for r_4



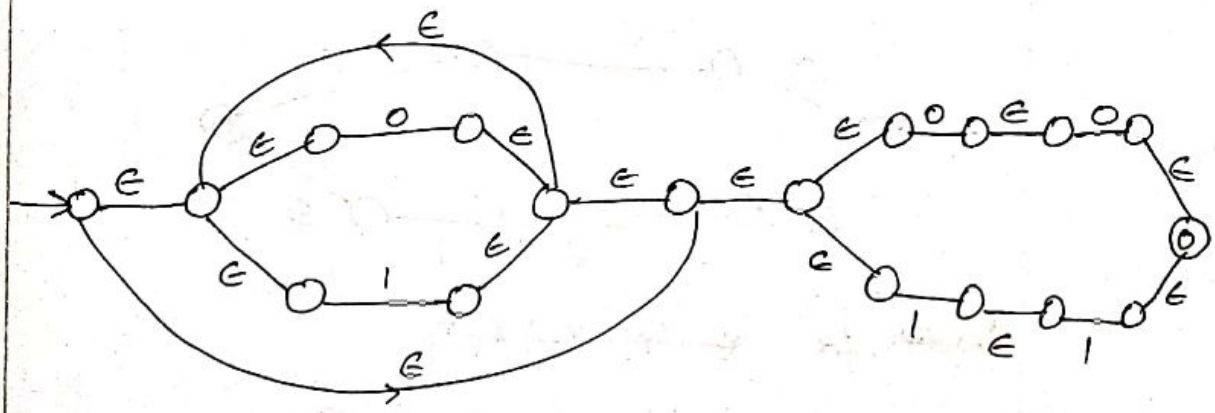
∈-NFA for r_2



∈-NFA for r_1



∈-NFA for r



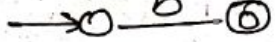
Q Construct ∈-NFA for RE $10 + (0+11)\,0^*1$

$r = r_1 + (r_2 + r_3)\, r_4\, r_5$

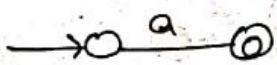
Q Convert the RE $a.(a+b)^*b.b$ to ϵ-NFA.

$r = r_1 r_2 r_3 r_4$.

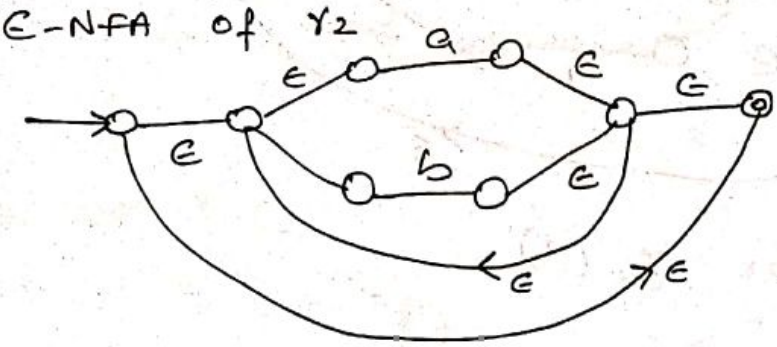
ϵ-NFA of r_3 & r_4 (b).



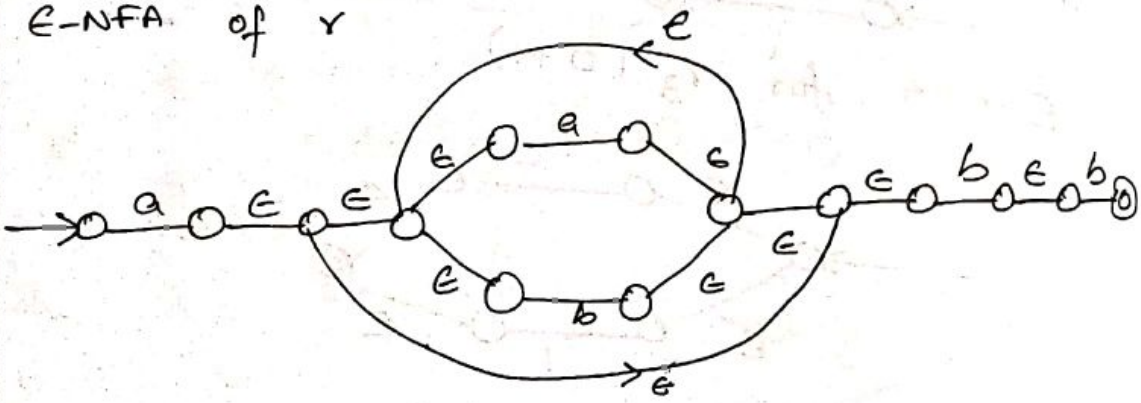
ϵ-NFA of r_1 (a)



ϵ-NFA of r_2



ϵ-NFA of r



Equivalence of DFA and Regular Expressions

An equivalent Regular expression are going to build from the set of strings which is labeled on certain paths of the DFA. In DFA paths are allowed to pass only a limited subset of states

Theorem:-

If $L = L(A)$ for some DFA A, then there is a

Regular expression R such that $L = L(R)$.

Proof :-

Suppose that the DFA A's states are $\{1, 2, 3 \ldots n\}$ for some integer n

Assume that R_{ij}^{k} be the name of the Regular expression whose language is the set of strings 'w' which is labeled on certain paths from state i to state j in DFA 'A', and the path has no intermediate node whose number is greater than k.

To construct the expression $R_{ij}^{(k)}$ 'k' should be start from $k=0$ to $k=n$ where 'n' is the total number of states in the given DFA 'A'. When $k=n$ then there is no restriction on the intermediate paths and no intermediate states are greater than k-value.

BASIS :-

The basis is when $k=0$ the restrictions on paths is that the path must have no intermediate states at all. There are only two kind of paths that meet such a criteria

i) An arc from node i to node j

ii) A path of length 0 that consists of only some node i

If $i \neq j$ then only case (i) is possible. We must examine DFA A and find those input symbols 'a' such that there is a transition from state i to state j on symbol 'a'.

a) If there is no such symbol a, then $R_{ij}^{0} = \phi$

b) If there is exactly one such symbol a, then $R_{ij}^{0} = a$

c) If there are symbols $a_1, a_2, \ldots a_k$ that label arcs from state i to state j then $R_{ij}^{0} = a_1 + a_2 + \cdots + a_k$

If $i = j$, then the legal paths are path of length 0 and all loops from i to itself.

The path of length 0 is represented by Regular expression ϵ Since the path has no symbols on it.

If the path contains any symbol 'a' then the Regular expression is $\epsilon + a$

If the path contain more than one symbol $a_1, a_2 \ldots a_k$ then the Regular expression is $\epsilon + a_1 + a_2 \ldots + a_k$

INDUCTION:-

The induction part is proved for $k \geq 1$. Suppose there is a path from state i to state j that goes through no states higher than k. There are 2 possible cases

i) The path does not go through state k at all. In this case label of path is the language is given by R_{ij}^{k-1}

ii) The path goes through state k atleast once. Then we can break the path into several pieces as below

The first part goes from state i to state k without passing through k.
The last part goes from state k to state j without passing through k
The middle part goes from state k to k itself without passing through k.

Note:-
If the path goes through k only once, then there is no middle part just a path from state i to k and state k to j.

The set of labels of all paths for case (ii) can be represented by the regular expression as $R_{ik}^{k-1}\left(R_{kk}^{k-1}\right)^{*} R_{kj}^{k-1}$

When we combine the expressions for the paths of the two cases above we get the regular expression as

$$R_{ij}^{k} = R_{ij}^{k-1} + R_{ik}^{k-1}\cdot\left(R_{kk}^{k-1}\right)^{*} R_{kj}^{k-1}$$

for the labels of all paths from state i to state j go through no state higher than k.

∴ By induction we prove that the Regular expression R_{ij}^{n} can be constructed for all values of i, j and n

Note:-
The RE for the language of the automaton is the sum of all expression R_{1j}^{n} such that state 1 is the start state and state 'j' is the accepting states and 'n' is the total number of states.

Converting DFA to Regular Expression

Basic formula on Regular Expression:-

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

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

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

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

$0+1^*0 = 1^*0$

$0+01^* = 01^*$

$\epsilon+00^* = 0^*$

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

$\phi 0 = \phi$; $\phi+0 = 0$

$\phi^* = \epsilon$

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

$\phi+1 = 1+\phi = 1$

$\epsilon 0 = 0$

$\epsilon^* = \epsilon$

$0+0 = 0$; $1+1 = 1$

$RR^* = R^*R = R^+$

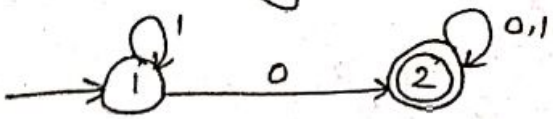
$\epsilon+RR^* = R^*$

$(01)^*0 = 0(10)^*$

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

$1+1^* = 1^*$

Q Convert the following DFA into Regular expression



Find $R_{ij}^{(k)}$ for all values of i, j and k

No: of states in the above DFA is 2.

K values varies as 0, 1 and 2.

When $k = 0$

$$R_{ij}^{(0)} = \begin{cases} \epsilon + a_1 + a_2 + \cdots + a_k & \text{if } i = j \\ a_1 + a_2 + \cdots + a_k & \text{if } i \neq j \end{cases}$$

$\therefore R_{11}^{(0)} = \epsilon + 1$

$R_{12}^{(0)} = 0$

$R_{21}^{(0)} = \phi$

$R_{22}^{(0)} = \epsilon + 0 + 1$

When $k = 1$

$$R_{ij}^{k} = R_{ij}^{k-1} + R_{ik}^{k-1}\left(R_{kk}^{k-1}\right)^{*} R_{kj}^{k-1}$$

$$\therefore R_{11}^{1} = R_{11}^{0} + R_{11}^{0}\left(R_{11}^{0}\right)^{*} R_{11}^{0}$$

$$= (\epsilon+1) + (\epsilon+1)(\epsilon+1)^{*}(\epsilon+1)$$

$$= (\epsilon+1) + (\epsilon+1)\,1^{*}(\epsilon+1) \quad [\because 1^{*} = \epsilon, 1, 11 \cdots]$$

$$= (\epsilon+1) + 1^{*}$$

$$= 1^{*}$$

$$R_{12}^{1} = R_{12}^{0} + R_{11}^{0}\left(R_{11}^{0}\right)^{*} R_{12}^{0}$$

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

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

$= 0 + 1^*0$

$= 1^*0$

$R_{21}^{1} = R_{21}^{0} + R_{21}^{0}(R_{11}^{0})^* R_{11}^{0}$

$= \phi + \phi(\epsilon+1)^*(\epsilon+1)$

$= \phi + \phi$ $(\because \phi R = R)$

$= \phi$

$R_{22}^{1} = R_{22}^{0} + R_{21}^{0}(R_{11}^{0})^* R_{12}^{0}$

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

$= (\epsilon+0+1) + \phi$

$= \epsilon+0+1$

when $k = 2$

$R_{ij}^{2} = R_{ij}^{1} + R_{12}^{1}(R_{22}^{1})^* R_{2j}^{1}$

$R_{11}^{2} = R_{11}^{1} + R_{12}^{1}(R_{22}^{1})^* R_{21}^{1}$

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

$= 1^* + \phi$ $\because (\phi R = \phi)$

$= 1^*$ $\because (\phi + R = R)$

$R_{12}^{2} = R_{12}^{1} + R_{12}^{1}(R_{22}^{1})^* R_{22}^{1}$

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

$= 1^*0 + 1^*0(0+1)^*(\epsilon+0+1)$ $[\because (\epsilon+1)^* = 1^*]$

$= 1^*0 + 1^*0(0+1)^*$ $[\because 1^*(\epsilon+1) = 1^*]$

$= 1^*0(0+1)^*$ $[\because 0 + 01^* = 01^*]$

$$R_{22}^{(2)} = R_{22}^{1} + R_{22}^{1}\left(R_{22}^{1}\right)^{*} R_{22}^{1}$$

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

$$= (\epsilon+0+1) + (\epsilon+0+1)(0+1)^{*}(\epsilon+0+1) \quad \because (\epsilon+1)^{*} = 1^{*}$$

$$= (\epsilon+0+1) + (\epsilon+0+1)(0+1)^{*} \quad \because 1^{*}(\epsilon+1) = 1^{*}$$

$$= (\epsilon+0+1) + (0+1)^{*} \quad \because (\epsilon+1)1^{*} = 1^{*}$$

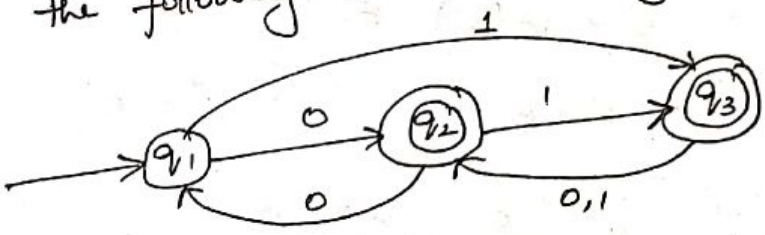
$$= (0+1)^{*} \quad \because (\epsilon+1) + 1^{*} = 1^{*}$$

The final Regular expression equivalent to the given DFA is constructed by taking the union of all the expression where the first state is the start state and the second state is accepting.

In this DFA the final Regular expression is R_{12}^{2}

So the final RE is $1^{*}0(0+1)^{*}$

Q Convert the following DFA to a Regular Expression



The above DFA contain two final states. The final Regular expression consists of the union of the two final states. The Regular expression is given by

$$R_{12}^{(3)} + R_{13}^{(3)}$$

when $k=0$

$$R_{ij}^{0} = \begin{cases} \epsilon + a_1 + a_2 + \cdots + a_k & \text{if } i=j \\ a_1 + a_2 + \cdots + a_k & \text{if } i \neq j \end{cases}$$

$R_{11}^{(0)} = \epsilon$

$R_{12}^{(0)} = 0$

$R_{13}^{(0)} = 1$

$R_{21}^{(0)} = 0$

$R_{22}^{(0)} = \epsilon$

$R_{23}^{(0)} = 1$

$R_{31}^{(0)} = \phi$

$R_{32}^{(0)} = 0+1$

$R_{33}^{(0)} = \epsilon$

when $K = 1$

$$R_{ij}^{K} = R_{ij}^{K-1} + R_{iK}^{K-1} \left(R_{KK}^{K-1}\right)^{*} R_{kj}^{K-1}$$

$$R_{11}^{(1)} = R_{11}^{0} + R_{11}^{0} \left(R_{11}^{0}\right)^{*} R_{11}^{0}$$

$$= \epsilon + \epsilon (\epsilon)^{*} \epsilon$$

$$= \epsilon + \epsilon \qquad \because \epsilon^{*} = \epsilon$$

$$= \epsilon$$

$$R_{12}^{(1)} = R_{12}^{0} + R_{11}^{0} \left(R_{11}^{0}\right)^{*} R_{12}^{0}$$

$$= 0 + \epsilon (\epsilon)^{*} 0$$

$$= 0 + 0 \qquad [\because \epsilon 0 = 0]$$

$$= 0$$

$$R_{13}^{1} = R_{13}^{0} + R_{11}^{0} \left(R_{11}^{0}\right)^{*} R_{13}^{0}$$

$$= 1 + \epsilon (\epsilon)^{*} 1$$

$= 1 + \epsilon 1$ $[\because \epsilon 1 = 1]$

$= 1 + 1$

$= \underline{1}$

$R_{21}^{(1)} = R_{21}^{0} + R_{21}^{0}\left(R_{11}^{0}\right)^{*} R_{11}^{0}$

$= 0 + 0(\epsilon)^{*}\epsilon$

$= 0 + 0\epsilon$

$= 0 + 0$

$= \underline{0}$

$R_{22}^{(1)} = R_{22}^{0} + R_{21}^{0}\left(R_{11}^{0}\right)^{*} R_{12}^{0}$

$= \epsilon + 0(\epsilon)^{*}0$ $[\because \epsilon^{*} = \epsilon \ \& \ \epsilon 0 = 0]$

$= \epsilon + 00$

$R_{23}^{(1)} = R_{23}^{0} + R_{21}^{0}\left(R_{11}^{0}\right)^{*} R_{13}^{0}$

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

$= 1 + 0\epsilon 1$

$= 1 + 01$

$R_{31}^{(1)} = R_{31}^{0} + R_{31}^{0}\left(R_{11}^{0}\right)^{*} R_{11}^{0}$

$= \phi + \phi(\epsilon)^{*}\epsilon$

$= \phi + \phi$

$= \underline{\phi}$

$R_{32}^{(1)} = R_{32}^{0} + R_{31}^{0}\left(R_{11}^{0}\right)^{*} R_{12}^{0}$

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

$= (0+1) + \phi 0$

$= (0+1) + \phi$

$= \underline{0+1}$

$$R_{33}^{(1)} = R_{33}^{0} + R_{31}^{0}\left(R_{11}^{0}\right)^{*} R_{13}^{0}$$

$$= \epsilon + \phi(\epsilon)^{*}1$$

$$= \epsilon + \phi 1$$

$$= \epsilon + \phi$$

$$= \epsilon$$

when $k = 2$

$$R_{ij}^{2} = R_{ij}^{1} + R_{iK}^{1}\left(R_{KK}^{1}\right)^{*} R_{Kj}^{1}$$

$$R_{11}^{2} = R_{11}^{1} + R_{12}^{1}\left(R_{22}^{1}\right)^{*} R_{21}^{1}$$

$$= \epsilon + 0\,(\epsilon + 00)^{*}\,0$$

$$= \epsilon + 0\,(00)^{*}\,0$$

$$= \epsilon + 0(00)^{*}0$$

$$= (00)^{*} \qquad (\because \epsilon + 00^{*} = 0^{*})$$

$$R_{12}^{2} = R_{12}^{1} + R_{12}^{1}\left(R_{22}^{1}\right)^{*} R_{22}^{1}$$

$$= 0 + 0(\epsilon + 00)^{*}(\epsilon + 00)$$

$$= 0 + 0(00)^{*}(\epsilon + 00) \qquad [\because (\epsilon + 1)^{*} = 1^{*}]$$

$$= 0 + 0(00)^{*} \qquad [\because (\epsilon + 1)1^{*} = 1^{*}]$$

$$= 0(00)^{*}$$

$$R_{13}^{2} = R_{13}^{1} + R_{12}^{1}\left(R_{22}^{1}\right)^{*} R_{23}^{1}$$

$$= 1 + 0(\epsilon + 00)^{*}(1 + 01)$$

$$= 1 + 0(00)^{*}(1 + 01) \qquad (\because \epsilon + 1^{*} = 1^{*})$$

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

$$= 1 + 0^{*}1 \qquad \because (00)^{*}(\epsilon + 0) = 0^{*}]$$

$$= 0^{*}1$$

$$R_{21}^{2} = R_{21}^{1} + R_{22}^{1}\left(R_{22}^{1}\right)^{*} R_{21}^{1}$$

$$= 0 + (\epsilon+00)(\epsilon+00)^{*}0 \qquad (\because (\epsilon+1)^{*} = 1^{*})$$

$$= 0 + (\epsilon+00)(00)^{*}0 \qquad (\because (\epsilon+1)1^{*} = 1^{*})$$

$$= 0 + (00)^{*}0$$

$$= 0(00)^{*}$$

$$R_{22}^{2} = R_{22}^{1} + R_{22}^{1}\left(R_{22}^{1}\right)^{*} R_{22}^{1}$$

$$= (\epsilon+00) + (\epsilon+00)(\epsilon+00)^{*}(\epsilon+00) \qquad (\because (\epsilon+1)^{*} = 1^{*})$$

$$= (\epsilon+00) + (\epsilon+00)(00)^{*}(\epsilon+00) \qquad (\because (\epsilon+1)1^{*} = 1^{*})$$

$$= (\epsilon+00) + 00^{*}(\epsilon+00) \qquad (\because 1(\epsilon+1) = 1^{*})$$

$$= (\epsilon+00) + 00^{*} \qquad (\because (\epsilon+1) + 1^{*} = 1^{*})$$

$$= 00^{*}$$

$$R_{23}^{2} = R_{23}^{1} + R_{22}^{1}\left(R_{22}^{1}\right)^{*} R_{23}^{1}$$

$$= (1+01) + (\epsilon+00)(\epsilon+00)^{*}(1+01)$$

$$= (1+01) + (\epsilon+00)(00)^{*}(1+01)$$

$$= (1+01) + (00)^{*}(1+01)$$

$$= \cancel{(1+01)(\epsilon+(00)^{*})}\ (1+01) + (00)^{*}(\epsilon+0)1$$

$$= \cancel{(1+01)(00)^{*}} \cdot (1+01)(0^{*})1 \qquad (\because (00)^{*}(\epsilon+0) = 0^{*})$$

$$= 0^{*}1$$

$$R_{31}^{2} = R_{31}^{1} + R_{32}^{1}\left(R_{22}^{1}\right)^{*} R_{21}^{1}$$

$$= \phi + (0+1)(\epsilon+00)^{*}0$$

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

$$R_{32}^{(2)} = R_{32}^{1} + R_{32}^{1}\left(R_{22}^{1}\right)^{*} R_{22}^{1}$$

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

$$= (0+1) + (0+1)(00)^{*}(\epsilon+00) \qquad ((\epsilon+00)^{*} = (00)^{*}$$

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

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

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

$$R_{33}^{(2)} = R_{33}^{1} + R_{32}^{1}\left(R_{22}^{1}\right)^{*} R_{23}^{1}$$

$$= \epsilon + (0+1)(\epsilon+00)^{*}(1+01)$$

$$= \epsilon + (0+1)(00)^{*}(1+01)$$

$$= \epsilon + (0+1)(00)^{*}(\epsilon+01)$$

$$= \epsilon + (0+1)(00)^{*}01$$

The regular expression is constructed by

$$r_{12}^{(3)} + r_{13}^{(3)}$$

$$r_{12}^{(3)} = r_{12}^{(2)} + r_{13}^{(2)}\left(r_{33}^{2}\right)^{*} r_{32}^{2}$$

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

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

$$r_{13}^{(3)} = r_{13}^{(2)} + r_{13}^{(2)}\left(r_{33}^{(2)}\right)^{*} r_{33}^{(2)}$$

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

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

$$\therefore r_{12}^{(3)} + r_{13}^{(3)} = 0^{*}1((0+1)0^{*}1)^{*} + 0(00)^{*} + 0^{*}1((0+1)0^{*}1)^{*}(0+1)(00)^{*}$$

Converting DFA to Regular Expression by state Elimination method

Steps to Convert DFA to RE :-

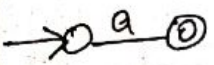
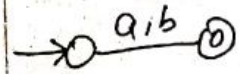
i) If the initial state of the given DFA is having any incoming edges then create a new initial state and make an ϵ move from the new initial state to the initial state of the DFA

ii) If the final state of the DFA is having any outgoing edges then create a new final state in the DFA and make an ϵ-move from the final state of the DFA to the new final state

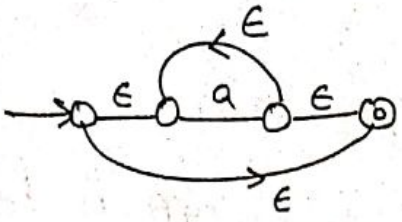
iii) There should be only one final state in the DFA, if more than one final state is present in the DFA then we should combine all the final states to the new final state through an ϵ-move.

iv) Eliminate all states in the DFA except the initial and final state and find out the regular expression

Basic RE and FA :-

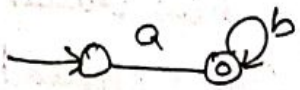
FA	RE
	ϕ
	a
	a+b

FA

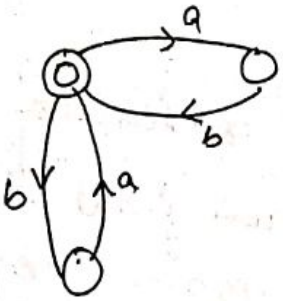


RE

a^*



ab^*

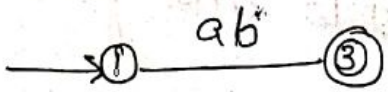


$(ab+ba)^*$

Q Construct RE from the following DFA

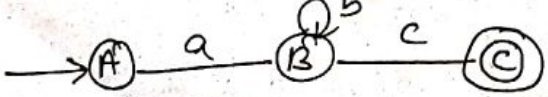
The above DFA, the initial state has no incoming edge and final state has no outgoing edge. So no need to add new initial & final state.

