DrillBit

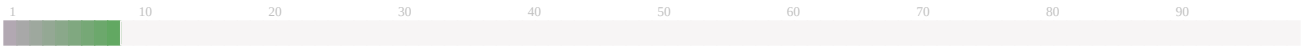
The Report is Generated by DrillBit Plagiarism Detection Software

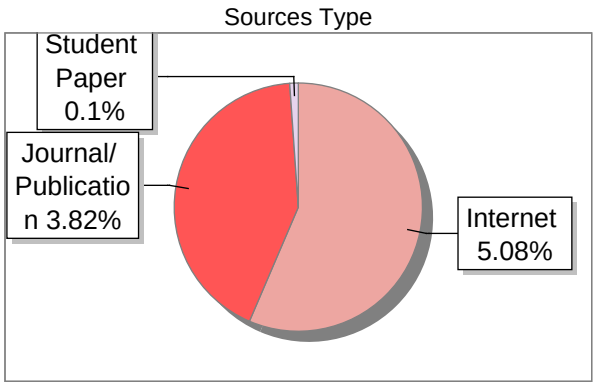
Submission Information

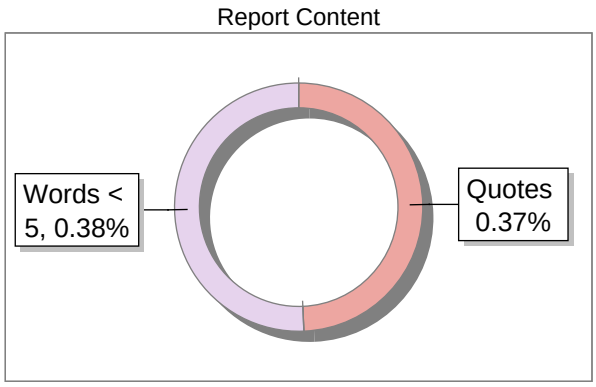
Author Name	AMAL CHAKRAVORTY
Title	Theory of Computation
Paper/Submission ID	3027210
Submitted by	librarian.adbu@gmail.com
Submission Date	2025-01-24 15:52:49
Total Pages, Total Words	65, 12514
Document type	Others

Result Information

Similarity **9 %**







Exclude Information

Quotes	Excluded
References/Bibliography	Excluded
Source: Excluded < 5 Words	Excluded
Excluded Source	**0 %**
Excluded Phrases	Not Excluded

Database Selection

Language	English
Student Papers	Yes
Journals & publishers	Yes
Internet or Web	Yes
Institution Repository	Yes

A Unique QR Code use to View/Download/Share Pdf File

DrillBit

DrillBit Similarity Report

9	39	A
SIMILARITY %	MATCHED SOURCES	GRADE

A-Satisfactory (0-10%)
B-Upgrade (11-40%)
C-Poor (41-60%)
D-Unacceptable (61-100%)

LOCATION	MATCHED DOMAIN	%	SOURCE TYPE
1	qdoc.tips	1	Internet Data
2	www.geeksforgeeks.org	1	Internet Data
3	Towards a programming language based on the notion of two-level grammar by Ja-1983	1	Publication
4	redcol.minciencias.gov.co	<1	Publication
5	docplayer.gr	<1	Internet Data
6	www.jacsicoe.in	<1	Publication
7	en.wikipedia.org	<1	Internet Data
8	pdfcookie.com	<1	Internet Data
9	Efficient learning of multiple context-free languages with multidimens by Ry-2011	<1	Publication
10	moam.info	<1	Internet Data
11	moam.info	<1	Internet Data
12	lmcs.episciences.org	<1	Publication
13	On the spectra of universal relational sentences by Harve-1984	<1	Publication