Eliminate state 2

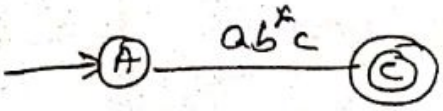


So the RE is $\underline{ab}$

Q Construct RE from the following DFA

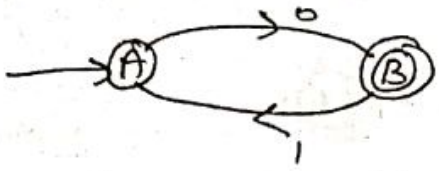


Eliminate state B

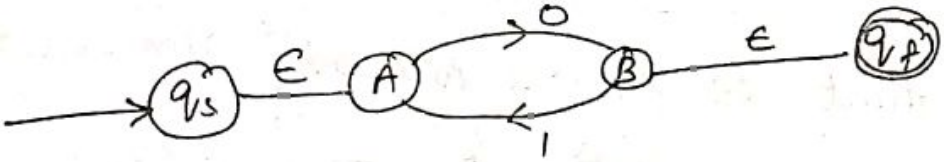


The RE is $\underline{ab^*c}$

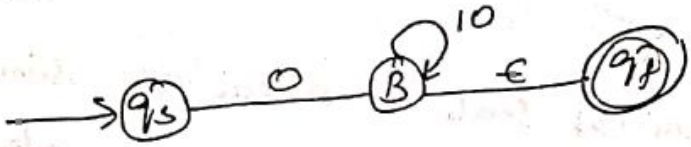
Q Construct RE from the following DFA



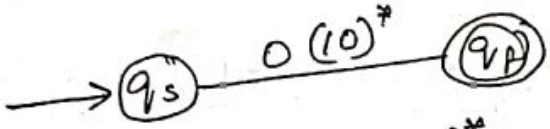
The above DFA, the initial state has incoming edge & final state have outgoing edge. So we need to create a new initial and final state to the DFA. Let it be q_s and q_f



Eliminate state A

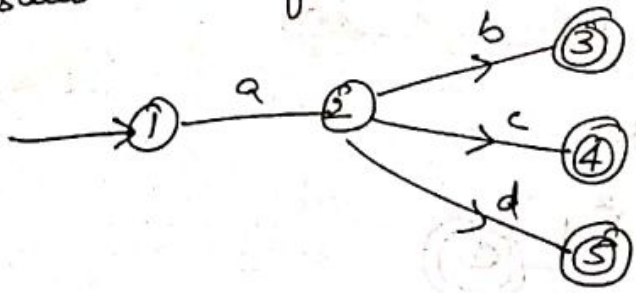


Eliminate state B

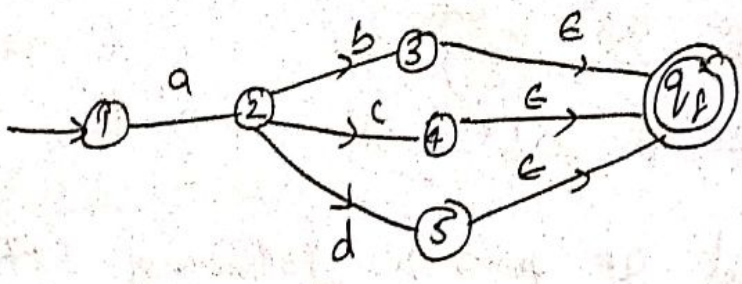


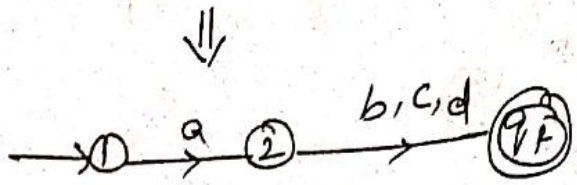
The RE is $\underline{0(10)^*}$

Q Construct RE for the following

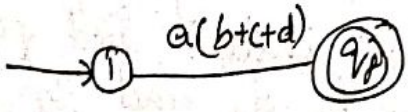


The above DFA has multiple final state so we need to add a new final state



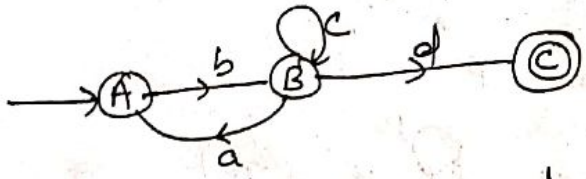


Eliminate 2

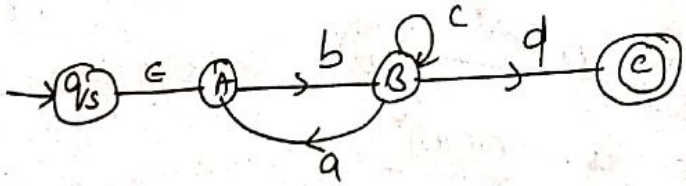


The RE is $a(b+c+d)$

Q. Construct RE for the following DFA.

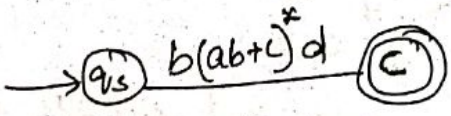


Adding new initial state to above DFA since the initial state is have an incoming edge.



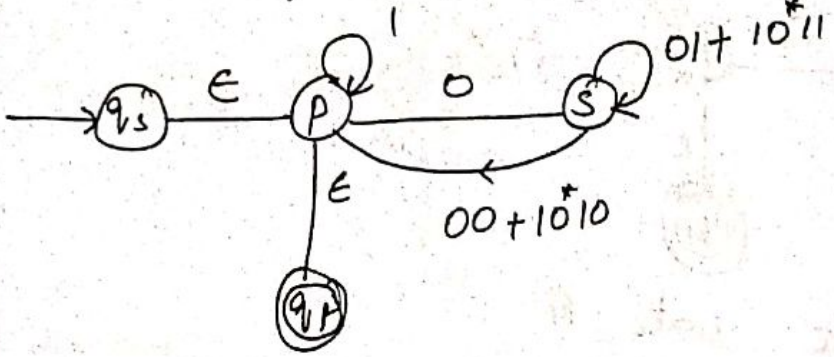
Eliminate A

Eliminate B

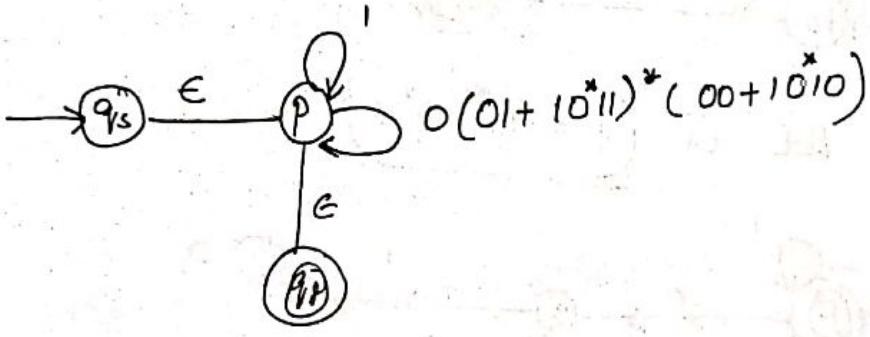


The RE is $b(ab+c)^*d$

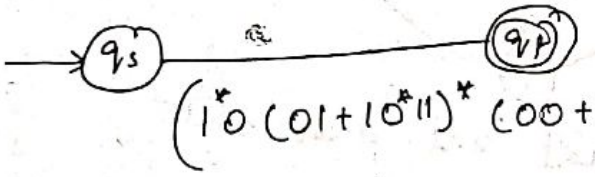
Eliminate q



Eliminate s



Eliminate P



The RE is $(1^*0(01+10^*11)^*(00+10^*10))^*$

Myhill Nerode Theorem

Myhill Nerode Theorem states the following 3 statements are equivalent

1. The set $L \subseteq \Sigma^*$ is accepted by some finite automata
2. L is the union of equivalence classes of right invariant equivalence relation of finite index.
3. Let equivalence relation R_L be defined by xR_Ly iff for all z in Σ^*, xz is in L exactly

when yz is in L then R_L is of finite index

proof

(1) $\Rightarrow$ (2)

Assume L is accepted by some DFA $M = (Q, \Sigma, \delta, q_0, F)$

Let R_m be the equivalence relation $x R_m y$ where x and y are the strings

$x R_m y$ iff $\delta(q_0, x) = \delta(q_0, y)$

And R_m is right invariant relation, so for any z

if $\delta(q_0, x) = \delta(q_0, y)$ then

$$\begin{aligned}\delta(q_0, xz) &= \delta(\delta(q_0, x), z) \\ &= \delta(\delta(q_0, y), z) \\ &= \delta(q_0, yz)\end{aligned}$$

$\therefore \delta(q_0, xz) = \delta(q_0, yz)$ so R_m is right invariant

The index of R_m is finite since the index is almost the number of states in Q

L is the union of equivalence classes that includes a string x such that $\delta(q_0, x)$ is in F. i.e equivalence class corresponds to final states.

(2) $\Rightarrow$ (3)

Let E be the equivalence relation satisfying statement (2) and which is a refinement of R_L

i.e E is a refinement of R_L, i.e every equivalence class of E is entirely contained in some equivalence classes of R_L

The index of R_L cannot be greater than the index of E so it is finite.

Assume xEy, since E is right invariant, for each z in Σ^*, $xzEyz$ and yz is in L iff xz is in L.

Then xR_Ly and hence the equivalence class of x in E is contained in the equivalence class of x in R_L

$\therefore$ Each equivalent class of E is contained with some equivalent of R_L

(3) $\Rightarrow$ (1)

First we have to show that R_L is right invariant

Suppose xR_Ly and let w be the string in Σ^*

we have to prove that xwR_Lyw.

ie for every z, xwz is in L exactly when ywz is in L

Since xR_Ly; ie $xwz \in L \Leftrightarrow ywz \in L \quad \forall w \& \forall z$

$\therefore \delta(q_0, xwz) = \delta(q_0, ywz)$

$\therefore xwz\ R_L\ ywz$

Hence R_L is right invariant.

Now consider FSA $M' = (K', \Sigma, \delta', q_0', F')$ as follows.

Let K' be the finite set of equivalence class of R_L and $[x]$ be the element of K' containing the string x where $x \in \Sigma^*$

$|K'|$ = index of R_L

Let $q_0' = [\epsilon]$ is the start state is the equivalence class to which empty string belongs

δ' can be defined as

$$\delta'([x], a) = [xa].$$

This definition is consistent. Since R_L is right invariant. If we choose 'y' instead of x from the equivalence class [x], then we have obtained

$$\delta'([x], a) = [ya]$$

or

$$\delta'([y], a) = [ya]$$ since $x R_L y$ i.e x and y belongs to same class

$$\therefore \delta'([x], a) = \delta'([y], a)$$

$$\Rightarrow [xa] = [ya]$$

F' can be given by

$$F' = \{[x] / x \in L\}$$

The final state consists of the set of equivalence class which belongs to L.

So the FA $M' = (K', \Sigma, \delta', q_0', F')$ accepts L,

since $\delta'(q_0', x) = [x]$ & 'x' is in $L(M')$ iff if $[x]$ is in F'

Hence proved.

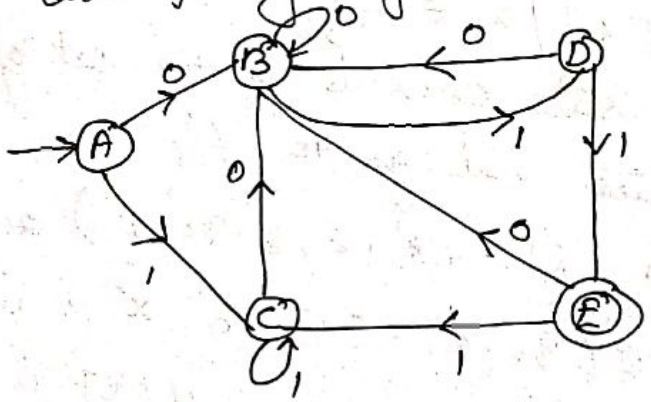
Minimal state FA computation

Minimization of DFA by table filling Method (Myhill Nerode theorem)

Steps for Minimizing DFA

1) Draw a table for all pairs of state (P,Q)
2) Mark all pairs where $P \in F$ and $Q \notin F$
3) If there are any unmarked pair (P,Q) such that $[\delta(P,x), \delta(Q,x)]$ is marked then mark [P,Q] where 'x' is any input symbol in Σ.
   Repeat this until no more markings can be made
4) Combine all the unmarked pairs and make them a single state in the minimized DFA

Q Compute Minimal DFA for the following given DFA using table filling algorithm.

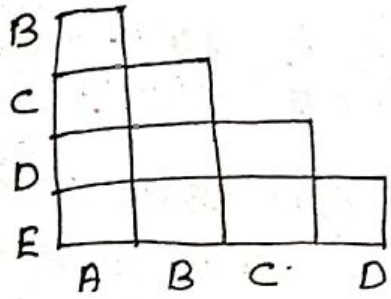


The table for the above DFA can be drawn as below

	A	B	C	D	E
A	AA	AB	AC	AD	AE
B	BA	BB	BC	BD	BE
C	CA	CB	CC	CD	CE
D	DA	DB	DC	DD	DE
E	EA	EB	EC	ED	EE

We are omitting the ** diagonly right half of the table because the pair present in right side is also present in left side

To avoid duplication omit the right half. The pairs which have multiple occurrences in the table are removed and the table now is reduced as follows

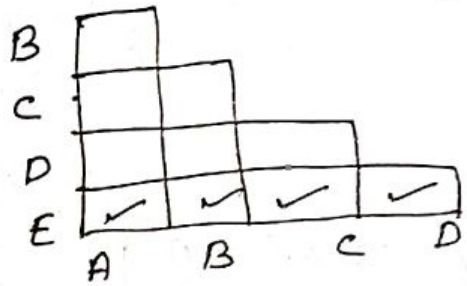


The final state in the given DFA is E. So mask all the cells in which the state E occurs

In (A,E) - A ∉ F
E ∈ F

In (B,E) - B ∉ F
E ∈ F

⋮



Now consider the transition table for the given DFA

	0	1
→A	B	C
B	B	D
C	B	C
D	B	E*
E*	B	C

In the table we have to make all unmarked pairs marked. For that check each pair one by one

Consider pair (A,B) = δ(A,0) = B δ(A,1) = C
δ(B,0) = B δ(B,1) = D

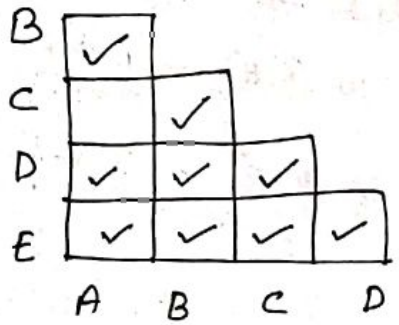
The pair (B,B) is not in table and the pair (C,D) is not yet marked in table so we cannot mark pair (A,B) now

Consider pair (A,C) = $\delta(A,0) = B$ $\delta(A,1) = C$
$\delta(C,0) = B$ $\delta(C,1) = C$

Consider pair (B,C) = $\delta(B,0) = B$ $\delta(B,1) = D$
$\delta(C,0) = B$ $\delta(C,1) = C$

Consider pair (A,D) = $\delta(A,0) = B$ $\delta(A,1) = C$
$\delta(D,0) = B$ $\delta(D,1) = E$

In the pair (A,D): $\delta((A,D),1) = (C,E)$ which is marked so we are marking the pair (A,D) in the table



Consider (B,D) ✓ = $\delta(B,0) = B$ $\delta(B,1) = D$
$\delta(D,0) = B$ $\delta(D,1) = E$

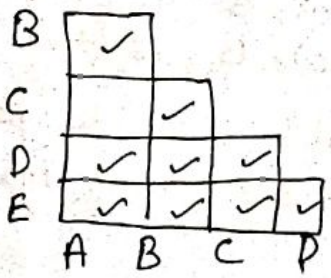
Consider (C,D) ✓ = $\delta(C,0) = B$ $\delta(C,1) = C$
$\delta(D,0) = B$ $\delta(D,1) = E$

Now again check (A,B) = (B,B) or (C,D) Now (C,D) pair is marked so mark (A,B) pair

For (A,C) we have (B,B) and (C,C) No marking

For (B,C) we have (B,B) and (C,D) (C,D) is marked so mark (B,C) pair in table.

So the final table we got is

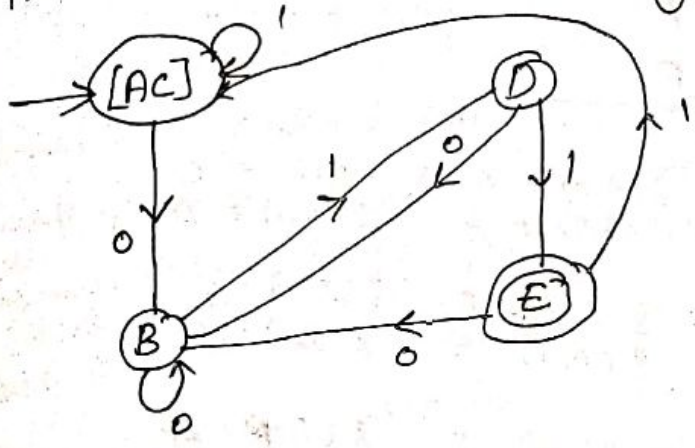


The only Unmarked pair in the table is (A,C) so group that pair in the minimal DFA.

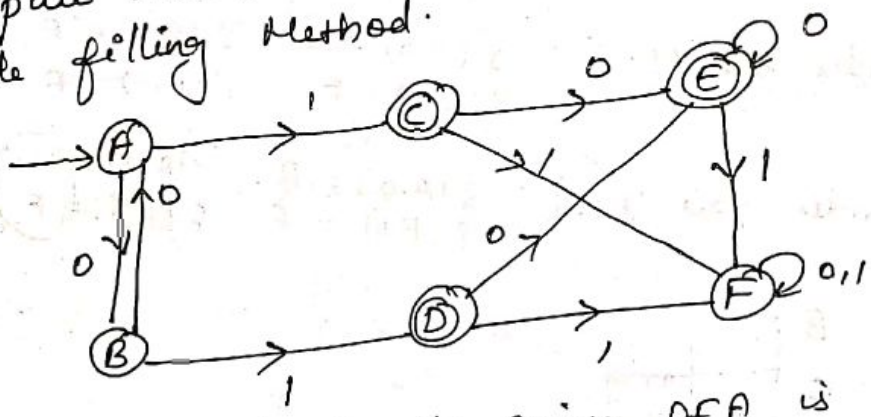
The states for the minimal DFA are

→(AC), B, D, E*

The minimal DFA is as given below



Q Compute Minimal DFA for the following DFA using table filling Method.



The final state in the given DFA is C, D and E

B				
C	X	X		
D	X	X		
E	X	X	X	X
F			X	X	X
	A	B	C	D	E

(C,E) } is not marked
(C,D) } because each of
(D,E) } this pair are equivalent ie in each pairs both states are final

ie C ∈ F & C ∈ F
E ∈ F D ∈ F

E ∈ F
D ∈ F

The STT of the DFA is given by

	0	1
→A	B	C*
B	A	D*
C*	E*	F
D*	E*	F
E*	E*	F
F	F	F

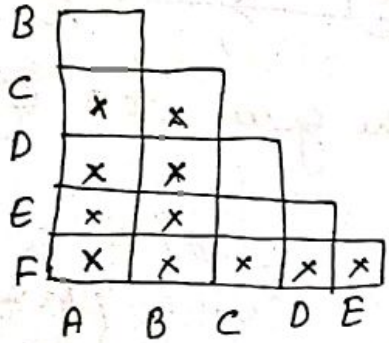
for pair (A,B) = $\delta(A,0) = B$ $\delta(A,1) = C$
$\delta(B,0) = A$ $\delta(B,1) = D$

Consider pair (C,D) = $\delta(C,0) = E$ $\delta(C,1) = F$
$\delta(D,0) = E$ $\delta(D,1) = F$

Consider pair (C,E) = $\delta(C,0) = E$ $\delta(E,1) = F$
$\delta(E,0) = E$ $\delta(C,1) = F$

Consider pair (D,E) = $\delta(D,0) = E$ $\delta(D,1) = F$
$\delta(E,0) = E$ $\delta(E,1) = F$

Consider pair (A,F) ✓ = $\delta(A,0) = B$ $\delta(A,1) = C$
$\delta(F,0) = F$ $\delta(F,1) = F$



Consider pair (B,F) ✓ = $\delta(B,0) = A$ $\delta(B,1) = D$
$\delta(F,0) = F$ $\delta(F,1) = F$

Now check (A,B) the δ on Σ is (A,B) & (C,D) - no marking
(C,D) = (E,E) & (F,F) - no marking
(C,E) = (E,E) & (F,F) - no marking
(D,E) = (E,E) & (F,F) - no marking

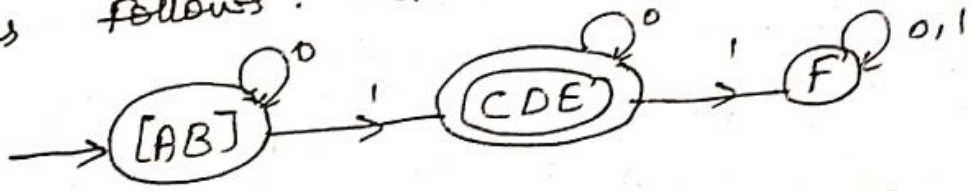
So the final table looks like

	A	B	C	D	E
B					
C	x	x			
D	x	x			
E	x	x			
F	x	x	x	x	x

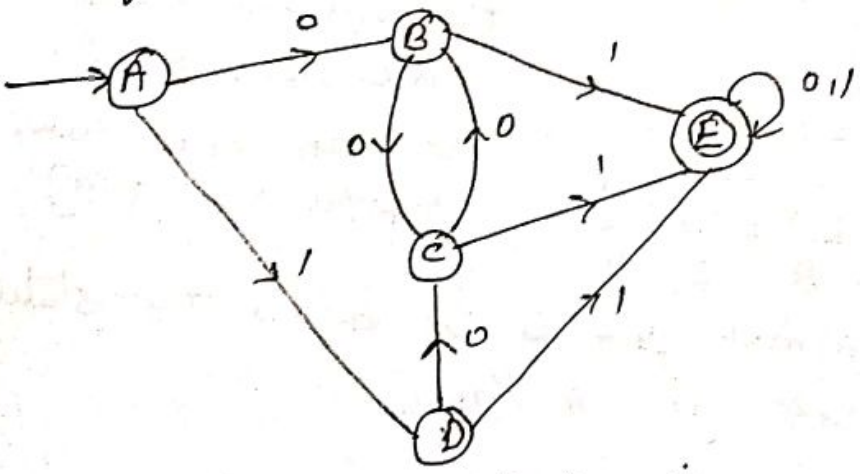
The unmarked pairs are
(A,B)
(C,D)
(C,E)
(D,E)

Group all the above unmarked pair

Consider unmarked pair (C,D), (C,E) & (D,E) we can group them together since the δ on input '0' and '1' gives the same pair (E,E) and (F,F) where E is non final and F is final state
We can group them as [CDE]. The minimal DFA is as follows. and the states are [AB], [CDE] & F



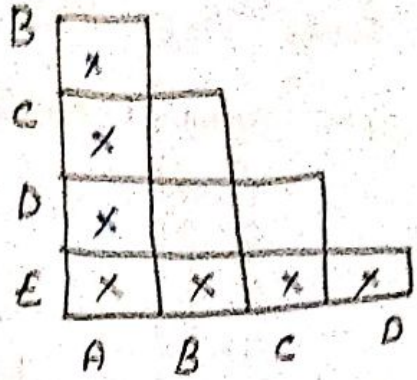
Q Compute minimal DFA for the following DFA using table filling Algorithm



The stt of the DFA is

	0	1
→ A	B	D
B	C	E*
C	B	E*
D	C	E*
E*	E*	E*

Since E is the final state, mark all cell of E



Consider (A,B) : $\delta(A,0) = B$ $\delta(A,1) = D, E$
$\delta(B,0) = C$ $\delta(B,1) = D, E$

(A,C) : $\delta(A,0) = B$ $\delta(A,1) = D, E$
$\delta(C,0) = B$ $\delta(C,1) = D, E$

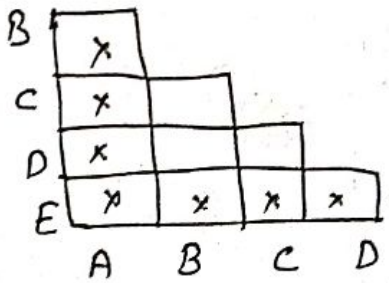
(B,C) = $\delta(B,0) = C$ $\delta(B,1) = E$
$\delta(C,0) = B$ $\delta(C,1) = E$

(A,D) = $\delta(A,0) = B$ $\delta(A,1) = D, E$
$\delta(D,0) = C$ $\delta(D,1) = D, E$

(B,D) = $\delta(B,0) = C$ $\delta(B,1) = E$
$\delta(D,0) = C$ $\delta(D,1) = E$

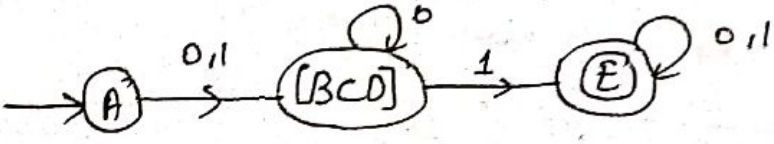
(C,D) = $\delta(C,0) = B$ $\delta(C,1) = E$
$\delta(D,0) = C$ $\delta(D,1) = E$

So the final table is



The unmarked pairs are (B,C), (B,D) & (C,D)
we can group them together as [BCD]

The minimal DFA is as below. The states of Minimal DFA is A, [BCD], E



Note:-
Two states in minimal DFA is equivalent or we can combine 2 states only if $\delta(P,w) \in F \Rightarrow \delta(q,w) \in F$ and $\delta(P,w) \notin F \Rightarrow \delta(q,w) \notin F$. Then we can combine P and q to a single state [Pq] in minimal DFA

Minimization of DFA by equivalent method (partition method)

steps for minimize a DFA using partition method is as follo

1) First eliminate any state in DFA that cannot be reached from the start state.

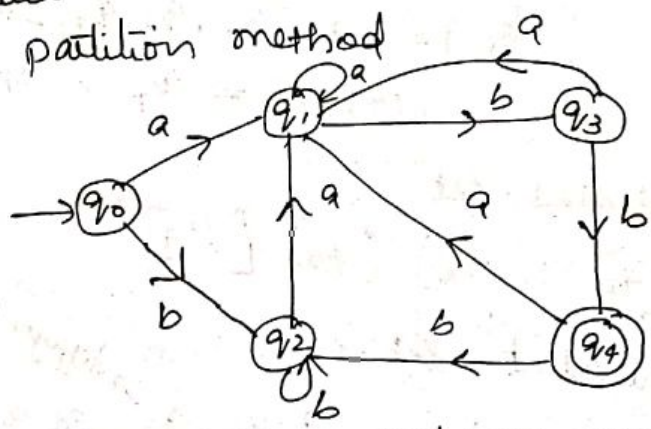
2) Then partition the remaining states into blocks, so that all states in the same block are equivalent and no pair of states from different block are equivalent

Generally the states are divided into
Equivalent
Distinguishable

→ States p and q are said to be equivalent if $\delta(p,w) \in F \Rightarrow \delta(q,w) \in F$ and $\delta(p,w) \notin F \Rightarrow \delta(q,w) \notin F$.
If the above two conditions satisfies then we can say the state p and q are equivalent.

→ States p and q are distinguishable if
$\delta(p,w) \in F \Rightarrow \delta(q,w) \notin F$ for any input w in Σ

Q Construct minimal DFA for the following DFA using partition method



i) Remove the states which are not reachable from the initial state
In the given DFA all states are reachable from the initial state.

ii) Construct state transition table of the DFA by excluding the states which are not reachable from initial state

	a	b
$\rightarrow q_0$	q_1	q_2
q_1	q_1	q_3
q_2	q_1	q_2
q_3	q_1	q_4^*
q_4^*	q_1	q_2

iii) 0 equivalent sets – Seperate nonfinal & final states

$[q_0\ q_1\ q_2\ q_3]\ [q_4]$

iv) 1 equivalent set - take previous equivalent and see whether it is 1 equivalent or not

take (q_0, q_1) state from 0 equivalent and for input a and b the transition is as follows

$(q_0, a) = q_1 \quad (q_0, b) = q_2$
$(q_1, a) = q_1 \quad (q_1, b) = q_3$

(q_1, q_2, q_3) are in same group of 0 equivalent so (q_0, q_1) state do not split .. so it is 1 equivalent

Similarly $q_0, q_2 \Rightarrow q_1, q_2$ = same group
$q_0\ q_3 \Rightarrow q_1, q_2, q_4$ = different group.
$\delta(q_3, b) \Rightarrow q_4$ so split q_3.

1 equivalent set
$[q_0\ q_1\ q_2]\ [q_3]\ [q_4]$

v) 2 equivalent set.

for $(q_0, q_1) \Rightarrow q_1, q_2, q_3$ → different group so q_1 is splitted

$[q_0 \quad [q_1]\ [q_3]\ [q_4]$

for $(q_0, q_2) \Rightarrow q_1, q_2$ - same group

So 2 equivalent set is

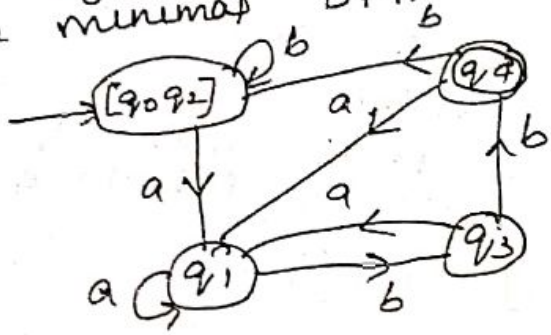
$[q_0\ q_2]\ [q_1]\ [q_3]\ [q_4]$

iii) 3 - equivalent

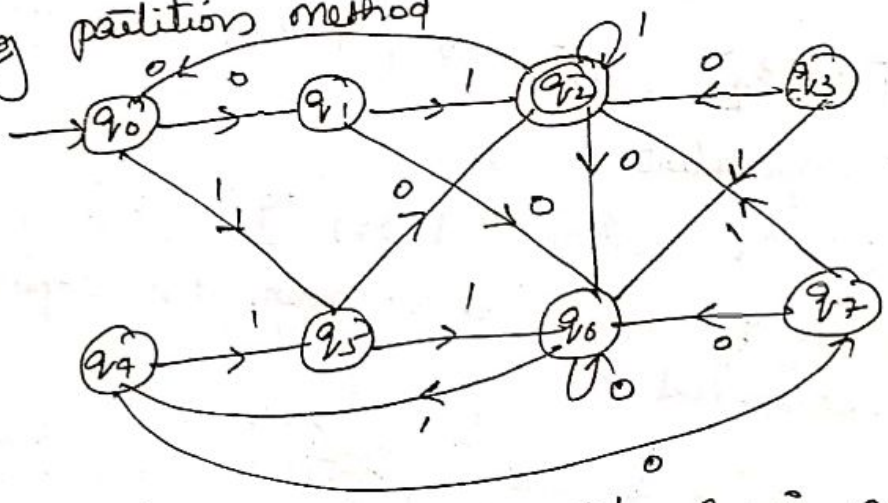
for $(q_0, q_2) \Rightarrow q_1, q_2$ same group

$[q_0\ q_2]\ [q_1]\ [q_3]\ [q_4]$

Since last two equivalence have same state for the group. we can stop here and can draw the minimal DFA



Q Construct minimal DFA from the following given DFA using partition method



In the above DFA the state q_3 is not reachable from the start state q_0. So eliminate the state q_3 and construct the state transition table

0 - equivalent.

$[q_0\ q_1\ q_4\ q_5\ q_6\ q_7]\ [q_2]$

1- equivalent

	0	1
q_0	q_1	q_5
q_1	q_6	q_2^*
q_2^*	q_0	q_2^*
q_4	q_7	q_5
q_5	q_2^*	q_6
q_6	q_6	q_4
q_7	q_6	q_2^*

$[q_0\ q_4 \quad [q_1 \quad [q_5] \quad [q_2]$

for $(q_0, q_5) \Rightarrow (q_1, q_2)$ so not equivalent with q_0

next check with $(q_1, q_5) = (q_6, q_2)$ so not equivalent with q_1

So q_5 go with separate group.

$[q_0\ q_4\ q_6 \quad [q_1 \quad [q_5] \quad [q_2]$

for $(q_0, q_7) \Rightarrow (q_1, q_6)$ & (q_5, q_2) so not equivalent.

Now check $(q_1, q_7) \rightarrow \delta(q_1, 0) = q_6 \quad \delta(q_1, 1) = q_2$
$\delta(q_7, 0) = q_6 \quad \delta(q_7, 1) = q_2$ So they are equivalent

$[q_0 q_4 q_6] \quad [q_1, q_7] \quad [q_5] \quad [q_2]$

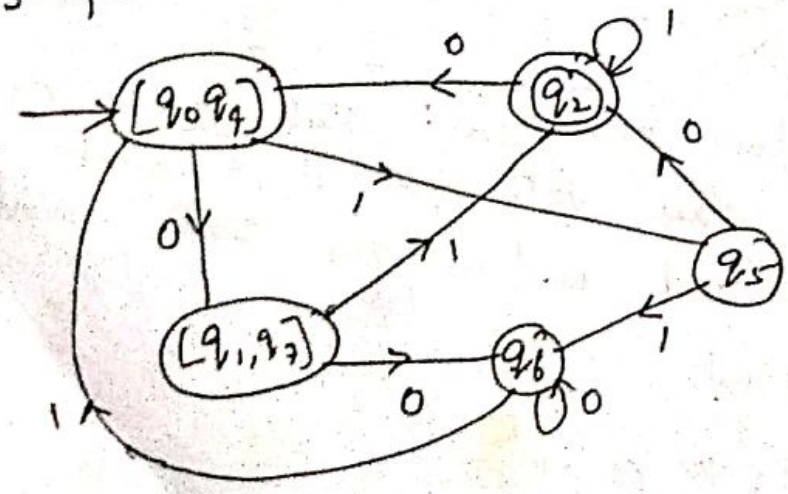
2- equivalent —

$[q_0\ q_4] \quad [q_6] \quad [q_1, q_7] \quad [q_5] \quad [q_2]$

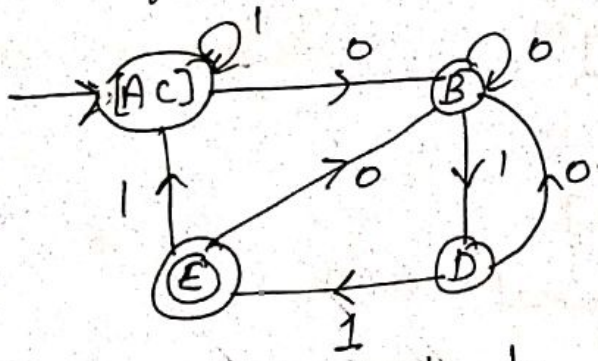
3 - equivalent

$[q_0\ q_4] \quad [q_6] \quad [q_1, q_7] \quad [q_5] \quad [q_2]$

So we get same set of groups for 2 equivalent and 3-equivalent.



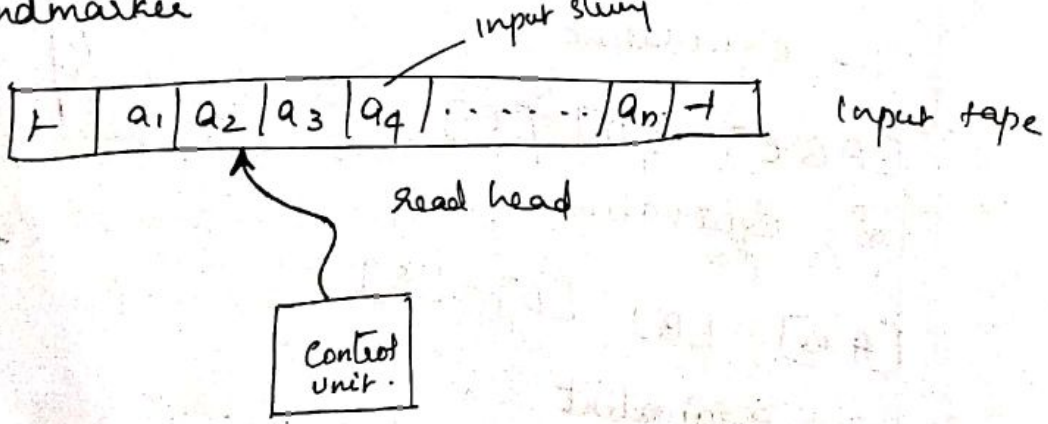
So the final minimal DFA is



Two - Way Finite Automata

2DFA is a generalized version of the DFA which can revisit characters already processed ie 2DFA can read the input back and forth with no limit

- 2DFA have a read head which can move left or right over the input string
- 2DFA consists of the symbols of the input string which is occupying in the cells of a finite tape one symbol per cell.
- The input symbol is enclosed in left and right endmarkers ⊢ and ⊣ which are not the elements of the input alphabet Σ
- The read head may not move outside of the endmarker



Formal Definition of 2DFA :-

A two-way deterministic finite automata (2DFA) is a quadruples $M = (Q, \Sigma, \delta, q_0, F)$

where

Q – finite set of states

Σ – finite set of input alphabet

q_0 – starting state

F – set of final states which is a subset of Q

δ is a transition function defined from

$Q \times \Sigma$ to $Q \times \{L, R\}$

If $\delta(q, a) = (p, L)$ then in state q, seeing an input symbol 'a', the DFA enters into state 'p' and moves its head left one cell.

If $\delta(q, a) = (p, R)$ then in state q, seeing an input symbol 'a', the DFA enters the state 'p' and moves its head Right one cell

A string is said to be accepted by a 2DFA if it reads off the right end of the tape and at the same time entering an accepting state.

eg:-

Consider the transition table given below. and check whether the string '101001' is accepted by the 2DFA or not.

States	0	1
$\rightarrow q_0$	(q_0, R)	(q_1, R)
q_1*	(q_1, R)	(q_2, L)
q_2	(q_0, R)	(q_2, L)

Acceptability of the string using 2DFA is given by

$q_0101001 \vdash 1q_101001$
$\vdash 10q_1001$
$\vdash 101q_001$
$\vdash 1010q_01$
$\vdash 10100q_1$
$\vdash 1010q_201$
$\vdash 101001q_01$
$\vdash 10100q_11$

Since the tape reaches at the right end of the input string and enters into final state the given input string is accepted.

Regular Grammar

According to chomsky hierarchy the language is divided into four type of grammar

Grammar type	Grammar Accepted	Automaton
Type-0	Unrestricted Grammar	Turing Machine
Type-1	Context Sensitive Grammar	Linear Bounded Automaton
Type-2	Context free Grammar	push down Automaton
Type-3	Regular Grammar	Finite state Automaton

A Grammar G is a set of rewriting or

productions Rule used to generate pattern of strings
A Grammar G can be described by 4 tuples as $G = (V, T, P, S)$ where

V - Set of variables or nonterminals
T - Set of terminal symbols
P - Set of production Rule for terminals and nonterminals in the form $\alpha \longrightarrow \beta$ where $\alpha, \beta \in V \cup T$ and atleast one symbol of α belongs to V
S - Start symbol

A Grammar G is said to be Regular if it may be Right linear or left linear.

A grammar is said to be Right linear, if all productions are of the form

$A \longrightarrow xB$
$A \longrightarrow x$

where $A, B \in V$ and $x \in T$

A grammar $G = (V, T, P, S)$ is said to the left linear if all productions are of the form

$A \longrightarrow Bx$
$A \longrightarrow x$

where $A, B \in V$ and $x \in T$

MOD - III

Pumping Lemma for Regular Languages

A language is said to be Regular if it can be Represented by DFA, NFA, ϵ-NFA or RE.

Pumping lemma is used to prove the language is not Regular

Pumping lemma uses pigeon hole principle to show that certain languages are not Regular.

Pigeon hole principle:-

If we put 'n' objects (pigeon) into 'm' boxes (holes) and if $n > m$, then atleast one box must have more than one item in it.

Theorem:- (pumping Lemma for Regular Set)

Let L be the Regular language. Then there exists a Constant 'n' (which is the total no: of state of the finite automata which accepts L) such that for every string 'z' in L such that $|z| \geq n$. then we can break the string 'z' into three string $z = uvw$ such that.

i) $v \neq \epsilon \quad (1 \leq |v| \leq n)$

ii) $|uv| \leq n$

iii) $uv^i w$ is in L for all $i \geq 0$

Proof :-

Suppose L is Regular. Then $L = L(A)$ for some DFA A.
Suppose A has 'n' states consider any string 'z' of length n or more ie $|z| \geq n$

Let the string be $z = a_1 a_2 a_3 \ldots a_m$ where $m \geq n$

The path corresponding to the above string z cannot be a chain in the DFA, it will be a cycle. because there are 'n' no: of states in the DFA and the length of the string is greater than 'n' $(|z| \geq n)$. [using pigeon hole principle].

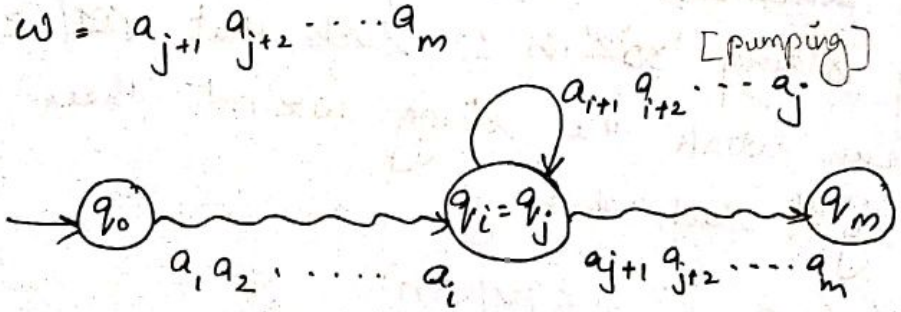
The length of the cycle will be minimum of 1 and the maximum of n

The above string z can be divided as $z = uvw$

$u = a_1 a_2 \ldots a_i$

$v = a_{i+1} a_{i+2} \ldots a_j$

$w = a_{j+1} a_{j+2} \ldots a_m$



The substring 'u' takes to q_i and 'v' takes from q_i back to q_i (since $q_i = q_j$) and w takes the balance substring of z.

Note:-
The substring 'u' can be empty when $i=0$ and w may be empty if $j=n=m$. However 'v' cannot be empty since $0 \le i < j \le n$

Consider the FA 'A' receives the input $uv^i w$ for $i \ge 0$ and $1 \le |v| \le n$.

If $i=0$ then automata goes from start state q_0 to q_i on input 'u'. Since $p_i = p_j$ the FA goes from p_i to final state on input w. Thus A accepts uw

If $i>0$ then FA 'A' goes from q_0 to p_i on input 'u', and 'A' circles from p_i to p_i for 'i' many times on input 'v^i' since $p_i = p_j$ and then goes to the accepting state on input 'w'. Thus for any $i \ge 0$, $uv^i w$ is also accepted by A. ie $uv^i w$ is in L.

Applications of Pumping Lemma

pumping Lemma is used to check whether certain sets are regular or not.

The steps to prove that certain sets are not regular are as follows.

i) Assume L is regular. There exists a constant 'n' be the number of states in the corresponding Finite automata.

ii) Choose a string 'z' such that $|z| \geq n$. Use the pumping Lemma to write $z = uvw$ with $|uv| \leq n$ and $|v| > 0$

iii) Find a suitable integer 'i' such that $uv^iw \notin L$. In some we have to prove $uv^iw \notin L$ by considering $|uv^iw|$. In some cases we have to use the structure of strings in L.

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

i) Suppose L is regular. There exists a constant 'n' be the number of states in the finite automata accepting L.

ii) Let $z = 0^n 1^n$ then $|z| = 2n > n$

By pumping lemma

$z = uvw$ with $|uv| \leq n$ & $|v| \geq 1$

iii) To find i, so that $uv^iw \in L$ for getting a contradiction. The string v can be in any one of the following forms.

a) v has only 0's i.e $v = 0^k$ for some $k \geq 1$

b) v has only 1's i.e $v = 1^l$ for some $l \geq 1$

c) v has both 0's and 1's i.e $v = 0^k 1^j$ for $k, j \geq 1$

for eg: when $n = 4$ in $z = 0^n 1^n$. the str looks like 00001111

$\underbrace{0000}_{u}\underbrace{1}_{v}\underbrace{111}_{w}$ — case a

$\underbrace{0000}_{u}\underbrace{11}_{v}\underbrace{1}_{w}$ — case b

$\underbrace{000}_{u}\underbrace{0}_{v}\underbrace{1111}_{w}$ — case c

In case a when $i=0$ in uv^iw the string $uw \notin L$ since the number of 0's is less than no: of 1's.

In case b when $i=0$ in uv^iw the string $uw \notin L$ since the number of 1's is less than no: of 0's

In case b when $i=2$ the string uv^iw becomes 000010111 in which 0101 substring occurs which is not an element of the given Language 0^i1^i. so $uv^iw \notin L$.

The above contradiction allows us to conclude the given L is not Regular.

Q Show that the set $L = \{a^{i^2} : i \geq 1\}$ is not Regular

Suppose L is Regular. Then there exists a constant n be the number of states in finite automata accepting L.