14	pdfcookie.com	<1	Internet Data
15	index-of.es	<1	Publication
16	moam.info	<1	Internet Data
17	Modeling and Verifying Concurrent Programs with Finite Chu Spaces by Xu-Ta-2010	<1	Publication
18	sist.sathyabama.ac.in	<1	Publication
19	www.geeksforgeeks.org	<1	Internet Data
20	Decentralized fault diagnosis approach without a global model for fault diagnosi by Sayed-Mouchaweh-2015	<1	Publication
21	dochero.tips	<1	Internet Data
22	qdoc.tips	<1	Internet Data
23	research.ijcaonline.org	<1	Publication
24	AN APPROACH TO CHECKING THE COMPLIANCE OF USER PERMISSION POLICY IN S, by TRUONG, NINH-THUAN- 2013	<1	Publication
25	www.biomedcentral.com	<1	Publication
26	www.pnas.org	<1	Publication
27	artsdocbox.com	<1	Internet Data
28	moam.info	<1	Internet Data
29	Urbanization and the Organization of Animal Production at Tell Jemmeh, by Wapnish, Paula Hes- 1988	<1	Publication
30	combinatorics.org	<1	Publication
31	fdokumen.id	<1	Internet Data

32	COMPARISON AND ANALYSIS OF PREDICTIVE ALGORITHMS FOR THE STOCK MARKET BY RISHI MAHESHWARI Yr-2021 SUBMITTED TO JNTUH COLLEGE OF ENG	<1	Student Paper
33	dochero.tips	<1	Internet Data
34	Lecture Notes in Computer Science Conformal and Probabilistic Predi, by Gammerman, Alexande- 2016	<1	Publication
35	Thesis Submitted to Shodhganga Repository	<1	Publication
36	www.mdpi.com	<1	Internet Data
37	feng.stafpu.bu.edu.eg	<1	Publication
38	learning-python.com	<1	Internet Data
39	towardsdatascience.com	<1	Internet Data

MODULE I- Theory of Automata (**Unit-1-** Introduction to Automata)

1.0. Introduction:

The Theory of Computation is a branch of theoretical computer science that deals with the study of algorithms and computational complexity. It aims to answer the fundamental questions about :

- What can be computed
- How efficiently the computation can be done
- And what limitations exist in respect of computational power

Now a days, machines (digital, analog, mechanical) play a very important role in the development of human's day to day activities. So, we need some mechanism (i.e., languages) to communicate with the machines. We need a language for communication with machines. But we do not require natural language (ex: English, France etc.) as these languages are too complex and on the other hand machine interaction require very fewer complex languages compared to natural languages. If the machine is simple, the language will be simple and if the machine is complex, the language will be complex.

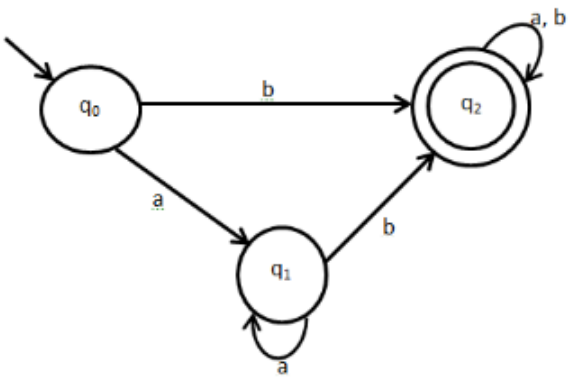
Machine languages are of two types: formal and informal language. Within the scope of the topic, we only discuss the formal language. According to Oxford dictionary: Language is the system of spoken or written communication used by a particular country, people, community, etc., typically consisting of words used within a regular grammatical and syntactic structure; (also) a formal system of communication.

In natural language, we define the list of words in a dictionary because they are finite and predefined, but we cannot list all the sentences which can be formed using these words as they are infinite. So, we a have a mechanism called grammer/rules using which we can decide which sentence is valid or which sentence is invalid.