Let $z = a^{n^2}$ then $|z| = n^2 > n$

i.e $z = \{a, aaaa, aaaaaaaaa \cdots\}$

By pumping lemma

we can write $z = uvw$ with $|uv| \leq n$ & $|v| \geq 1$

Consider $i = 2$ in uv^iw

$|uv^iw| = |u| + 2|v| + |w| > |u| + |v| + |w|$ as $|v| > 0$

This means

$n^2 = |uvw| = |u| + |v| + |w| < |uv^2w|$ as $|uv| \leq n$ & $|v| > 1$

$\therefore |uv^2w| = |u| + 2|v| + |w| = n^2 + n$ [Since $|v|$ varies from 1 to n]

i.e. $n^2 < |uv^2w| \leq n^2 + n < n^2 + n + n + 1$

The perfect square after n^2 is $(n+1)^2 = n^2 + 2n + 1$

Hence $|uv^2w|$ strictly lies between n^2 and $(n+1)^2$ but $n^2 + n \neq (n+1)^2$ i.e it is not a perfect square and so $uv^2w \notin L$. This is a contradiction to pumping Lemma.

So L is not Regular

Otherwise we can prove as below also.

Let L is Regular

$L = \{a, aaaa, aaaaaaaaaa \ldots\}$

Define uv^iw form and let $i = 3$.

$z = uv^iw$

Let $z = 000000000$

Representing z in uv^iw form as

$\underset{u}{00}\underset{v}{(000000)^3}\underset{w}{0}$ with $i=3$.

But $uv^3w \notin L$. So it is a contradiction

$\therefore L$ is not Regular

Q Show that $L=\{a^p / p \text{ is a prime}\}$ is not Regular

Suppose L is Regular and let 'n' be the number of states in FSA accepting L.

Let 'p' be the prime number greater than 'n'.

Let $z = uvw = a^p$

i.e $a^p = uvw$

$|uvw| = p$

According to pumping lemma, the next element in the language is uv^iw ; put $i=p+1$

Then

$$|uv^iw| = |uvw| + |v^{i-1}|$$
$$= |uvw| + |v^{p+1-1}|$$
$$= |uvw| + |v^p|$$
$$= p + np \quad \because 1 \le |v| \le n$$
$$= p(1+n)$$

The length of the string $|uv^iw|$ is not prime. So it is a contradiction to the pumping lemma

So the given language L is not Regular

eg:

Let L be a Regular and 'n' be the number of States with FA accepting L.

Let $L = \{a, aa, aaa, aaaaa, aaaaaaa \ldots\}$

Any string can be chosen in such a way that

$|xy| \le n$ & $|y| \ge 1$

Consid $z = \underset{uvw}{aaa}$

By pumping lemma. uv^2w is in L

$uv^2w = a(a)^2a$

$= aaaa$ which does not belong to L

This is a contradiction

Hence L is not Regular.

Q Show that $L = \{0^n 1^m \mid n > m \text{ & } n \ge 0\}$ is not Regular

Assume L is Regular and let x be the no:- of States in FA which accepts L.

Let $z = uvw$ & $v = 0^p$ (since $|uv| \le n$ & $|v| \ge 1$

$= 0^{n+1} 1^n$ (since $n > 0$, $m \ge 0$ $\therefore n, m > 0$

Next string according to pumping lemma is

$uv^2w = 0^{n+1} 0^p 1^n$

$= 0^{n+1+p} 1^n$

$uv^3w = 0^{n+1+2p} 1^n$

$uv^2w = 0^{n+1+3p}1^n$

$uv^0w = 0^{n+1-p}1^n$

$n+1-p$ cannot be always greater than n.

So it is the contradiction of pumping lemma

Hence L is not Regular

Closure properties of Regular sets

A set is closed under an operation if, whenever the operation is applied to members of the set, the result is also a member of the set.

The principal closure properties for regular language.

1. The Union of the two Regular language is regular
2. The intersection of two Regular languages is Regular
3. The complement of a Regular language is Regular
4. The difference of two Regular language is Regular
5. The Reversal of a Regular language is Regular
6. The closure of the Regular language is Regular
7. The concatenation of regular language is Regular
8. A homomorphism (substitution of strings for symbols) of a Regular language is Regular
9. The inverse homomorphism of a Regular language is Regular.

Q Show that $L = \{0^n 1^m \mid n > m \text{ and } m \geq 0\}$ is not regular

Assume L is regular and let 'x' be the number of states in the finite automata

Choose a string $z = 0^n 1^m$ where $n > x$

Since $|uv| \leq x$, uv consist entirely of 0's

Choose $i = 0$ in $uv^i w$

$\therefore z = uv^i w = 0^n 1^m$

when $i = 0$ then it has the effect of dropping $|v|$ i.e 0's out of the string, without effecting the number of 1's.

The resultant string has less no: of zeros.

So that it has fewer 0's than 1's.

$\therefore$ It is a contradiction of pumping lemma

$\therefore$ L is not regular.

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

Suppose L is regular

There exists a constant 'n' be the number of states in FA which accepts L.

Let $z = a^n b^n b^n a^n$ then $|z| = 4n > n$

By pumping lemma, we can write

$z = uvw$ such that $|uv| \leq n$ & $|v| \geq 1$

Assume V contains no b's only a's

$\therefore uv = a^n$ & $w = b^n b^n a^n$

Let $z = uv^i w$ and let $i = 0$ then

$uv^0 w$ contains less no. of a's than in the RHS

$\therefore uw \notin L$

So L is not Regular

Q Show that $L = \{ww / w \in (a+b)^*\}$ is not Regular.

Suppose L is Regular then there exists a constants 'n' be the number of states in FA which accepts L.

Let $z = a^n b^n a^n b^n$ then $|z| = 4n > n$

By pumping lemma, we can write

$z = uvw$; $|uv| \leq n$ & $|v| > 0$

Assume uv contains only a

$\therefore uv = a^n$ & $w = b^n a^n b^n$

Then z can be Represented as $uv^i w$ Let $i = 0$

Then $|v|$ become empty & the remaining string

$uw \notin L$.

$\therefore L$ is not Regular.

Type 2 formalisms :-

Context Free Languages

The languages generated by the Context Free grammar is called context Free Languages.

Context Free Grammars (CFGs)

A CFG is generally represented by four tuples (V, T, P, S) where

V is set of non terminals
T is set of terminals
P is the production rule
S is the start symbol

Consider the grammar given below

$S \rightarrow aSB$
$S \rightarrow aB$
$B \rightarrow b$

For the above given grammar

$V = \{S, B\}$

$T = \{a, b\}$

$P = \{ S \rightarrow aSB,\ S \rightarrow aB,\ B \rightarrow b \}$

$S = \{S\}$

Two approaches are used to infer whether the given input string belongs to the CFG

i) Recursive inference
ii) Derivation

Recursive inference approach: use the productions rules from body to head. Here we take strings from each variable, concatenate them in proper order and infer that the resulting strings is in the language of the variable in the head.

for eg: check whether the aabb can be derived by the grammar given below

$S \to aSB$
$S \to aB$
$B \to b$

aabb
$\Rightarrow$ aaBb
$\Rightarrow$ aaBB
$\Rightarrow$ aSB
$\to$ S

body ↓ head.

Derivation approach use the grammar productions from head to body to derive a string.

eg: Check whether the string aabb can be derived from the grammar

$S \to aSB$
$S \to aB$
$B \to b$

$S \to aSB$
$\to aaBB$
$\to aabB$
$\to aabb$

head ↓ body

Derivation of the string from the given grammar can be classified into two

i) Leftmost derivation (LMD)
ii) Rightmost derivation (RMD)

In the leftmost derivation the leftmost symbol in the RHS of the production is replaced to derive the terminal

In the Rightmost derivation the Rightmost Symbol in the RHS of the production is replaced to derive the terminal.

For eg:- To derive a string aabb from the CFG

$S \rightarrow aSB$
$S \rightarrow aB$
$B \rightarrow b$

The left most derivation is as follows

$S \xrightarrow{LMD} aSB$
$\xrightarrow{LMD} aaBB$
$\xrightarrow{LMD} aabB$
$\xrightarrow{LMD} aabb$

The Right most derivation is as follows

$S \xrightarrow{RMD} aSB$
$\xrightarrow{RMD} aSb$
$\xrightarrow{RMD} aaBb$
$\xrightarrow{RMD} aabb$

Derivation tree (parse tree)

The derivation in a CFG can be represented using trees which are representing derivations are called derivation tree.

A derivation tree for a CFG, $G = (V, T, P, S)$ has the following conditions

i) The root is labeled with the start symbol
ii) The interior node is labeled with the variables in V ie non terminals

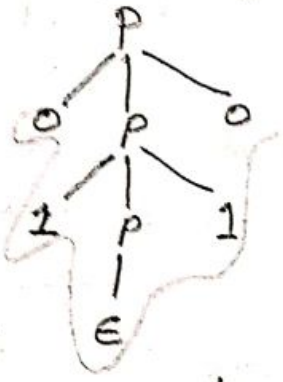
iii) Each leaf node is labeled by either a variable in V, a terminal in T or ϵ.

iv) If an interior node is labeled A, and its children are labeled $X_1, X_2, \ldots, X_k$ respectively from the left, then $A \to X_1 X_2 \ldots X_k$ is a production in P. Note that the only time one of the 'X' can be ϵ if that is the label of the only child, and $A \to \epsilon$ is a production of G.

eg: Consider the following grammar

$P \to 0P0$
$P \to 1P1$
$P \to \epsilon$

Derive the string 0110 using derivation tree



Yield of the parse tree

If we look at the leaves of the any parse tree and concatenate them from the left, we get a string called the yield of the tree, which is always a string that is derived from the root variable. The yield of the parse tree is a terminal string if all leaves are labeled either with a terminal or with ϵ.

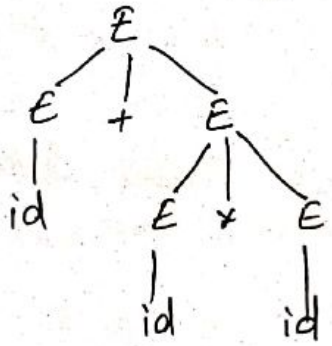
Ambiguity

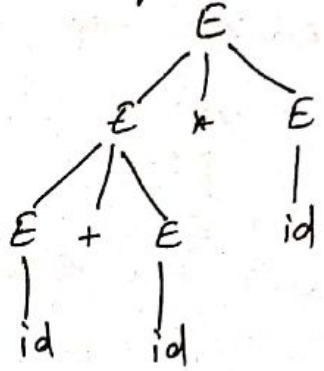
A grammar G is said to be ambiguous if it has two or more leftmost derivation, or two or more Right most derivation or two or more parse tree

Consider the grammar given below

$E \rightarrow E+E$
$E \rightarrow E*E$
$E \rightarrow id$

Derive the string id + id * id from the above G





The above is the two form of derivation tree for the given grammar to derive the string id + id * id. So that given grammar is ambiguous Since it derive two parse tree for the same string

Design of CFG

Q Construct CFG over {a,b} of all string of length 2

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

$RE = (a+b)(a+b)$.

CFG is given by

$S \rightarrow AA$
$A \rightarrow a/b$

Q Construct CFG for the string $a^n b^m c^{n+m}$; $n, m \geq 1$

$a^n b^m c^m c^n$

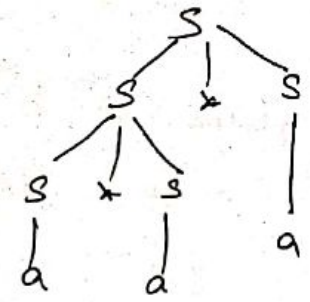
$S \rightarrow aSc \mid aAc$

$A \rightarrow bAc \mid bc$

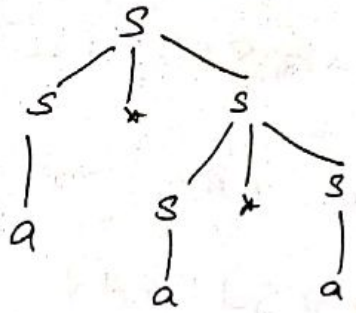
Q Show that $S \rightarrow S+S \mid S*S \mid a$ is ambiguous grammar

Consider string $W = a * a * a$

LMD



RMD

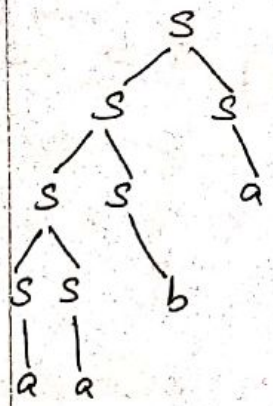


For the string $a * a * a$ we have two derivation tree so the given grammar is ambiguous.

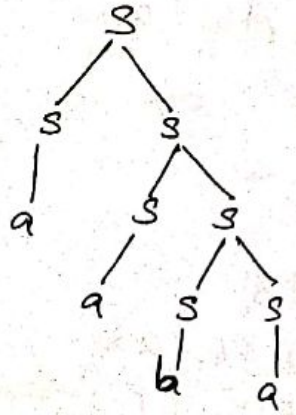
Q Show that the grammar $S \rightarrow SS \mid a \mid b$ is ambiguous

LMD

aaba



RMD



Since the grammar generate two derivation tree for the same string it is ambiguous.

Simplification of CFGs

The simplification of the CFGs can be done by 3 ways. They are

i) Eliminating useless symbols
ii) Eliminating ϵ- production
iii) Eliminating Unit production

Eliminate useless symbols

A symbol in the CFGs is useful if it is

i) generating
ii) Reachable

A symbol 'X' is said to be generating in a grammar G if it can derive some terminal string w

$X \overset{*}{\Rightarrow} w$

A symbol 'X' is said to be reachable if there is a derivation $S \overset{*}{\Rightarrow} \alpha X \beta$ for some α and β i.e X must be reachable from the start symbol.

A symbol which is generating and reachable is useful in the grammar G.

Q Eliminate useless symbols from the following Grammar

$S \rightarrow AB \mid a$
$A \rightarrow b$

Generating symbols = {A, S}
Reachable symbols = {A, B}

Since B is not generating we can eliminate B from the grammar G

$S \rightarrow a$
$A \rightarrow b$

Now A is not reachable from start symbol S so we can eliminate A from G. So the final Grammar is

$S \rightarrow a$

Q Eliminate useless symbols from the grammar given bel

$S \rightarrow aS/A/C$
$A \rightarrow a$
$B \rightarrow aa$
$C \rightarrow aCb$

Generating symbols = {A, B}
Reachable symbols = {A, C}

Eliminating C since it is not generating

$S \rightarrow aS/A$
$A \rightarrow a$
$B \rightarrow aa$

Eliminating B since it is not reachable

$S \rightarrow aS/A$
$A \rightarrow a$

So final Grammar is $S \rightarrow aS/A$, $A \rightarrow a$

B is not reachable from S. So eliminating B from G

$S \rightarrow aAa$ ← Final Grammar
$A \rightarrow bBB$
$B \rightarrow ab$

Eliminate ∈- production

To eliminate ∈ from the Grammar G, we need to find the nullable variables.
A variable 'A' is said to be nullable if

$A \xrightarrow{*} \in$

If A is nullable then whenever A appears in a production body like $B \rightarrow CAD$, 'A' might derive ∈. We can make two versions of the production, one without A in the body ($B \rightarrow CD$), which corresponds to the case when A would have been used to derive ∈ and the other with 'A' still present $B \rightarrow CAD$. If we use the version with 'A' present, then we cannot allow 'A' to derive ∈.

Q. Eliminate ∈- production from the following Grammar

$S \rightarrow AB$
$A \rightarrow aAA \mid \in$
$B \rightarrow bBB \mid \in$

Nullable Variables : {A, B, S} where A & B are direct nullable variable where as S is indirect nullable variable.

S- production with and without nullable variable can be given by

$S \rightarrow AB \mid B \mid A$

A & B- production with and without nullable variable can be given by

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

$B \rightarrow bBB \mid bB \mid b$

So the Grammar without ϵ- production is as given below

$S \rightarrow AB \mid B \mid A$
$A \rightarrow aAA \mid aA \mid a$
$B \rightarrow bBB \mid bB \mid b$

Q. Eliminate ϵ- production from the following G

$S \rightarrow AbaC$
$A \rightarrow BC$
$B \rightarrow b \mid \epsilon$
$C \rightarrow D \mid \epsilon$
$D \rightarrow d$

Nullable variables are $\{A, B, C\}$

Eliminate ϵ- production from the Grammar as follows

$S \rightarrow Abac \mid bac \mid Aba \mid ba$
$A \rightarrow BC \mid C \mid B$
$B \rightarrow b$
$C \rightarrow D$
$D \rightarrow d$

Q. Eliminate ϵ- production from the following Grammar

$S \rightarrow AaB \mid aaB$

$A \rightarrow \epsilon$

$B \rightarrow bbA \mid \epsilon$

The nullable variables are A & B.

The Grammar without ϵ-production is as given below

$S \rightarrow AaB \mid aB \mid Aa \mid a \mid aaB \mid aa$

$B \rightarrow bbA \mid bb$

Elimination of Unit production

Unit production of the Grammar G is of the form $A \rightarrow B$ where A and B are the elements in V^*

The Unit production in the Grammar is eliminated by the substitution method i.e if $A \rightarrow B$ is a Unit production in the Grammar and $B \rightarrow \alpha$ is the production in the Grammar then we can replace Unit production $A \rightarrow B$ by $A \rightarrow \alpha$ if α is a terminal symbol i.e $\alpha \in T^*$

Q. Eliminate Unit production from the following Grammar

$S \rightarrow Aa \mid B$

$B \rightarrow A \mid bb$

$A \rightarrow a \mid bc \mid B$

Unit productions in the above grammar

$S \rightarrow B$
$B \rightarrow A$
$A \rightarrow B$

The elimination of unit productions by substitution method in the CFG is given as

$S \rightarrow Aa \mid bb \mid a \mid bc$
$B \rightarrow a \mid bc \mid bb$
$A \rightarrow a \mid bc \mid bb$

Q Eliminate unit production from the grammar given below

$S \rightarrow AB$
$A \rightarrow a$
$B \rightarrow C \mid b$
$C \rightarrow D$
$D \rightarrow E$
$E \rightarrow a$

Unit production are $B \rightarrow C$, $C \rightarrow D$, $D \rightarrow E$

The final grammar after unit production elimination is given by

$S \rightarrow AB$
$A \rightarrow a$
$B \rightarrow b \mid a$
$C \rightarrow a$
$D \rightarrow a$
$E \rightarrow a$

Steps to Convert CFG to CNF is as follows:-

i) Eliminate useless symbols from the Grammar
ii) Eliminate ϵ-production from the grammar
iii) Eliminate unit productions
iv) If the RHS of any production is of the form $A \to aB$ then Replace the production by
$A \to XB$ and $X \to a$
Repeat this step for every production which is of the form $A \to aB$
v) Replace each production which is of the form $A \to B_1 B_2 B_3 \cdots B_n$ where $n \geq 3$ by $A \to B_1 C_1$ and $C_1 \to B_2 B_3 \cdots B_n$
Repeat this step for all the productions having 3 or more symbols in the RHS.

Q Convert the following CFG to CNF

$S \to ABA \mid BaA \mid A$
$A \to Ba \mid S \mid \epsilon$
$B \to Ba \mid b \mid Ca$
$C \to Ca$
$D \to DaD \mid a$

Eliminating useless symbol for the CFG

$S \to ABA \mid BaA \mid A$
$A \to Ba \mid S \mid \epsilon$
$B \to Ba \mid b$

Eliminate ε-productions for the above grammar

$S \rightarrow ABA \mid BA \mid AB \mid B \mid BaA \mid Ba \mid A$

$A \rightarrow Ba \mid S$

$B \rightarrow Ba \mid b$

Eliminate unit production for the above CFG

$S \rightarrow ABA \mid BA \mid AB \mid BaA \mid Ba \mid b$

$A \rightarrow Ba \mid ABA \mid BA \mid AB \mid BaA \mid b$

$B \rightarrow Ba \mid b$

To CNF.

$S \rightarrow AB\overset{\times}{A} \mid B\overset{\checkmark}{A} \mid A\overset{\checkmark}{B} \mid Ba\overset{\times}{A} \mid \overset{\times}{B}a \mid \overset{\checkmark}{b}$

$A \rightarrow \overset{\times}{B}a \mid A\overset{\times}{B}A \mid B\overset{\checkmark}{A} \mid A\overset{\checkmark}{B} \mid Ba\overset{\times}{A} \mid \overset{\checkmark}{b}$

$B \rightarrow \overset{\times}{B}a \mid \overset{\checkmark}{b}$

For the production $S \rightarrow ABA$, Replace it by new variable as

$S \rightarrow AX_1$

$X_1 \rightarrow BA$

For the production $S \rightarrow BaA$ Replace it by

$S \rightarrow BX_2$

$X_2 \rightarrow aA$

& $X_2 \rightarrow X_3A$

$X_3 \rightarrow a$

For the production $S \rightarrow Ba$ it is replace by

$S \rightarrow BX_3$

The production $A \rightarrow Ba$ can be replace by

$A \rightarrow BX_3$

The production $A \rightarrow ABA$ can be replaced by

$A \rightarrow AX_1$

The production $A \rightarrow BaA$ can be replaced by

$A \rightarrow BX_2$

The production $B \rightarrow Ba$ can be replaced by

$B \rightarrow BX_3$

Therefore the final CNF can be given by

$S \rightarrow AX_1 \mid BA \mid AB \mid BX_2 \mid BX_3 \mid b$

$A \rightarrow BX_3 \mid AX_1 \mid BA \mid AB \mid BX_2 \mid b$

$B \rightarrow BX_3 \mid b$

$X_1 \rightarrow BA$

$X_2 \rightarrow X_3 A$

$X_3 \rightarrow a$

Q Convert the following CFG to CNF

$S \rightarrow bA \mid aB$

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

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

The given grammar is in simplified form.
To convert it into CNF.

Greibach Normal Form (GNF)

A Grammar G is said to be in GNF iff all the production in the grammar is of the form $A \rightarrow a\alpha$ where 'a' is a terminal and 'α' is is zero or more nonterminals.

Steps to Convert CFG to GNF :-

i) Simplify the Given CFG by eliminating useless Symbols, ϵ-production and unit production.

ii) Convert the Grammar to Chomsky Normal Form

iii) Change the name of all nonterminal Symbols into some A_i's in the ascending order of 'i'.

iv) After Renaming the production should be of the form $A_i \rightarrow A_j X$ then $i < j$ and 'i' should never be greater than 'j'

v) Find the production $A_i \rightarrow A_j x$ in which $i > j$ and substitute some production in RHS to make $i = j$

vi) If $i = j$ in production $A_i \rightarrow A_j x$ then it can replaced by left Recursion method. by introducing new variable.

Left Recursion:

A production of the form $A \rightarrow A\alpha/\beta$ is said to be Left Recursion.

The left Recursion can be eliminate by the following two productions

$$A \longrightarrow A\alpha \mid \beta$$

$$\Downarrow$$

$$A' \longrightarrow \alpha A' \mid \alpha$$
$$A \longrightarrow \beta A' \mid \beta$$

In general the production can be written as

$$A \longrightarrow A\alpha_1 \mid A\alpha_2 \mid A\alpha_3 \mid \dots \mid \beta_1 \mid \beta_2 \mid \dots \beta_n$$

$$\Downarrow$$

$$A' \longrightarrow \alpha_1 A' \mid \alpha_2 A' \mid \alpha_3 A' \mid \dots \mid \alpha_1 \mid \alpha_2 \mid \alpha_2 \dots \alpha_n$$
$$A \longrightarrow \beta_1 A' \mid \beta_2 A' \mid \beta_3 A' \mid \dots \mid \beta_1 \mid \beta_2 \mid \beta_3 \dots \beta_n$$

Q Convert the following CFG to GNF

$S \rightarrow AB$
$A \rightarrow BS \mid a$
$B \rightarrow SA \mid b$

The above grammar is in simplified form and it is in CFG.

i) Renaming the above productions by A_i:

Let $S \rightarrow A_1$
$A \rightarrow A_2$
$B \rightarrow A_3$

The productions becomes

$$A_1 \rightarrow A_2 A_3 \longrightarrow (1)$$
$$A_2 \rightarrow A_3 A_1 \mid a \longrightarrow (2)$$
$$A_3 \rightarrow A_1 A_2 \mid b \longrightarrow (3)$$

The production ① & ② satisfies rule $A_i \rightarrow A_j x$ where $i<j$ but production ③ does not.

So Sub the production A_1 ie ② in ③

$A_3 \rightarrow A_2 A_3 A_2 \mid b$

The above production do not satisfy $i<j$ since

$3 > 2$

So substituting production ② in the above productions

$A_3 \rightarrow A_3 A_1 A_3 A_2 \mid a A_3 A_2 \mid b$

LR

Removing Left Recursion

$A \rightarrow A\alpha \mid \beta_1 \mid \beta_2$

$A_3 \rightarrow A_3 A_1 A_3 A_2 \mid a A_3 A_2 \mid b$

$\Downarrow$

$A' \rightarrow A_1 A_3 A_2 A' \mid A_1 A_3 A_2$

$A_3 \rightarrow a A_3 A_2 A' \mid b A' \mid a A_3 A_2 \mid b$

The above production A_3 are in GNF since all production in A_3 starts with terminal.

Substituting A_3 production in A_2

$A_2 \rightarrow a A_3 A_2 A' A_1 \mid b A' A_1 \mid a A_3 A_2 A_1 \mid b A_1 \mid a$

The A_2 production is in GNF. To get A_1 production in GNF. Substituting A_2 in A_1

$A_1 \rightarrow aA_3A_2A'A_1A_3 \mid bA'A_1A_3 \mid aA_3A_2A_1A_3 \mid bA_1A_3 \mid aA_3$

To get A' in GNF. Substitute A_1 in A' and A' becomes.

$A' \rightarrow aA_3A_2A'A_1A_3A_3A_2A' \mid bA'A_1A_3A_3A_2A' \mid aA_3A_2A_1A_3A_3A_2A \mid$
$bA_1A_3A_3A_2A' \mid aA_3A_3A_2A' \mid aA_3A_2A'A_1A_3A_3A_2 \mid$
$bA'A_1A_3A_3A_2 \mid aA_3A_2A_1A_3A_3A_2 \mid bA_1A_3A_3A_2 \mid$
$aA_3A_3A_2$

The final grammar in GNF forms are as given below

$A_1 \rightarrow aA_3A_2A'A_1A_3 \mid bA'A_1A_3 \mid aA_3A_2A_1A_3 \mid bA_1A_3 \mid aA_3$

$A_2 \rightarrow aA_3A_2A'A_1 \mid bA'A_1 \mid aA_3A_2A_1 \mid bA_1 \mid a$

$A_3 \rightarrow aA_3A_2A' \mid bA' \mid aA_3A_2 \mid b$

$A' \rightarrow aA_3A_2A'A_1A_3A_3A_2A' \mid bA'A_1A_3A_3A_2A' \mid aA_3A_2A_1A_3A_3A_2A' \mid$
$bA_1A_3A_3A_2A' \mid aA_3A_3A_2A' \mid aA_3A_2A'A_1A_3A_3A_2 \mid$
$bA'A_1A_3A_3A_2 \mid aA_3A_2A_1A_3A_3A_2 \mid bA_1A_3A_3A_2$

Q Convert the following CFG to GNF

$S \rightarrow CA \mid BB$
$B \rightarrow b \mid SB$
$C \rightarrow b$
$A \rightarrow a$

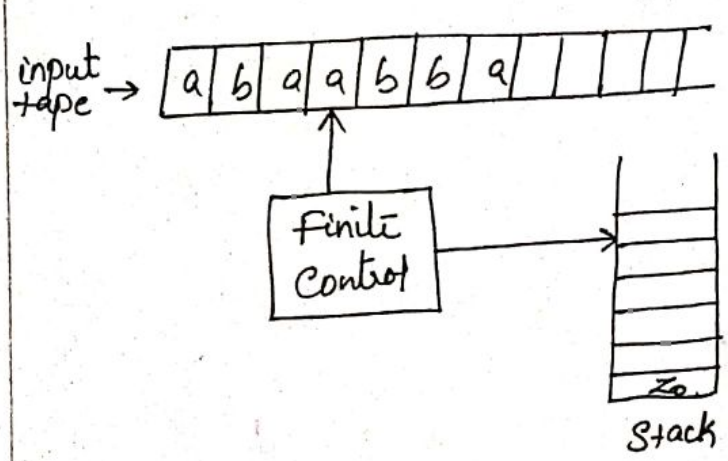
MOD - IV

Push Down Automata

Push Down Automata defines the context free Langu

Push Down Automata (PDA) is same as that of the nondeterministic finite automaton with ε-transi along with an extra memory element called sta PDA can only access the information on its stack a LIFO way

The model of the PDA is given below



Finite control reads the inputs, one symbol at a time. The PDA observe the symbols on top of the stack, and the current input symbol to make the transition

Formal definition of PDA

Formal notation of push down Automata involves seven components & can be given as follows

$$P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$$

Q: Finite set of states

Σ: finite set of input symbols

Γ: finite stack alphabet

q_0: start state

F: set of accepting or final states

z_0: start symbol of PDA's stack

δ: transition function. δ takes 3 arguments (q, a, X) where

'q' is a state in Q

'a' is either an input symbol in Σ or $a = \epsilon$, the empty string, which is assumed not to be an input symbol

'X' is a stack symbol that is a member of Γ

$\delta: Q \times (\Sigma \cup \epsilon) \times \Gamma \rightarrow Q \times \Gamma^*$

The output of δ is a finite set of pairs (p, γ) where 'p' is the new state and 'γ' is the string of stack symbols that replaces 'X' at the top of the stack. If $\gamma = \epsilon$, then the stack is popped, if $\gamma = X$, then the stack is unchanged and if $\gamma = YZ$ then the stack top 'X' is being replaced by 'Z' and 'Y' is pushed onto the top of the stack.

Instantaneous Description of PDA

The PDA configurations goes from one to another in response to the input symbol and the contents of the stack.

The Configurations of the PDA can be represent by a triple (q, w, r) where

1. q is the state
2. w is the remaining input to be encountered
3. r is the stack contents.

such a triple notation is called instantaneous description or (ID) of the push down automata.

The ID can be represented by the symbol '$\vdash$' that represents one or many moves of a PDA.

Let $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ be a PDA. Suppose $\delta(q, a, X) \to (p, \alpha)$ then for all strings w in Σ^* and β in Γ^* we can write $(q, aw, X\beta) \vdash (p, w, \alpha\beta)$.

eg:- ~~Let us consider the action of the PDA on the input ### . Since q_0 is~~

Acceptance of Input

The PDA can accept the input in two ways

i) PDA can accept input by consuming it and entering into an accepting state

ii) PDA can accept the input by empty stack.

Acceptance by Final state

Let $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ be a PDA. Then the language accepted by P by final state is

$$\{w \mid (q_0, w, z_0) \overset{*}{\underset{P}{\vdash}} (q, \epsilon, a)\}$$

for some state q in F and any stack string a. Starting in the initial ID with w waiting on the input, P consumes w from the input and enters an accepting state. The contents of the stack at that time is irrevalent.

Acceptance by empty stack

For each PDA $P = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$ we can define the language $N(P)$ accepted by P by empty stack is given by

$$N(P) = \{w \mid (q_0, w, z_0) \overset{*}{\vdash} (q, \epsilon, \epsilon)\}$$

for any state q. That is $N(P)$ is the set of inputs w that P can consume and at the same time emptying the stack.

Equivalence of acceptance by Final state & empty stack

Theorem:

If $L = N(P_N)$ is the language accepted for some PDA $P_N = (Q, \Sigma, \Gamma, \delta_N, q_0, z_0)$ by empty stack then there is a PDA P_F such that $L = L(P_F)$

Proof:

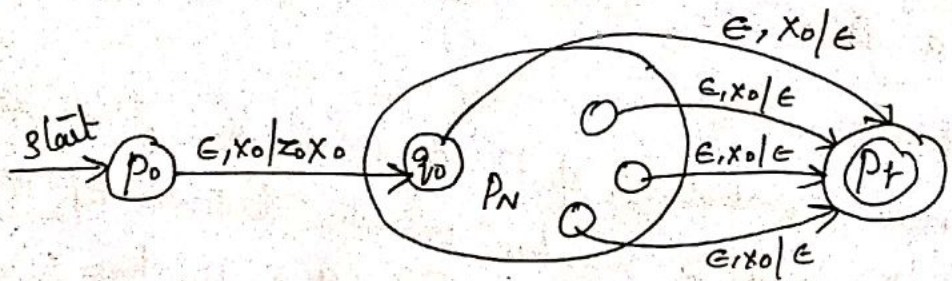
A new symbol $x_0 \notin \Gamma$ is both the start symbol of P_F and a marker on the bottom of the stack that lets the P_F to know when P_N has reached an

empty stack.
Also a new start state p_0 is need to push Z_0 a
start symbol of P_N onto the top of the stack
and enter the state q_0, the start state of P_N.
Then P_F simulates P_N until the stack of P_N
is empty, which the P_F detected by seeing X_0 o
top of the stack.
Finally a new state P_f which is the accepting
state of P_F is reached when P_N emptied its
stack.
The specification of P_F is as follows

$$P_F = (Q \cup \{p_0, p_f\}, \Sigma, \Gamma \cup \{X_0\}, \delta_F, p_0, X_0, \{p_f\})$$

where δ_F is defined by:

1. $\delta_F(p_0, \epsilon, X_0) = \{(q_0, Z_0X_0)\}$, P_F makes a spontaneous transition to the start state of P_N pushing its start symbol Z_0 onto the stack
2. For all states q in Q, a in Σ or $a = \epsilon$, and stack symbols Y in Γ. $\delta_F(q, a, Y)$ contains all the pairs in $\delta_N(q, a, Y)$
3. $\delta_F(q, \epsilon, X_0)$ contains (p_f, ϵ) for every state q in Q



w is in $L(P_F)$ iff w is in $N(P_N)$

If: w is in $L(P_N)$

Given that $(q_0, w, Z_0) \vdash_{P_N}^{*} (q, \epsilon, \epsilon)$ for some state q.

Z_0 can be inserted at the bottom of the stack and

Conclude $(q_0, w, Z_0X_0) \vdash_{P_N}^{*} (q, \epsilon, X_0)$.

P_F has all moves same as that of P_N (By Rule 2) so that $(q_0, w, Z_0X_0) \vdash_{P_F}^{*} (q, \epsilon, X_0)$

(till X_0 is on the top of stack)

So we get

$$(p_0, w, X_0) \vdash_{P_F} (q_0, w, Z_0X_0) \vdash_{P_F}^{*} (q, \epsilon, X_0) \vdash_{P_F} (p_f, \epsilon, \epsilon)$$

Thus P_F accepts w by final state.

only if:

Rule 1 causes P_N to enter the initial ID of P_F except that P_F will have its own bottom of stack marker Z_0 which is the symbol of P_F's stack. So it is given that

$$(q_0, w, Z_0) \vdash_{P_N}^{*} (q, \epsilon, \epsilon)$$

That is w is in $N(P_N)$

From Final state to empty stack

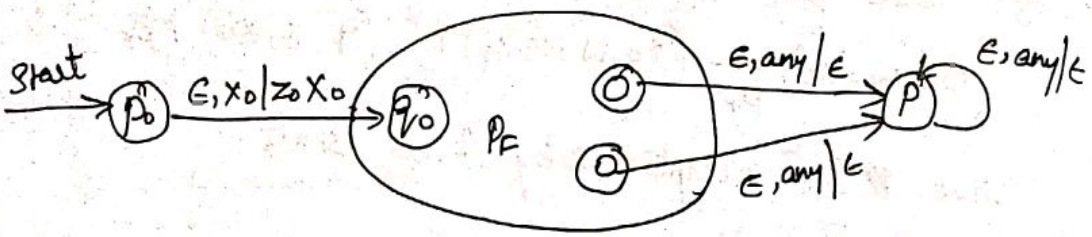
Theorem: Let L be the language accepted by $L(P_F)$ for some PDA $P_F = (Q, \Sigma, \Gamma, \delta_F, q_0, Z_0, F)$. Then there is a PDA P_N such that $L = N(P_N)$

Proof:

From each accepting state of P_F, add a transition

on ϵ to the new state p. In the state p, P_N pops its stack and does not consume any input. Thus whenever P_F enters an accepting state after consuming input w, P_N will empty its stack after consuming w.

P_N starts in a new state p_0 whose function is to push the start symbol of P_F on the stack and go to the start state of P_F.



The constructed is given as Let

$$P_N = (Q \cup \{p_0, p\}, \Sigma, \Gamma \cup \{X_0\}, \delta_N, p_0, X_0)$$

where δ_N is defined by

1. $\delta_N(p_0, \epsilon, X_0) = \{(q_0, Z_0 X_0)\}$. By pushing the start symbol of P_F onto the stack and going to the start state of P_F.
2. For all states q in Q, input symbols a in Σ or $a = \epsilon$ and Y in Γ, $\delta_N(q, a, Y)$ contains every pair that is in $\delta_F(q, a, Y)$. That is P_N simulates P_F.
3. For all accepting states q in F and stack symbol Y in Γ or $Y = X_0$. $\delta_N(q, \epsilon, Y)$ contains (p, ϵ). By this rule whenever P_F accepts, P_N can start emptying its stack without consuming any more input.

4. For all stack symbols Y in Γ or $Y = X_0$, $\delta_N(p, \epsilon, Y) = \{(p, \epsilon)\}$. One state p, which only occurs when P_F has accepted, P_N pops every symbol on its stack, until the stack is empty. No further input is consumed.

(If) part

Suppose $(q_0, w, Z_0) \overset{*}{\underset{P_F}{\vdash}} (q, \epsilon, \alpha)$ for some accepting state q and stack string α. Every transition of P_F is a move of P_N.

Let X_0 be the symbol present below all the symbols of Γ on the stack.

We know that

$(q_0, w, Z_0X_0) \overset{*}{\underset{P_N}{\vdash}} (q, \epsilon, \alpha X_0)$. Then P_N can be given as $(p_0, w, X_0) \underset{P_N}{\vdash} (q_0, w, Z_0X_0) \overset{*}{\underset{P_N}{\vdash}} (q, \epsilon, \alpha X_0) \overset{*}{\underset{P_N}{\vdash}} (p, \epsilon, \epsilon)$

The first move is by Rule (1) and the last sequence of moves is by Rule (3) and (4). Thus w is accepted by P_N by empty stack.

(Only-If) part.

P_N can empty its stack by entering the final state p' of P_F. X_0 is the bottom of the stack symbol and P_F do not have any move ~~any~~ on X_0 since it is not a symbol on the stack. The only

Way P_N can enter state p is if the simulated P_F enters an accepting state. The first move of P_N is ~~surely~~ the move given in Rule (1). Thus every accepting computation of P_N will be given as

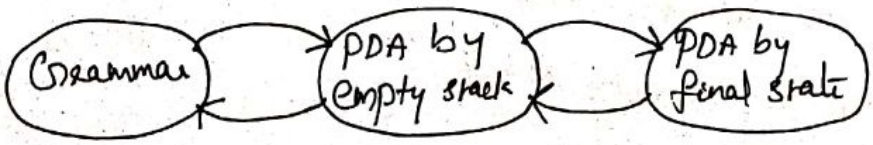
$(p_0, w, x_0) \vdash_{P_N} (q_0, w, z_0x_0) \vdash^*_{P_N} (q, \epsilon, \alpha x_0) \vdash^*_{P_N} (p, \epsilon, \epsilon)$

where q is the accepting state of P_F.

All the moves of P_N between the ID's (q_0, w, z_0x_0) and $(q, \epsilon, \alpha x_0)$ are same moves as that of P_F. Thus same computation can occur in P_F without the x_0 on the stack, that is $(q_0, w, z_0) \vdash^*_{P_F} (q, \epsilon, \alpha)$

Thus P_F accepts w by final state. So w is in $L(P_F)$.

Equivalence of NPDA's and CFG's



From Grammar to PDA

For a given CFG G, we can construct a PDA that simulates the leftmost derivation of G. Any left sentential form that is not a terminal string can be written as $xA\alpha$ where x is the terminal appear to its left, 'A' is the non-terminal and 'α'

is the string of terminals and nonterminals appear to the right of A.

'Aα' is called the tail of the left-sentential form.

If a left sentential form consists of terminals only then its tail is ∈.

The construction of PDA from Grammar can be done by simulating the sequence of left-sentential forms that the grammar uses to generate a given terminal string ω by the PDA.

The tail of each sentential form xAα appears on the stack with A at the top. and 'x' will be consumed from the input leaving ω follows its prefix x. The top symbol of the stack A is being replaced by β if A → β is the production for 'A' in G.

The PDA P that accepts L(G) by empty stack is as follows.

$$P = (\{q\}, T, V \cup T, \delta, q, S)$$

where δ transition function is given by

1. For each variable A

$$\delta(q, \epsilon, A) = \{q, \beta \mid A \to \beta \text{ is a production of } G\}$$

2. For each terminal a, $\delta(q, a, a) = \{(q, \epsilon)\}$.

Theorem:-

If PDA P is constructed from CFG G by the

above then $N(P) = L(G)$

Proof:-

We have to prove that w is in $N(P)$ if and only if w is in $L(G)$

If (part)

Suppose w is in $L(G)$. Then w has a leftmost derivation

$$S = \gamma_1 \underset{lm}{\Rightarrow} \gamma_2 \underset{lm}{\Rightarrow} \cdots \underset{lm}{\Rightarrow} \gamma_n = w$$

By induction on i we can show than $(q, w, S) \overset{*}{\vdash}_P (q, y_i, \alpha_i)$ where y_i and α_i are the representation of left sentential form γ_i.

where y_i is the remaining input to be consumed and α_i is the tail of the γ_i

BASIS :-

For $i = 1$, $\gamma_1 = S$, Thus $x_1 = \epsilon$. and $y_1 = w$ Since $(q, w, S) \overset{*}{\vdash} (q, w, S)$ by 0 moves. Thus basis is proved.

INDUCTION :-

Consider the case of the second and subsequent left sentential forms. we assume

$(q, w, S) \overset{*}{\vdash} (q, y_i, \alpha_i)$ and prove $(q, w, S) \overset{*}{\vdash} (q, y_{i+1}$

Since α_i is a tail it begins with a variable A.

By replacing A by β on top of the stack by Rule 1

Using the productions of A in G and then replacing B by terminals by Rule 2 allows us to match any terminals on top of the stack with the next input symbol. So we reach the PDA with ID as $(q, y_{i+1}, \alpha_{i+1})$ which represent the next left sentential form γ_{i+1}.

Since the tail of γ_n is empty (which is w) $\alpha_n = \epsilon$ Thus $(q, w, S) \vdash^* (q, \epsilon, \epsilon)$ which proves that P accepts w by empty stack.

(Only-if) part.

Let x be the input accept by N(P) then

If $(q, x, A) \vdash_P^* (q, \epsilon, \epsilon)$ then $A \overset{*}{\underset{G}{\Rightarrow}} x$

The proof is an induction on the number of moves taken by P.

BASIS:

If P has one move. The only possibility is that $A \to \epsilon$ is a production of G. In this case $x = \epsilon$

So $(q, \epsilon, A) \vdash_P^* (q, \epsilon, \epsilon)$

INDUCTION:

Suppose P takes n moves; $n > 1$

The first move must be of Rule(1) where A is replaced by one of its production bodies on the top of the stack.

Suppose the production used is $A \to Y_1 Y_2 \cdots Y_k$, where each Y_i is either a terminal or variable. In the next $n-1$ moves P must consume x from the input by poping each of the $Y_1 Y_2$ from the stack one by one.

The input x can be break as $x = x_1 x_2 \cdots x_k$ | each x_i is the part of the input string and is consumed by P by popping each Y_i from the top of the stacks.

If Y_i is a terminal, then there must be only one move involved and it matches the one symbol of x_i against Y_i which are the same. so $Y_i \overset{*}{\Rightarrow} x_i$

so we have derivation

$$A \Rightarrow Y_1 Y_2 \cdots Y_k \overset{*}{\Rightarrow} x_1 Y_2 \cdots Y_k \overset{*}{\Rightarrow} \cdots \overset{*}{\Rightarrow} x_1 x_2 \cdots x_k$$

That is $A \overset{*}{\Rightarrow} x$

To complete the proof let $A = S$ (start symbol) and $x = w$ (input). Since w is in $N(P)$.

$(q, w, S) \overset{*}{\vdash} (q, \epsilon, \epsilon)$. and we proved that $S \overset{*}{\Rightarrow} w$. That is w is in $L(G)$.

Deterministic push Down Automata

A PDA is deterministic if there is never a choice of move in any situation. These choices are of two kinds. If $\delta(q, a, X)$ contains more than one pair

then the PDA is nondeterministic because we can choose among these pairs when deciding on the next move. If $\delta(q,a,x)$ is a singleton then we have a choice between using a real input symbol or making an ϵ move.

A PDA $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ to be deterministic if and only if the following conditions are met:

1. $\delta(q,a,x)$ has at most one member for any q in Q, a in Σ or $a = \epsilon$ and x in Γ
2. If $\delta(q,a,x)$ is nonempty, for some a in Σ, then $\delta(q,\epsilon,x)$ must be empty.

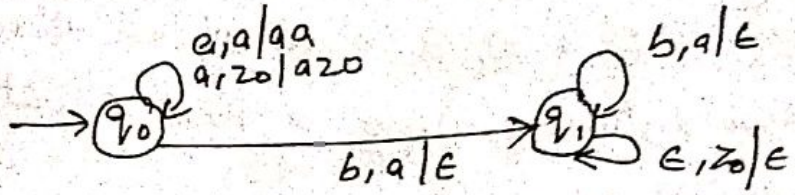
Design of PDA

Q Design a PDA to accept the language $a^n b^n / n \geq 1$

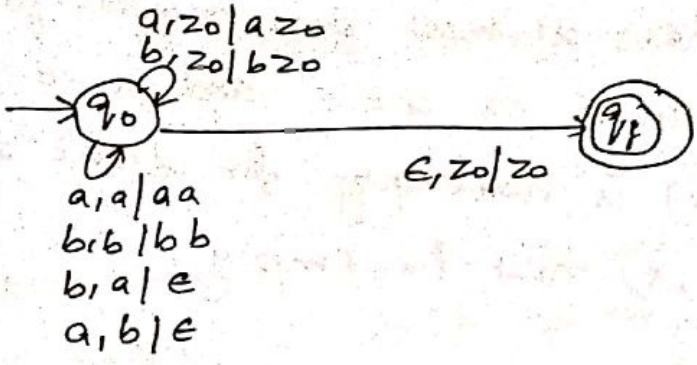
The ID of the above PDA can be given by

$\delta(q_0, a, Z_0) = (q_0, aZ_0)$
$\delta(q_0, a, a) = (q_0, aa)$
$\delta(q_0, b, a) = (q_1, \epsilon)$
$\delta(q_1, b, a) = (q_1, \epsilon)$
$\delta(q_1, \epsilon, Z_0) = (q_2, Z_0)$

Q Design a PDA for the language $a^n b^n / n \geq 1$ by empty stack

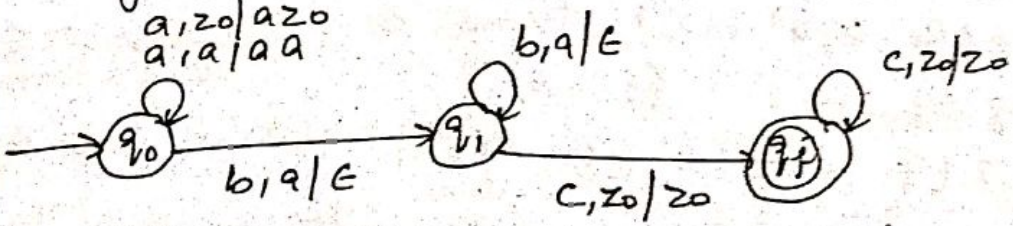


Q Design a PDA to accept the string w such that $n_a(w) = n_b(w)$

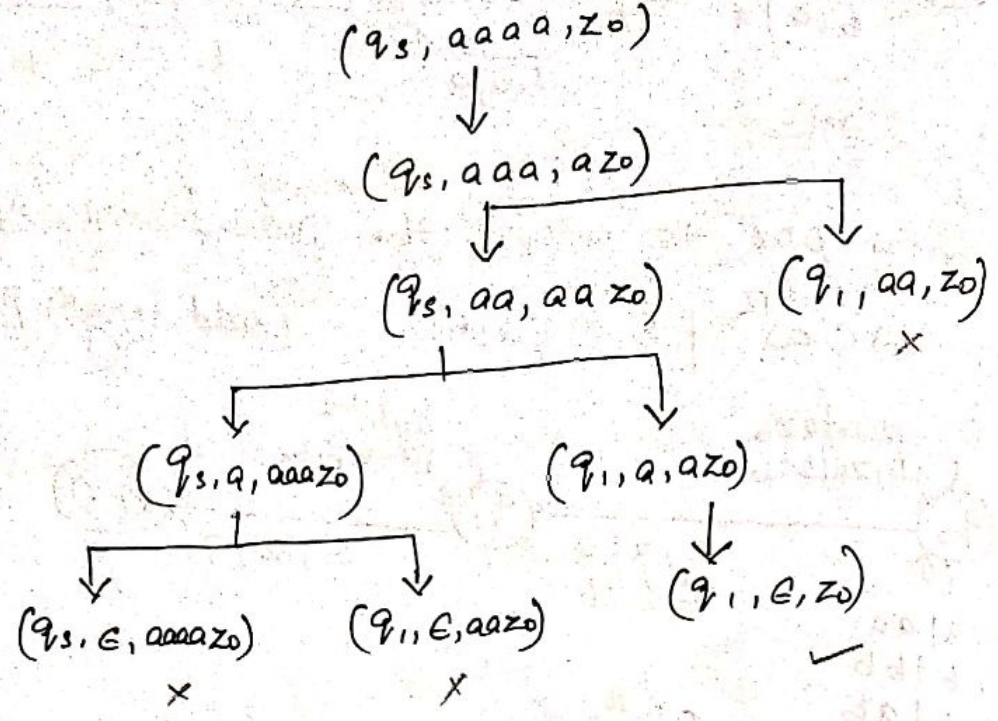


$\delta(q_0, a, z_0) = (q_0, az_0)$
$\delta(q_0, b, z_0) = (q_0, bz_0)$
$\delta(q_0, a, a) = (q_0, aa)$
$\delta(q_0, b, b) = (q_0, bb)$
$\delta(q_0, a, b) = (q_0, \epsilon)$
$\delta(q_0, b, a) = (q_0, \epsilon)$
$\delta(q_0, \epsilon, z_0) = (q_f, z_0)$

Q Design a PDA to accept the language $a^n b^n c^m / n, m \geq 1$



Consider the input aaaa



In deterministic PDA $\delta(q,a,a) = (p,a)$

But in nondeterministic PDA $\delta(q,a,a) = (p,a), (q,r), (s,t)$

Note:-

NPDA is more powerful than DPDA since NPDA accepts more languages than DPDA

Closure properties of CFL

CFL is closed under substitutions

CFL is closed under union, concatenation, closure and Homomorphism

CFL is closed under reversal

CFL is not closed under intersection and complement

CFL is closed under inverse homomorphism

If L is a CFL and R is a regular language then $L \cap R$ is a CFL.

Decision problems Related with CFL's

i) Emptiness:

In a grammar G, $L(G)$ is nonempty if and only if 'S', the starting symbol is useful, otherwise $L(G)$ is empty

ii) Finite:

Assume that G is in chomsky normal form with no unit productions, no useless symbols and no nullable symbols.

In CNF all productions are of the form

$A \rightarrow a$

or

$A \rightarrow BC$

Let 'H' be a graph with vertices V. Draw an arrow from $A \rightarrow B$ and $A \rightarrow C$ for each production $A \rightarrow BC$. Then $L(G)$ is finite if and only if H has no cycles.

iii) Infinite :-

Assume that G is in chomsky normal form with no unit productions, no useless symbols and no nullable symbols.

In CNF all productions are of the form

$A \rightarrow a$

or

$A \rightarrow BC$

Let H be the graph with vertices V. Draw an arrow from $A \to B$ and $A \to C$ for each production $A \to BC$. Then $L(G)$ is infinite if and only if it has atleast one cycle in a directed graph.

Example:-

Consider the grammar given below

$S \to AB$

$A \to BC/a$

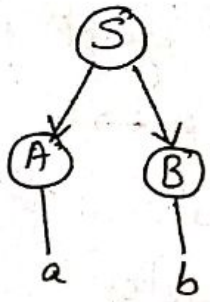
$B \to CC/b$

$C \to AB/a$

i) Check the above given grammar is empty or not

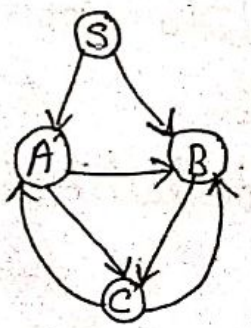
for the input ab

$S \to AB \to aB \to ab$



ii) Check whether the above given grammar is infinite or not:

If the grammar is infinite then the directed graph contains the cycle.

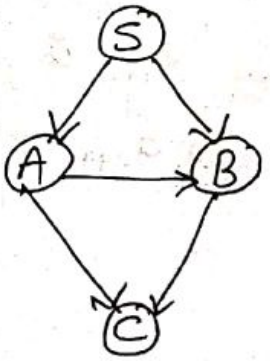


Cycle – BCB

– ACA

iii) Check whether the grammar given below is finite or not

$S \rightarrow AB$
$A \rightarrow BC/a$
$B \rightarrow CC/b$
$C \rightarrow a$



The directed graph do not contain any cycle So the given grammar is finite.

Membership Algorithm:-

CYK (Cocke - Younger - Kasami) algorithm is used for finding whether the given string is a member of the given grammar or not.

The input to the CYK algorithm should be a Chomsky Normal Form that

CYK Algorithm starts with a CNF grammar $G=(V,T,P,S)$ for the language L. The input to the algorithm is a string $w = a_1 a_2 \ldots a_n$ in T^*.

The Complexity of the algorithm is $O(n^3)$ in the

algorithm constructs a table that tells whether w is in L in $O(n^3)$ time

The construction of the triangular table is given below.

The horizontal axis corresponds to the positions of the string $w = a_1 a_2 \cdots a_n$ which we have a length of 5. The table entry X_{ij} is the set of variables 'A' such that $A \overset{*}{\Longrightarrow} a_i a_{i+1} \cdots a_j$

X_{15}				
X_{14}	X_{25}			
X_{13}	X_{24}	X_{35}		
X_{12}	X_{23}	X_{34}	X_{45}	
X_{11}	X_{22}	X_{33}	X_{44}	X_{55}
a_1	a_2	a_3	a_4	a_5

To fill the table, we are moving row-by-row upwards. Each row corresponds to one length of substring, the bottom row is for strings of length 1, the second from bottom row for strings of length 2, and so on, until the top row corresponds to the one substring of length 'n' which is 'w' itself.

CYK Algorithm

begin

1. for $i = 1$ to n do
2. $V_{i1} = \{ A \mid A \rightarrow a$ is a production and the i^{th} symbol of x is $a \}$
3. For $j = 2$ to n do
4. For $i = 1$ to $n-j+1$ do

   begin
5. $V_{ij} = \phi$
6. For $k = 1$ to $j-1$ do
7. $V_{ij} = V_{ij} \cup \{ A \mid A \rightarrow BC$ is a production, B is in V_{ik} and C is in $V_{i+k, j-k} \}$

   end.

end.

Q Consider the following CFG,

$S \rightarrow AB \mid BC$

$A \rightarrow BA \mid a$

$B \rightarrow CC \mid b$

$C \rightarrow AB \mid a$

Check whether the input baaba is in the CFG or not.

To check whether the input is in CFG or not Draw the triangular table in which the bottom row consists of five columns.

S,A,C (15)				
ϕ (14)	S,A,C (24)			
ϕ (13)	B (23)	B (33)		
A,S (12)	B (22)	S,C (32)	A,S (42)	
B (11)	A,C (21)	A,C (31)	B (41)	A,C (51)
b	a	a	b	a

$t_{11} = B$ $(B \to b)$

$t_{21} = A, C$ $(A \to a, C \to a)$

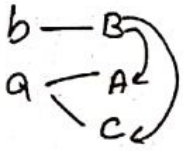
$t_{31} = A, C$ $(A \to a, C \to a)$

$t_{41} = B$ $(B \to b)$

$t_{51} = A, C$ $(A \to a, C \to a)$

t_{12} contain string of length 2.

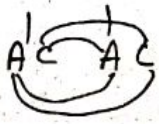
ie t_{12} contain ba



$\therefore ba = (B.A)$ or $(B.C)$

$\therefore t_{12} = A, S$ $(A \to BA, S \to BC)$

$t_{22} =$ a a



$\therefore t_{22} = B$

AA, AC, CA, CC

ϕ ϕ ϕ B

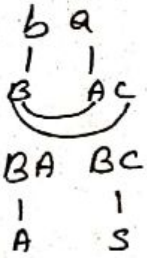
$t_{32} =$ a b

AB CB

S,C ϕ

$\therefore t_{32} = Sc$

$t_{42} = b\ a$

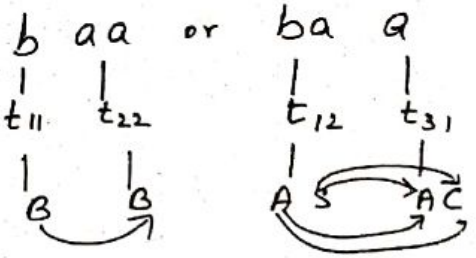


$\therefore t_{42} = A, S$

$t_{13} = b\ a\ a$

$\Downarrow$

$b\ aa$ or $ba\ a$



$\therefore t_{13} = \phi$

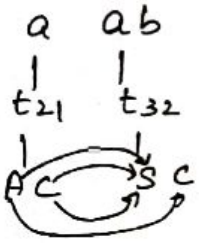
. BB AA AC SA SC

ϕ ϕ ϕ ϕ ϕ

$t_{23} = a\ a\ b$

$\Downarrow$

$a\ ab$ $a\ a$ b



$\therefore t_{23} = B$

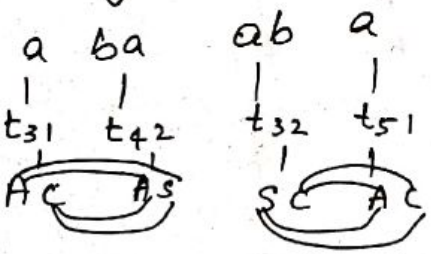
AS AC CS CC BB

ϕ ϕ ϕ B ϕ

$t_{33} = a\ b\ a$

$\Downarrow$

$a\ ba$ $ab\ a$



$\therefore t_{23} = B$

AA AS CA CS SA SC CA CC

ϕ ϕ ϕ ϕ ϕ ϕ ϕ B

t_{14} - b a a b

b	aab	ba	ab	baa	b
t_{11}	t_{22}	t_{12}	t_{32}	t_{13}	t_{41}
B	B	A S	S C	ϕ	B

$\therefore t_{14} \neq \phi$

BB	A S A C	S S S C	ϕ
ϕ	ϕ ϕ	ϕ ϕ	

t_{24} - a a b a

a	aba	aa	ba	aab	a
t_{21}	t_{33}	t_{22}	t_{42}	t_{23}	t_{51}
A C	B	B	A S	B	A C

$\therefore t_{24} = S, A, C$

A B	C B	B A	B S	B A	B C
S, C	ϕ	A	ϕ	A	S

t_{15} = b a a b a

b	a a b a	ba	aba	baa	ba	baab	a
t_{11}	t_{24}	t_{12}	t_{33}	t_{13}	t_{42}	t_{14}	t_{51}
B	S A C	A S	B	ϕ	A S	ϕ	A C

B S	B A	B C	A B	S B	ϕ	ϕ
ϕ	A	S	S, C	ϕ		

$\therefore t_{15} = S, A, C$

The topmost cell contain start symbol of CNF 'S'.
So the given string baaba is in $L(G_2)$.

Pumping Lemma For CFL

Theorem:-

Let 'L' be a CFL. Then there exists a constant 'n' such that if z is any string in L such that |z| is at least n, then we can write z = uvwxy with the following conditions.

1. $|vwx| \leq n$. That is the middle portion is not too long.
2. $vx \neq \epsilon$. Since 'v' and 'x' are the pieces to be 'pumped' this condition says that atleast one of the strings we pump must not be empty
3. For all $i \geq 0$, uv^iwx^iy is in L. That is the two strings v and x may be pumped any number of time, including 0, and the resulting string will still be a member of L.

Proof:-

Start with a CNF grammar $G = (V, T, P, S)$ such that $L(G) = L - \{\epsilon\}$

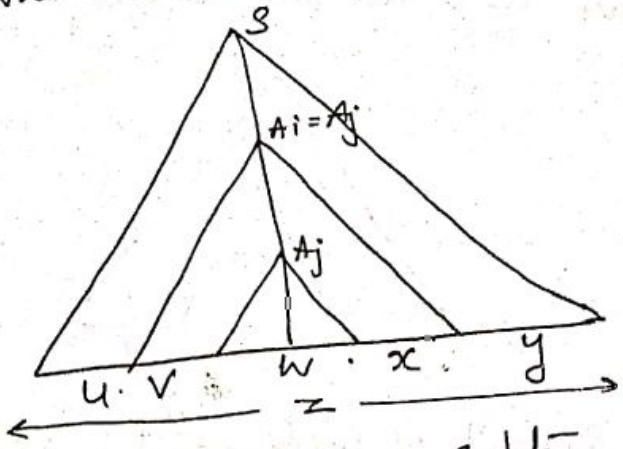
Let G has 'm' variables. Choose $n = 2^m$

Suppose that z is in L and $|z| \geq n$

We claim that any parse tree whose longest path is of length m or less must yield of length 2^{m-1}

Such a parse tree cannot yield z which has the path of length atleast m+1

The parse tree with longest path 2^{m+1} is possible by dividing the tree as below.

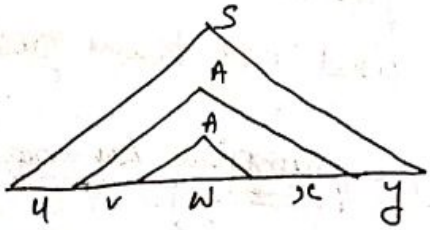


String 'w' is the yield of the subtree rooted Aj. String 'v' and 'x' are the strings at the left and right of w is the yield of the larger subtree rooted at A_i.

Since the G is CNF, there are no unit productions and the strings 'v' and 'x' could not both be ∈

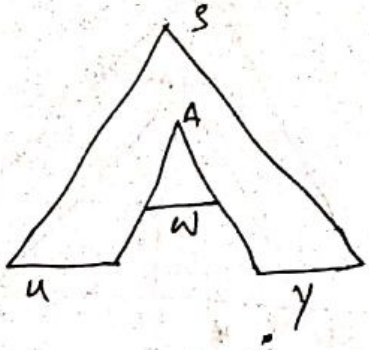
The string 'u' and 'y' are the portions of 'z' that are left and right of the subtree rooted at A_i.

If $A_i = A_j = A$ then the new parse tree becomes

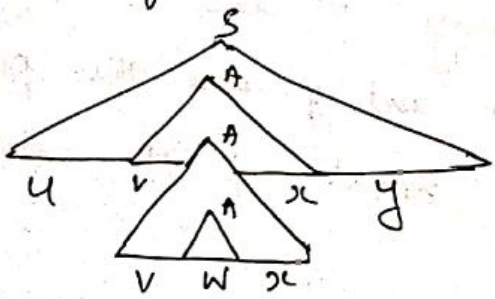


First replace the subtree rooted at A_i which yield vwx by the subtree rooted at A_j which yield w. We can do so because each of these subtrees have root labelled A

The resulting tree yield uwy which corresponds to case $i=0$ in the string $uv^i wx^i y$



Second method of replacing is by using A_j which yield vwx in each subtree A_i is replaced by A_j.
Since A_i and A_j having root labeled as A we can substitute one subtree over another rooted subtree. The yield of this substitution is uv^2wx^2y The parse tree of this substitution is as given below



Thus there are parse trees in G for all strings of the form $uv^i wx^i y$ and we have proved the pumping lemma

Applications of the Pumping Lemma

Pumping Lemma is used to prove the given language is not a context Free language. The steps for proving the language is not context free is as given below

1. Pick a language L that we want to show is not a CFL
2. Pick n such that $|z| \geq 2^m$ where m is the Variables in G
3. Break z into $uvwxy$ subject to the constraints that $|vwx| \leq n$ & $vx \neq \epsilon$
4. Pick appropriate i and show that uv^iwx^iy is not in in L. which is a contradiction to CFL

Q Show that $a^n b^n c^n \mid n \geq 0$ is not a CFL.

Assume L is context Free.

L must have a pumping length p.

Consider a string z such that $z = a^p b^p c^p$

Let $z = uvwxy$

put $p = 4$ then $z = a^4 b^4 c^4$

$\therefore z =$ aaaabbbbcccc

Case-1: v and y contains only one type of symbol

$\underline{aaa}\ \underline{a}\ \underline{bbb}\ \underline{b}\ \underline{c}\ \underline{c}\ \underline{c}\ \underline{c}$

$u\ v\ w\ x\ y$.

for $i = 2$ in ~~uvwxyz~~ uv^iwx^iy

aaaaaabbbbccccc $\notin L$. This is a contradiction. So L is not CFL

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

$$\underbrace{a\,a\,a}_{u}\;\underbrace{a\,b}_{v}\;\underbrace{b\,b}_{w}\;\underbrace{b\,c}_{x}\;\underbrace{c\,c\,c}_{y}$$

put $i=2$ in uv^iwx^iy.

then z becomes aaaabababbbbcbccccc $\notin L$

so it is a contradiction $\therefore$ L is not CFL

Q Show that $ww \mid w \in \{0,1\}^*$ is not CFL

Assume L is CFL

L must having a pumping length p

Consider string $z = 0^p1^p0^p1^p$

Break z into uvwxy

put $p=3$ then $z = 000111000111$

$$\underbrace{000}_{u}\;\underbrace{1}_{v}\;\underbrace{1}_{w}\;\underbrace{0}_{x}\;\underbrace{00111}_{y}$$

put $i=2$ in uv^iwx^iy then

z becomes $0001110000111 \notin L$

This is a contradiction. so L is not CFL.

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

Assume L is CFL.

Let n be the natural number

Let $z = a^p$ where $p > n$

$z = uvwxy$

put $p = 7$

then $z = \underbrace{a}_{u}\underbrace{a}_{v}\underbrace{a}_{w}\underbrace{a}_{x}\underbrace{aaa}_{y}$.

put $i = 5$ in $uv^iw x^i y$

$aaaaaaaaaaaaa \notin L$

This is a contradiction ∴ so L is not in CFL.

Q Prove that $L = \{ 0^{2^i} \mid i > 1 \}$ is not context free

Assume L is context free.

Let n be the natural number

Let $z = 0^{2^n}$

put $n = 3$ then $z = 0^{2^3}$

ie $z = 00000000$

Break z as uvwxy

$z = \underbrace{0}_{u}\underbrace{0}_{v}\underbrace{0}_{w}\underbrace{0}_{x}\underbrace{0000}_{y}$

put $i = 2$ in $uv^iw x^i y$

Then z becomes $0000000000 \notin L$

This is a contradiction. ∴ L is not a CFL.

Type 1 Formalism :-

Context Sensitive Grammar

A grammar $G = (V, T, P, S)$ is said to be context sensitive if all production of the grammar is of the form $\alpha \longrightarrow \beta$ where $\alpha, \beta \in (V \cup T)^+$ and $|\alpha| \leq |\beta|$

Context sensitive Grammar does not contain the empty string ie $\in$. ie A CSG never generate $X \longrightarrow \in$ production.

A Language L is said to be context sensitive if there exists a context sensitive grammar G such that $L = L(G)$ or $L = L(G) \cup \{\in\}$

for eg:-

Consider the language $L = a^n b^n c^n \mid n \geq 1$ is a Context sensitive Language. Derive string $a^3 b^3 c^3$ from the following given productions

$S \rightarrow abc \mid aAbc$

$Ab \rightarrow bA$

$Ac \rightarrow Bbcc$

$bB \rightarrow Bb$

$aB \rightarrow aa \mid aaA$

Initially start with the start symbol S

$S \rightarrow aAbc$
$\rightarrow abAc$
$\rightarrow abBbcc$
$\rightarrow aBbbcc$
$\rightarrow aaAbbcc$
$\rightarrow aabAbcc$
$\rightarrow aabbAcc$
$\rightarrow aabbBbccc$
$\rightarrow aabBbbccc$
$\rightarrow aaBbbbccc$
$\rightarrow aaabbbccc = a^3b^3c^3$

Properties of CSL

1) CSL is closed under union, concatenation, positive closure.
2) CSL is closed under ϵ-free homomorphism, inverse homomorphism
3) CSL is closed under intersection, substitution and Reversal
4) Intersection of Regular set with CSL gives CSL

Linear Bounded Automata (LBA)

A Linear Bounded Automation (LBA) is a non deterministic TM. It is denoted by 8 tuples of the form $M = (Q, \Sigma, \Gamma, \delta, q_0, \text{¢}, \$, F)$ where

Q - Set of states
Σ - input symbol
Γ - tape symbols
q_0 - initial state
F - final state
δ is the transition function which can be given by

$$Q \times \Sigma \longrightarrow \Gamma \times (R/L) \times Q$$

¢ and $ are symbols in Σ

¢ → left end marker of input

$ → Right end marker of input.

The input symbol of the LBA includes two special symbols ¢ and $. The LBA has no moves left from ¢ and Right from $.

The Language L is accepted by LBA M can be given by $\{w/w$ is in $(\Sigma - \{¢, \$\})^*$ and

$q_0 ¢ w \$ \xrightarrow[M]{*} \alpha q \beta$ for some q in F.

The Instantaneous Description (ID) of LBA is denoted by $\alpha q \beta$ where $\alpha, \beta \in \Gamma^*$

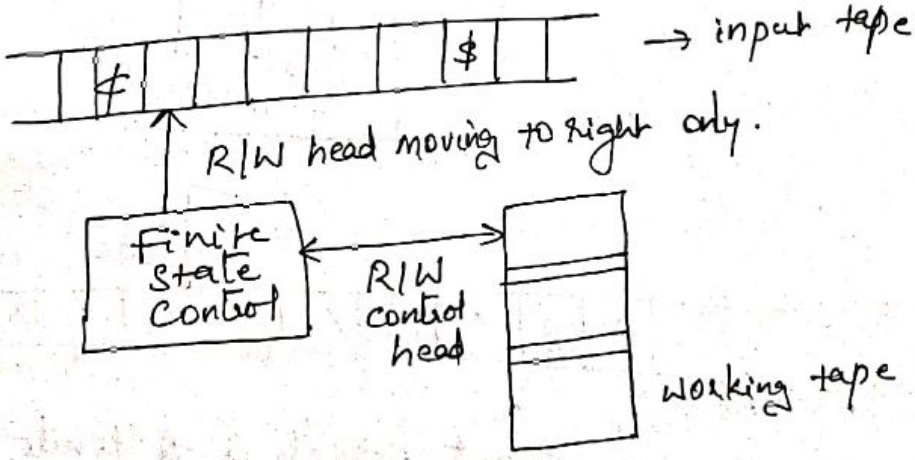
The initial configuration is given by $q_0 ¢ w \$$.

A string w is accepted by LBA if $q_0 ¢ w \$ \vdash^* \alpha q_f \beta$ for some $q_f \in F$ and $\alpha, \beta \in \Gamma^*$

Model of LBA

A Linear Bounded automation is like a standard TM. It has an unbounded tape but how much of the tape can be used is a function of the input.

The usable part of the tape is restricted to exactly the cells taken by the input. The input is bounded inside two special symbols ¢ and $ in the tape. ie input is bounded between the left end marker and right end marker. The end markers cannot be rewritten. The R/W head of the finite control cannot move to the left of ¢ and to the right of $. R/W head can move right or left within the restricted cell of the tape. Tape is infinite but tape head is linearly bounded within the input.



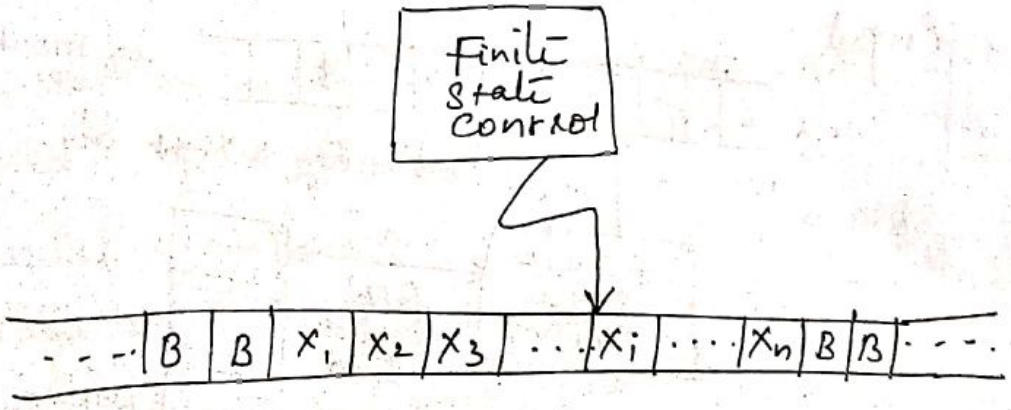
Type 0 Formalism:

Turing Machine (TM)

Alan Turing introduced a new mathematical model called Turing Machine during the year of 1936. It is mostly used to define languages and to compute integer functions. TM is a finite automaton that has a single tape of infinite length on which it may read and write data. One advantage of TM over programs is its simplicity i.e we can represent the TM configuration using a simple notation like ID's of the PDA.

Notation for Turing Machine

The machine consists of a finite control, which can be in any of a finite set of state. There is a tape divided into squares or cells, each cell can hold any one of the finite number of symbols



Initially, the input which is a finite length strings of symbols chosen from the input alphabet is

placed on the tape. All other tape cells extending infinitely to the left and right initially hold a special symbol called the blank symbol. The blank is a tape symbol, but not an input symbols. and there can be other tape symbols besides the input symbols and the blank.

There is a tape head that is always positioned at one of the tape cells. The TM is said to be scanning that cell. Initially the tape head is at the leftmost cell that holds the input.

A move of the TM is a function of the state of the finite control and the tape symbol scanned. In one move, the TM will

1. Change state. The next state may be ~~the~~ same as the current state.
2. Write a tape symbol in the cell scanned. This tape symbol replaces whatever symbol was in that cell. The symbol written may be the same as the symbol currently there.
3. Move the tape head left or right.

The formal notation of the Turing Machine (TM) can be given as 7-tuple in the form given below

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

where Q : set of states of the finite control

Σ : set of input symbols

Γ : set of tape symbols

q_0 : start state

B : Blank symbol. This symbol is in Γ but not in Σ. Initially blank symbols appears in all cells except the cells contain finite input symbols

F : set of final or accepting states

δ : transition function. The argument of $\delta(q,x)$ are a state q and a tape symbol x. The value of $\delta(q,x)$ is defined as a triple (p,Y,D) where

1 : p is the next state in Q

2 : Y is the symbol in Γ, written in the cell being scanned by replacing the current symbol

3 : D is the direction of tape head. It can be either 'left' or 'right' direction denoted by 'L' or 'R'

Instantaneous Description of TM

The ID of the TM M can be denoted as $X_1X_2\cdots X_{i-1}qX_iX_{i+1}\cdots X_n$ in which 'q' is the state of the TM.

The tape head is scanning the i^{th} symbol from the left

$X_1 X_2 \cdots X_n$ is the portion of the tape between the leftmost and the rightmost nonblank

The move of the TM can be described by '$\vdash$'

Suppose $\delta(q, X_i) = (p, Y, L)$ is next move is leftward.

Then $X_1 X_2 \cdots X_{i-1} q X_i X_{i+1} \cdots X_n \vdash_M X_1 X_2 \cdots X_{i-2} p X_{i-1} Y X_{i+1} \cdots X_n$

The tape head is now positioned at cell $i-1$. There are two important exceptions.

1. If $i = 1$, then M moves to the blank to the left of X_1. In that case $q X_1 X_2 \cdots X_n \vdash_M p B Y X_2 \cdots X_n$
2. If $i = n$ and $Y = B$ then the symbol B is written over X_n. Thus $X_1 X_2 \cdots X_{n-1} q X_n \vdash_M X_1 X_2 \cdots X_{n-2} p X_n$

Now suppose $\delta(q, X_i) = (p, Y, R)$ is next move is rightward

Then $X_1 X_2 \cdots X_{i-1} q X_i X_{i+1} \cdots X_n \vdash_M X_1 X_2 \cdots X_{i-1} Y p X_{i+1} \cdots X_n$

The tape head has moved to '$i+1$'th cell. The two important exceptions in rightward move is

1. If $i = n$ then $i+1^{st}$ cell hold a blank. Thus
$X_1 X_2 \cdots X_{n-1} q X_n \vdash_M X_1 X_2 \cdots X_{n-1} Y p B$
2. If $i = 1$ and $Y = B$. then the symbol B is written over X_1. Thus $q X_1 X_2 \cdots X_n \vdash_M p X_2 \cdots X_n$

Consider the example:-

Design a TM which recognizes the language $L = \{0^n 1^n / n \geq 1\}$

The TM will accept equal number of zeros followed by equal number of one's by changing all zeros to x and all ones to y until all 0's and 1's

have been matched.

Starting at the left end of the input, it enters a loop in which it changes a 0 to an X and moves to the right over whatever 0's and Y's it sees, until it comes to a 1. It changes the 1 to a Y. and moves left over Y's and 0's until it finds an X. At that point it looks for a 0 immediately to the right and if it finds one, changes it to X and repeat the process, changing a matching 1 to a Y.

The formal specification of the TM M is

$M = (\{q_0, q_1, q_2, q_3, q_4\}, \{0,1\}, \{0,1,X,Y,B\}, \delta, q_0, B, \{q_4\})$

where δ is given by

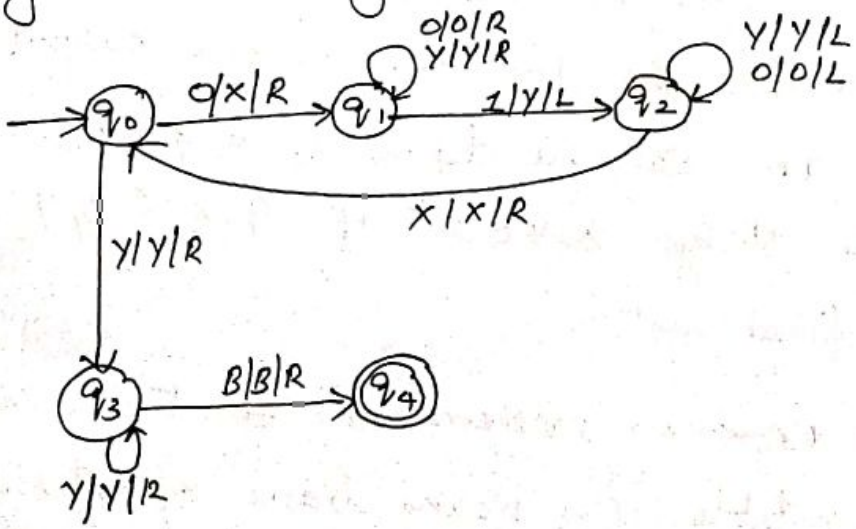
$\delta(q_0,0) = (q_1, X, R)$
$\delta(q_0,Y) = (q_3, Y, R)$
$\delta(q_1,0) = (q_1, 0, R)$
$\delta(q_1,1) = (q_2, Y, L)$
$\delta(q_1,Y) = (q_1, Y, R)$
$\delta(q_2,0) = (q_2, 0, L)$
$\delta(q_2,X) = (q_0, X, R)$
$\delta(q_2,Y) = (q_2, Y, L)$
$\delta(q_3,Y) = (q_3, Y, R)$
$\delta(q_3,B) = (q_4, B, R)$

The ID of the TM for the input 0011

The initial ID is $q_0 0011$. The entire sequence of moves of M is given as

$$
\begin{aligned}
q_0 0011 &\vdash X q_1 011 \\
&\vdash X 0 q_1 11 \\
&\vdash X q_2 0 Y 1 \\
&\vdash q_2 X 0 Y 1 \\
&\vdash X q_0 0 Y 1 \\
&\vdash X X q_1 Y 1 \\
&\vdash X X Y q_1 1 \\
&\vdash X X q_2 Y Y \\
&\vdash X q_2 X Y Y \\
&\vdash X X q_0 Y Y \\
&\vdash X X Y q_3 Y \\
&\vdash X X Y Y q_3 B \\
&\vdash X X Y Y B q_4 B
\end{aligned}
$$

The transition diagram of the TM which accepts language $0^n 1^n$ is given below



TM as language acceptors

Let $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ be a Turing machine. Then $L(M)$ is the set of strings w in Σ^* such that $q_0 w \vdash^* \alpha p \beta$ for some state p in F and any tape strings α and β.

The set of languages we can accept using a TM is often called the Recursively Enumerable languages or RE languages.

TM as Transducers

A TM can function as a transducer. That is a string is given as input to TM, it produces some output

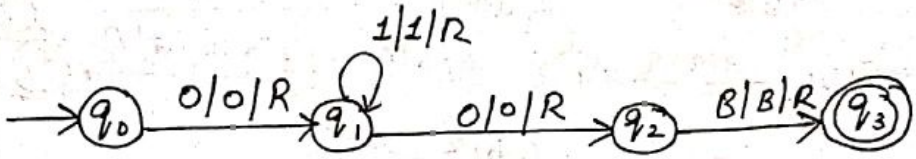
Input to the computation is a set of symbols on the tape. At the end of computation, whatever remains on the tape is the output.

A TM can be defines as a function $y = f(x)$ for the strings $x, y \in \Sigma^*$ if $q_0 x \vdash^* q_f y$ where q_f a final state.

All common mathematical functions are turing computable. This means basic operations such as addition, subtraction, multiplication, division can be performed on it. That is TM is an abstract model of the computer system.

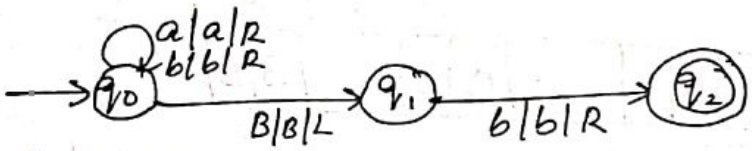
Designing Turing Machines

Q Construct a TM that recognizes the language 01^*0

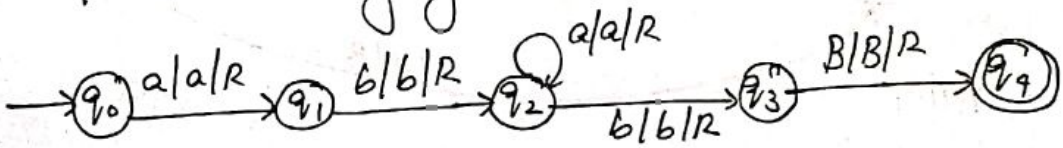


Q Construct a TM of all string over {a,b} which ends with b

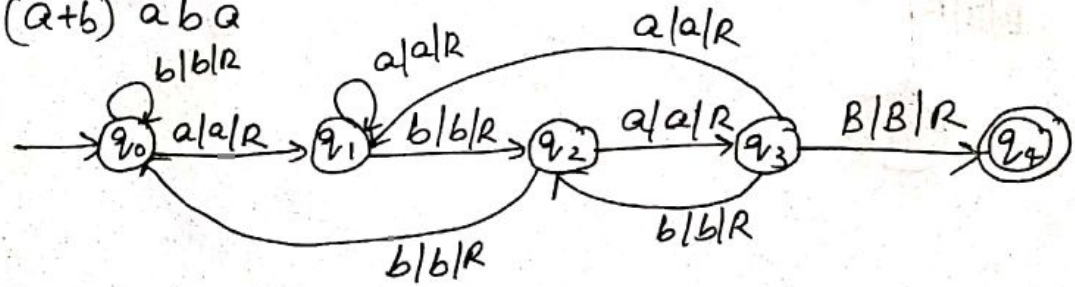
$L = \{ b, ab, aab, bbb, abbab \cdots \}$



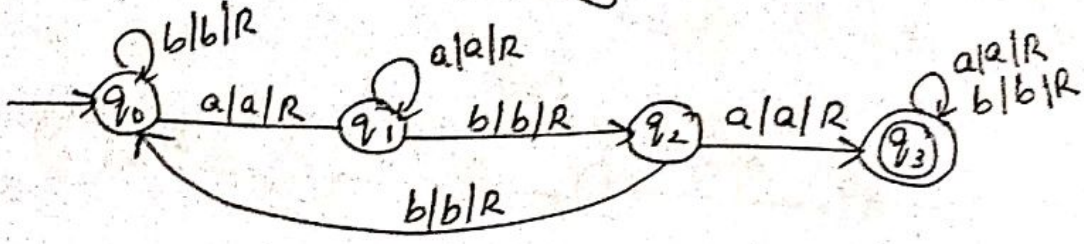
Q Construct a TM of all strings over {a,b} which accepts the language aba^*b



Q Construct a TM to accept the language $(a+b)^*aba$



Q Construct a TM to accept Strings over {a,b} in which substring aba is present

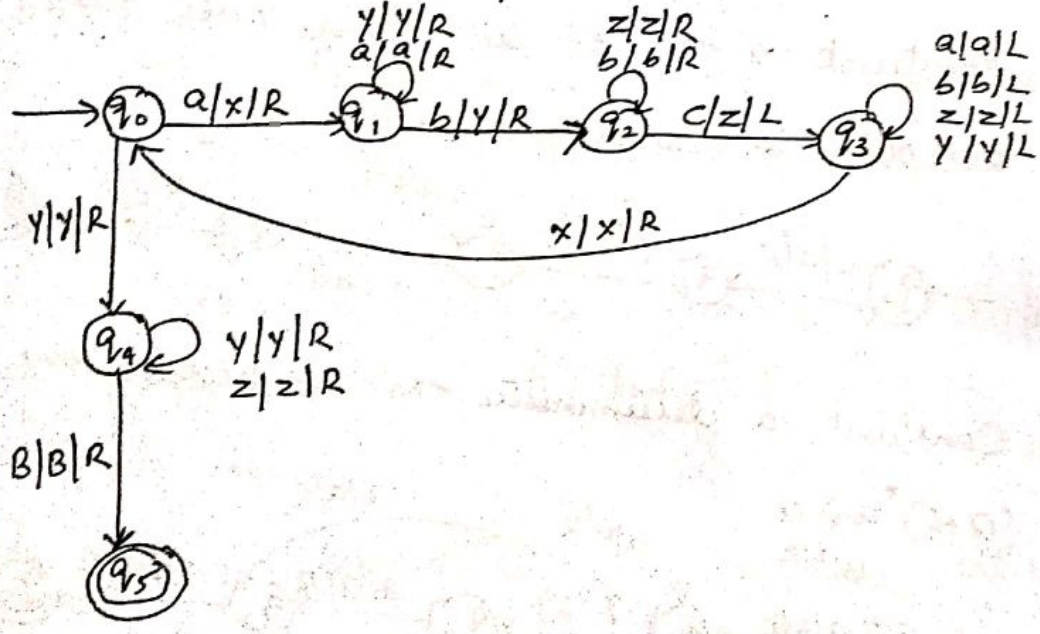


Q Construct a TM that accept the language $a^n b^n c^n \mid n \geq 1$ over $\Sigma = \{a, b, c\}$

Let $L = \{abc, aabbcc - - -\}$

a|a| - - |b|b| - - - |c|c| ... |B|B| - - - -

Each a, b and c is replaced with X, Y and Z



MOD - VI

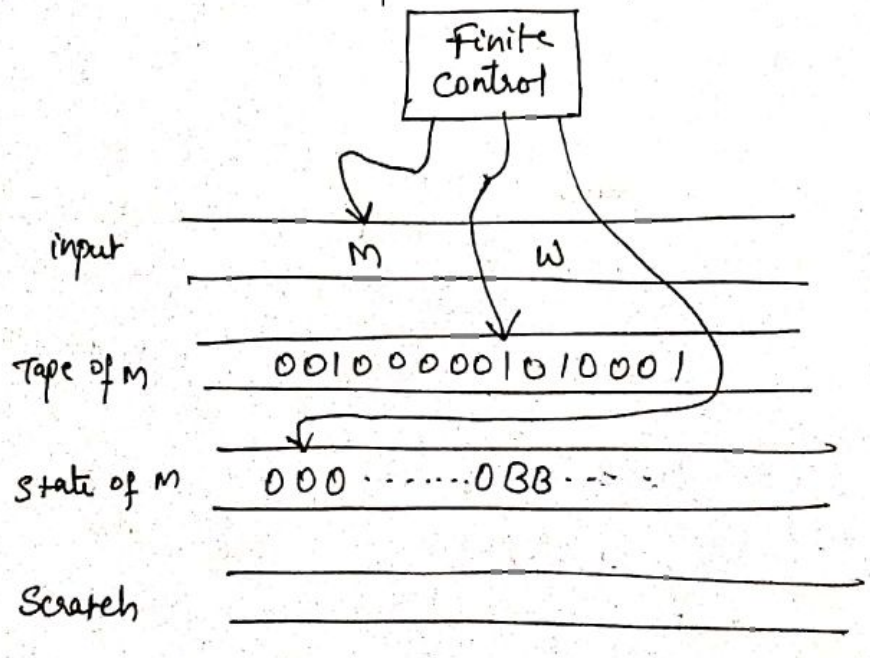
Variants of TMs

Universal Turing Machine

A Universal Turing Machine, UTM that can feed an input composed of a pair (M, w), where M is a TM with the binary input alphabet and 'w' is the input string.

The UTM can be described interms of multitape TM. The transitions of M are stored initially on the first tape along with the string w. A second tape will be used to hold the simulated tape of M, using the same format as for the code of M. That is tape symbol X_i of M will be Represented by 0^i and tape symbols will be Seperated by single 1's.

The third tape of UTM holds the state of M, with state q_i Represented by i 0's



The operation of UTM can be summarized as follows.

1. Examine the input to make sure that the code for M is a valid code for some TM. If not the UTM halts without accepting. Since invalid codes are assumed to represent the TM with no moves and such a TM accepts no inputs.

2. Initialize the second tape to contain the input w in its encoded form. ie for each 0 of w, place '10' on the second tape and for each 1 of w place '100' then. All cells beyond those used for w will hold the blank of UTM.

3. Place 0, the start state of M on the third tape and move the head of UTM's second tape to the first simulated cell.

4. To simulate a move of M, UTM searches on its first tape for a transition $0^i 1 0^j 1 0^k 1 0^l$ such that 0^i is the state on the tape 3, and 0^j is the tape symbol of M that begins at the position on tape 2 which is scanned by UTM.

For each transition of M, UTM should

i) change the contents of tape 3 to 0^k, ie Simulate the state change of M.

ii) Change the tape symbol of M ie Replace 0^j on tape 2 by 0^l

iii) Move the head of tape 2 one cell Right or left.

5. If M has no transitions that matches the simulated state and tape symbol then no transitions will take places. Thus M halts and UTM also halts for the given input

6. If M enters its accepting state, then UTM accepts the input string.

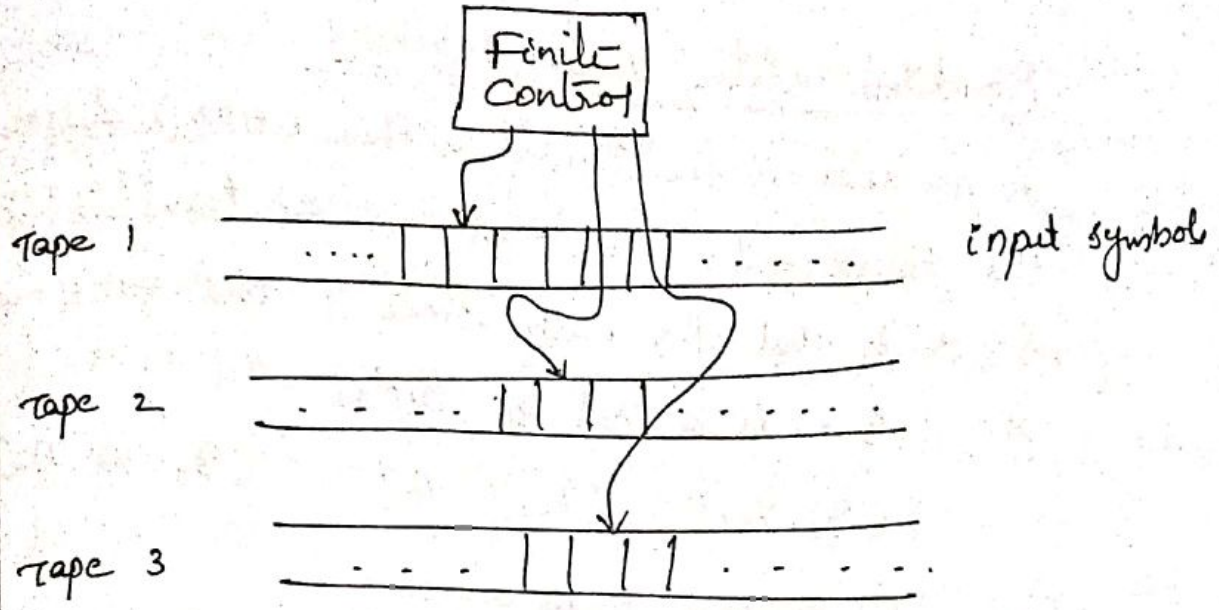
In this above manner UTM simulates M on w ie UTM accepts the pair (M, w) if f M accepts w.

Multitape Turing Machine

A multitape Turing Machine has a finite control (states) and some finite number of tapes. Each tape is divided into cells and each cell can hold any symbol of the finite tape alphabet. The set of tape symbols includes a blank and the input symbols. The set of states includes initial and accepting states.

Initially.

1. The input symbols is placed on the first tape
2. All other cells of all the tapes hold the blank
3. The finite control is in the initial state.
4. The head of the first tape is at the left end of the input.
5. All other tape heads at at some arbitrary cell. Since all cells are blank it doesn't matter where the head is placed initially.



A move on the multitape TM depends on the state and the symbol scanned by each of the tape heads.

In one move, the multitape TM does the following

1. The control enters a new state, which could be same as the previous state.
2. On each tape a new tape symbol is written on the cell scanned. Any of these symbol may be same as the symbol previously there.
3. Each of the tape heads makes a move, which can be either left, right or stationary. The heads move independently, so different heads may move in different directions and some may not move at all.

As like one-tape TM, multitape Turing Machine accept the string by entering an accepting state.

Nondeterministic Turing Machine

A nondeterministic Turing Machine (NTM) differs from the deterministic in terms of the transition function δ such that for each state q and tape symbol X, $\delta(q, X)$ is a set of triples

$$\{(q_1, Y_1, D_1), (q_2, Y_2, D_2), \ldots, (q_k, Y_k, D_k)\}$$

where k is any finite integer.

At each step NTM can choose any triples for the next move. It cannot pick a state from one triple, a tape symbol from another and the direction from yet another triples.

The NTM, M accepts an input w if there is any sequence of choices of moves that leads from the initial ID with 'w' as input to an ID with an accepting state.

Enumeration Machine

An enumeration TM is a special kind of TM which prints out a set of strings as output.

An enumerator TM prints the strings of the language one at a time. If w is the string in the language, the enumerator prints w.

Enumeration machine M has a special non halting output state. whenever M enters into the output

states, the contents of the tape are output. Enumeration TM, M runs forever outputting strings. Let L(M) be the language accepted by enumeration TM then L(M) can be given as

$L(M) = \{w : w$ is eventually output by $M\}$

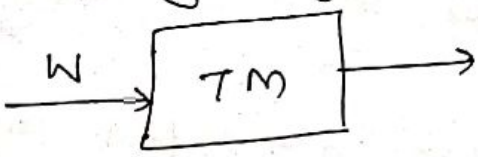
A TM enumerates a language L if ~~the~~ list the elements of L. It is convenient to allow a multitape machine with one tape functions as an output tape.

Let T be the k-tape TM and $L \subseteq \Sigma^*$. T enumerates L if it operates with satisfying the following condition

1. The tape head on the first tape never moves to the left and no nonblank symbol printed on tape 1 is modified or erased.
2. For every $x \in L$ there is some point in the operation of TM T when tape 1 has contents $x_1 \# x_2 \# x_3 \# \ldots x_n$ for some $n \geq 0$. where the strings $x_1, x_2, \ldots x_n$. $x \in L$. If L is finite then nothing is printed after the # following the last element of L
If L is infinite the T continues to move for ever.

Recursively Enumerable Languages (Type-0 Language)

RE Languages or Type-0 ~~ge~~ languages are generated by type-0 Grammars. An RE language can be accepted or Recognized by the TM which means it will enter into final state for the strings of language and may or may not enter into Rejecting states for the strings which are not part of the language. It means TM can loop forever for the string which are not a part of the language. RE is also called as turing Recognizable languages.



If $w \in L$ then TM accept
If $w \notin L$ then TM enters nonaccepting state or loops for ever.

Recursive Languages

A Language L is said to be Recursive if L is accepted by some Turing Machine M ie $L = L(M)$ such that

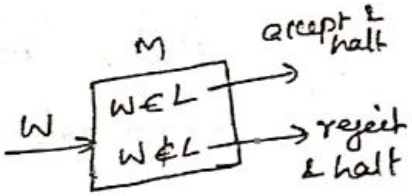
1. If w is in L, then M accepts and halts
2. If w is not in L, then M enters nonaccepting state and halts.

A TM of this type of languages always have an algorithm that have well defined sequence

of steps that always finishes and produces an answer. Recursive Languages are also called as Turing decidable Languages.

for eg:- Let $L = a^n b^n c^n / n \geq 1$ is Recursive language because we can construct a TM which will move to final state if the string is of the form $a^n b^n c^n$ else move to non-final state so TM will always halts.

The Relationship between RE and Recursive languages are given below



Note:-
Every Recursive language is Recursively enumerable Languages.

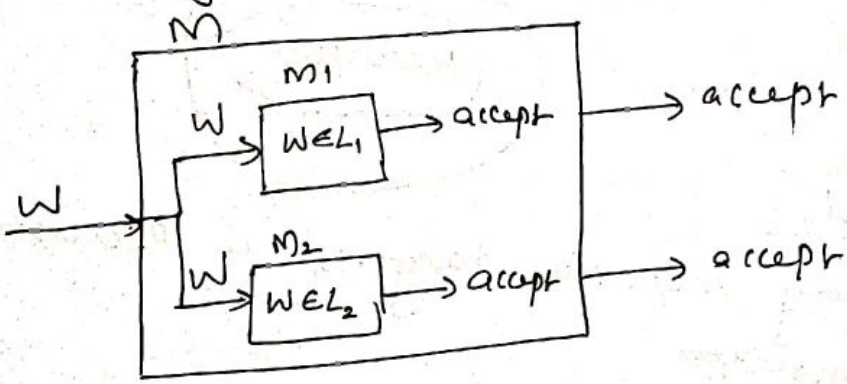
Propeties of Recursively Enumerable Languages:

1. Union of two Recursively Enumerable is Recursively Enumerable.

proof:-
Let L_1 and L_2 be two Recursively enumerable languages accepted by Turing Machine M_1 and M_2 respectively. Then $L_1 \cup L_2$ is accepted by Turing machine M, where $X = W_1 \cup W_2$ for $W_1 \in L_1$,

and $W_2 \in L_2$. To determine if M_1 or M_2 accepts X, we run both M_1 and M_2 at once using a two tape Turing Machine M.

M first simulates M_1 on the first tape and then M_2 on second tape. If ~~enter~~ either one enter the final state, the input is accepted by Turing Machine M. Otherwise it continues to loop for ever or enters into non accepting states. Hence it shows that $L_1 \cup L_2$ is Recursively enumerable language.



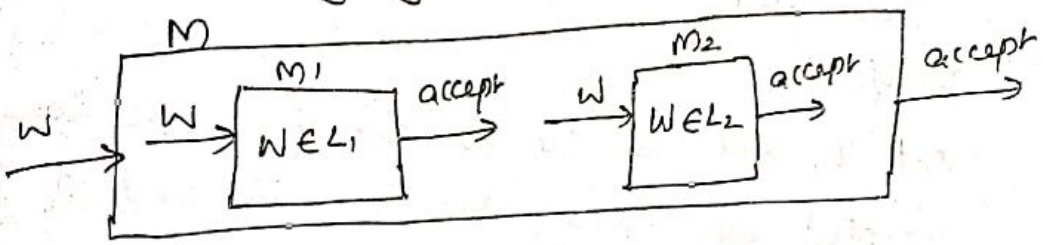
2. Intersection of two Recursive enumerable language is also a Recursive enumerable

Proof:-

Let L_1 and L_2 be two Recursively enumerable languages accepted by Turing Machine M_1 and M_2 respectively. Then $L_1 \cap L_2$ is accepted by Turing Machine M, where $X = W_1 \cap W_2$ for $W_1 \in L_1$ and $W_2 \in L_2$. To determine if M_1 and M_2 accepts X, we run both M_1 and M_2 at once using a two tape Turing

Machine M.

M first simulates M_1 on the first tape and then M_2 on second tape. If both M_1 and M_2 enters the final state, the input is accepted by Turing Machine M. Otherwise it continues to loop for ever or enters into non accepting state. Hence it shows that $L_1 \cap L_2$ is Recursively enumerable language.



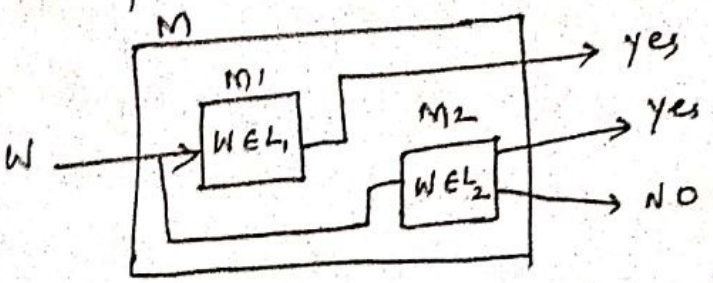
Properties of Recursive Language

1. Union of two Recursive language is Recursive

Proof:

Let L_1 and L_2 be the two Recursive languages accepted by Turing Machines M_1 and M_2 respectively Construct a TM M, which first acts on M_1. If M_1 accepts the input then M also accepts it. but if M_1 rejects then M moves on to M_2 and simulates it. If M_2 accepts the input string then M accepts it. Since both M_1 and M_2 are the algorithms which produce an answer M surely halts. Thus $L_1 \cup L_2$ is accepted by TM M, where $x = w_1 \cup w_2$ for $w_1 \in L_1$ and $w_2 \in L_2$

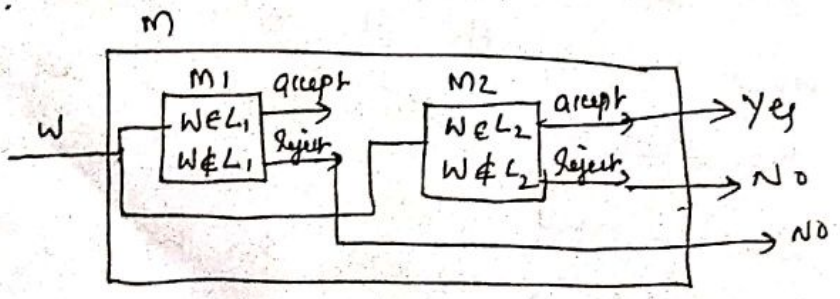
Hence $L_1 \cup L_2$ is also recursive since a TM M exists for it.



2. Intersection of two recursive language is recursive.

Proof:

Let L_1 and L_2 be the two recursive languages accepted by Turing Machine M_1 and M_2 respectively. Construct a TM M, which first acts on M_1. If M_1 accepts the input then the Turing Machine M simulates M_2 on the same input. If both the TM M_1 and M_2 accepts the string then M accepts it. Since both M_1 and M_2 are the algorithms which produce an answer & surely halts. Thus $L_1 \cap L_2$ is accepted by TM M, where $x = w_1 \cap w_2$ where $w_1 \in L_1$ and $w_2 \in L_2$. Hence $L_1 \cap L_2$ is also recursive since a TM M exists for it.

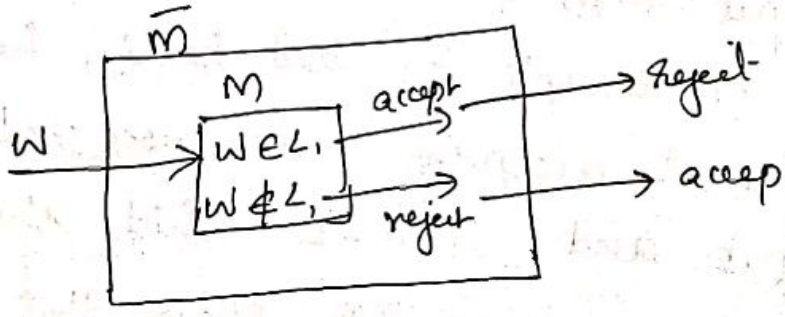


3. The complement of a Recursive language is also Recursive.

proof:-

Let L be a Recursive language and $\bar{L}$ be its complement. Also let M be a TM that halts on all inputs and accepts L. Here the accepting states of M are made nonaccepting states of $\bar{M}$ with no transitions. ie here $\bar{M}$ halts without accepting.

If s is new accepting state in $\bar{M}$, then there is no transition from this state. Also if L is Recursive, then $L = L(M)$ for some TM M, which always halts. Then M is transformed into $\bar{M}$ so that $\bar{M}$ accepts when M does not accept input and vice versa. So $\bar{M}$ always halts and accepts $\bar{L}$. Hence $\bar{L}$ is Recursive. It shows that the complement of L is also Recursive.



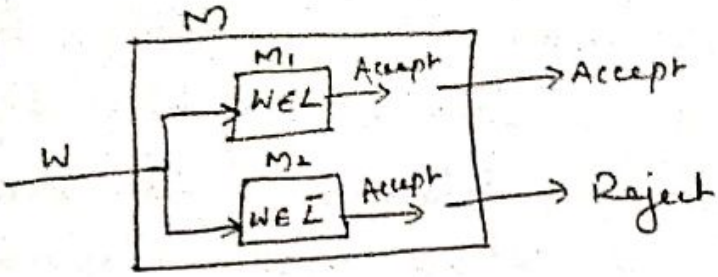
Theorem:- 4

If a Language L and its complement are both Recursively Enumerable then L is Recursive

OR

A Language L is recursive if and only if both Language and its complement are recursively enumerable.

Proof:-
Let $L = L(M_1)$ and $\bar{L} = L(M_2)$ is the language L is accepted by TM M_1 and its complement $\bar{L}$ is accepted by TM M_2. Both M_1 and M_2 are simulated in parallel by a TM M. We can make M a two-tape TM, and then convert it to a one-tape TM, to make the simulation easy. One tape of M simulates M_1, while the other tape of M simulates the tape of M_2. The states of M_1 and M_2 are each components of the state of M. If the input to M is in L then M_1 accepts it and halts, hence M accepts it and halts. If input to M is not in L, then it is in $\bar{L}$ so M_2 accepts it and halts, hence M halts without accepting. Hence M halts on every input and $L(M) = L$ which shows that it is recursive. Hence it shows that if both L and $\bar{L}$ are recursively enumerable then L is recursive.



Decidability and Halting problem

A problem is said to be decidable if there exists an algorithm which gives correct answer yes or NO.

A problem is said to be a halting problem if there is no algorithm exists which gives the solution yes or no to the given problem

eg: The problem of recognizing the same language by two DFA is decidable.

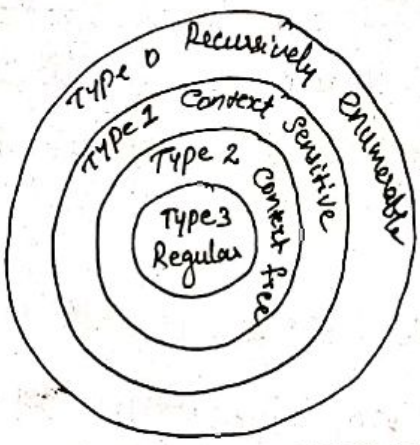
The problem of generating same language by Context Free grammar is undecidable (Halting)

The problem of deciding whether the TM reach the Halt state for some initial tape is Halting problem

Chomsky Hierarchy

According to chomsky Hierarchy the classes of languages are classified into Regular, Context free, Context sensitive and recursively enumerable. and there after chomsky classified the grammar into

four types in terms of productions as type 3, type 2, type 1 and type 0. Each level of hierarchy can be characterized by a class of grammars and by a certain type of abstract machine or model of computation



Class	Grammars	Languages	Automata
Type - 3	Regular	Regular	Finite Automata (NFA, DFA, ε-NFA)
Type - 2	Context Free	Context Free	pushdown Automata
Type - 1	Context sensitive	Context sensitive	Linear bounded Automata
Type - 0	Unrestricted	Recursively enumerable	Turing Machine

Type 3: The languages defined by type-3 grammars are accepted by finite state automata.

Rules are of the form $A \longrightarrow \epsilon$
$A \longrightarrow \alpha$
$A \longrightarrow \alpha B$

where $A, B \in N$ and $\alpha \in \Sigma$.

Type 2: Languages defined by Type-2 grammars are accepted by push down automata. Natural language is almost definable by type-2 tree structures. Rules are of the form

$A \rightarrow \alpha$ where $A \in N$ and $\alpha \in (N \cup \Sigma)^*$

Type 1: Languages defined by Type-1 grammars are accepted by linear bounded automata. Rules of the type-1 grammar are of the form

$$\alpha A \beta \rightarrow \alpha B \beta$$
$$S \rightarrow \in$$

where $A, S \in N$ and $\alpha, \beta, B \in (N \cup \Sigma)^*$

and $B \neq \in$

Type 0: Languages defined by Type-0 grammars are accepted by Turing Machines. Rules are of the form $\alpha \rightarrow \beta$, where α and β are strings over a Vocabulary 'V' and $\alpha \neq \in$

Classes And Applications Of Automata

Automata theory is the study of abstract machines and automata, as well as the computational problems that can be solved using them. It is a theory in theoretical computer science. The word *automata* (the plural of *automaton*) comes from the Greek word αὐτόματα, which means "self-making". An automaton (Automata in plural) is an abstract self-propelled computing device which follows a predetermined sequence of operations automatically. An automaton with a finite number of states is called a Finite Automaton (FA) or Finite State Machine (FSM).

The figure at right illustrates a finite-state machine, which belongs to a well-known type of automaton. This automaton consists of states (represented in the figure by circles) and transitions (represented by arrows). As the automaton sees a symbol of input, it makes a transition (or jump) to another state, according to its transition function, which takes the previous state and current input symbol as its arguments.

Automata theory is closely related to formal language theory. In this context, automata are used as finite representations of formal languages that may be infinite. Automata are often classified by the class of formal languages they can recognize, as in the Chomsky hierarchy, which describes a nesting relationship between major classes of automata. Automata play a major role in theory of computation, compiler construction, artificial intelligence, parsing and formal verification.

Automata

What follows is a general definition of automaton, which restricts a broader definition of system to one viewed as acting in discrete time-steps, with its state behavior and outputs defined at each step by unchanging functions of only its state and input.[5]

Informal description

An automaton *runs* when it is given some sequence of *inputs* in discrete (individual) *time steps* or steps. An automaton processes one input picked from a set of *symbols* or *letters*, which is called an *input alphabet*. The symbols received by the automaton as input at any step are a sequence of symbols called *words*. An automaton has a set of *states*. At each moment during a run of the automaton, the automaton is *in* one of its states. When the automaton receives new input it moves to another state (or *transitions*) based on a *transition function* that takes the previous state and current input symbol as parameters. At the same time, another function called the *output function* produces symbols from the *output alphabet*, also according to the previous state and current input symbol. The automaton reads the symbols of the input word and transitions between states until the word is read completely, if it is finite in length, at which point the automaton *halts*. A state at which the automaton halts is called the *final state*.

To investigate the possible state/input/output sequences in an automaton using formal language theory, a machine can be assigned a *starting state* and a set of *accepting states*. Then, depending on whether a run starting from the starting state ends in an accepting state, the automaton can be said to *accept* or *reject* an input sequence. The set of all the words accepted by an automaton is

called the *language recognized by the automaton*. A familiar example of a machine recognizing a language is an electronic lock which accepts or rejects attempts to enter the correct code.

Classes of automata

The following is an incomplete list of types of automata.

Discrete, continuous, and hybrid automata

Normally automata theory describes the states of abstract machines but there are discrete automata, analog automata or continuous automata, or hybrid discrete-continuous automata, which use digital data, analog data or continuous time, or digital and analog data, respectively.

Applications

Each model in automata theory plays important roles in several applied areas. Finite automata are used in text processing, compilers, and hardware design. Context-free grammar (CFGs) are used in programming languages and artificial intelligence. Originally, CFGs were used in the study of the human languages. Cellular automata are used in the field of artificial life, the most famous example being John Conway's Game of Life. Some other examples which could be explained using automata theory in biology include mollusk and pine cones growth and pigmentation patterns. Going further, a theory suggesting that the whole universe is computed by some sort of a discrete automaton, is advocated by some scientists. The idea originated in the work of Konrad Zuse, and was popularized in America by Edward Fredkin. Automata also appear in the theory of finite fields: the set of irreducible polynomials which can be written as composition of degree two polynomials is in fact a regular language.[15] Another problem for which automata can be used is the induction of regular languages.

Automata simulators

Automata simulators are pedagogical tools used to teach, learn and research automata theory. An automata simulator takes as input the description of an automaton and then simulates its working for an arbitrary input string. The description of the automaton can be entered in several ways. An automaton can be defined in a symbolic language or its specification may be entered in a predesigned form or its transition diagram may be drawn by clicking and dragging the mouse. Well known automata simulators include Turing's World, JFLAP, VAS, TAGS and SimStudio.[16]

Connection to category theory

One can define several distinct categories of automata[17] following the automata classification into different types described in the previous section. The mathematical category of deterministic automata, sequential machines or *sequential automata*, and Turing machines with automata homomorphisms defining the arrows between automata is a Cartesian closed category,[18][19] it has both categorical limits and colimits. An automata homomorphism maps a quintuple of an

automaton A_i onto the quintuple of another automaton A_j.[20] Automata homomorphisms can also be considered as *automata transformations* or as semigroup homomorphisms, when the state space, ***S***, of the automaton is defined as a semigroup $\mathbf{S}_g$. Monoids are also considered as a suitable setting for automata in monoidal categories.[21][22][23]

Categories of variable automata

One could also define a *variable automaton*, in the sense of Norbert Wiener in his book on *The*

Human Use of Human Beings via the endomorphisms . Then, one can show that such variable automata homomorphisms form a mathematical group. In the case of non-deterministic, or other complex kinds of automata, the latter set of endomorphisms may become, however, a *variable automaton groupoid*. Therefore, in the most general case, categories of variable automata of any kind are categories of groupoids or groupoid categories. Moreover, the category of reversible automata is then a 2-category, and also a subcategory of the 2-category of groupoids, or the groupoid category.

Applications of various Automata

Automata is a machine that can accept the Strings of a *Language L* over an *input alphabet* .
So far we are familiar with the Types of Automata . Now, let us discuss the expressive power of Automata and further understand its Applications.

Expressive Power of various Automata:
The Expressive Power of any machine can be determined from the class or set of Languages accepted by that particular type of Machine. Here is the increasing sequence of expressive power of machines :

As we can observe that FA is less powerful than any other machine. It is important to note that DFA and NFA are of same power because every NFA can be converted into DFA and every DFA can be converted into NFA .
The Turing Machine i.e. TM is more powerful than any other machine.

(i) Finite Automata (FA) equivalence:

```
Finite Automata
≡ PDA with finite Stack
≡ TM with finite tape
≡ TM with unidirectional tape
≡ TM with read only tape
```

(ii) Pushdown Automata (PDA) equivalence:

```
PDA ≡ Finite Automata with Stack
```

(iii) Turing Machine (TM) equivalence:

```
Turing Machine
≡ PDA with additional Stack
≡ FA with 2 Stacks
```

The **Applications** of these Automata are given as follows:

1. Finite Automata (FA) –

- For the designing of lexical analysis of a compiler.
- For recognizing the pattern using regular expressions.
- For the designing of the combination and sequential circuits using Mealy and Moore Machines.
- Used in text editors.
- For the implementation of spell checkers.

2. Push Down Automata (PDA) –

- For designing the parsing phase of a compiler (Syntax Analysis).
- For implementation of stack applications.
- For evaluating the arithmetic expressions.
- For solving the Tower of Hanoi Problem.

3. Linear Bounded Automata (LBA) –

- For implementation of genetic programming.
- For constructing syntactic parse trees for semantic analysis of the compiler.

4. Turing Machine (TM) –

- For solving any recursively enumerable problem.
- For understanding complexity theory.
- For implementation of neural networks.
- For implementation of Robotics Applications.
- For implementation of artificial intelligence.