1.1.Definition of an Automaton:

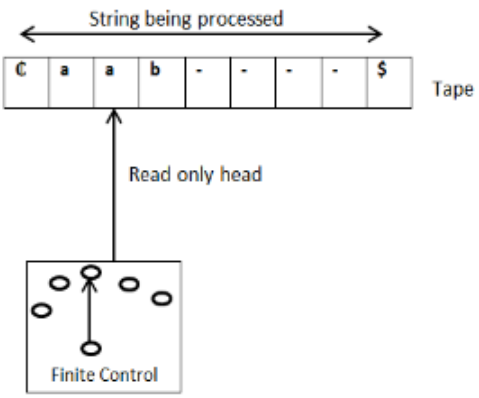
An automaton (pl.: automata) is a self-operating machine or control mechanism designed to automatically follow a sequence of operations, or respond to predetermined instructions. The word automaton is the latinization of the Ancient Greek automaton, which means "acting of one's own will". It was first used by Homer to describe an automatic door opening.

It is used to recognize patterns in strings of symbols. Finite automata have varieties of applications in computer world like lexical analysis, pattern matching. The figure given below is a finite automaton with three states.



This type of automata has no temporary storage, as it is severely limited in its capacity to remember things during computation.

Block Diagram of a finite Automaton:



The various components of finite automaton are shown above and explained below:

Input tape-

- The end block of the tape contains the end markers ¢ at the left end and the end marker $ at the right end.
- The absence of end markers indicates that the tape is of infinite length.
- The left-to-right sequence of symbols between the two end markers in the input string to be processed.

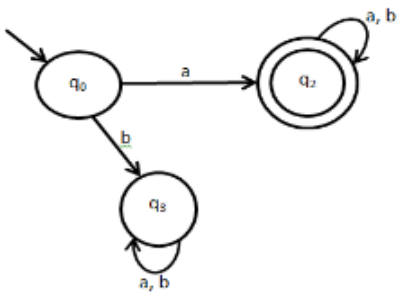
Read only Head-

- The tape has a read only head which examines one block at a time and can move one block either to the left or to the right side.
- At the beginning of the operation the head is always at the leftmost block of the input tape.
- Then the machine restricts the moment of the read only head from left to right direction only and one block every time when it reads symbols.

Finite control-

- There is a finite control that determines the state of the automaton and also controls the movement of the head.
- The input of finite control is usually the symbol under the read head (assumed a) and the present state of the machine (assumed q) to give the following outputs;
- A motion of read head along the tape to the next block.
- The next state of the finite state machine given by δ (q, a).

The state diagram of a finite automaton is shown below:



Here q_0 is the initial state and q_2 is the final state. On input symbol a, the machine changes state from q_0 to q_2 which is the final state. In state q_2 if symbol a or b is encountered, the machine goes into loop.On input symbol b, the machine changes state from initial state q_0 to a dead state q_3 .In q_3 , there is no way to move to the final state. Thus the above machine accepts only those strings which are generated by the language L and L= { a, aa, ab, aaa,aba,aab,ab,……..}. Finally we can conclude that this automaton or machine accepts the strings which are started with symbol 'a' and rejects the other strings.

1.3. Summary:

i) Theory of Computation is a branch which belongs to theoretical computer science that focuses on algorithms and computational complexity.

ii) Types of Machine Languages**:** Formal and Informal languages.

iii) Automaton is a self-operating machine designed to follow a sequence of operations or respond to predefined instructions.

iv) Finite Automaton is a theoretical machine with a finite set of states.

1.4. Check your progress:

1. Explain the main objectives of the Theory of Computation. What fundamental questions does it aim to answer?
2. Why are natural languages not suitable for machine communication, and how do machine languages differ in terms of complexity?
3. In the context of formal languages, what is the difference between a symbol, an alphabet, a string, and a language? Provide examples for each.
4. In the context of formal languages, what is the difference between a symbol, an alphabet, a string, and a language? Provide examples for each.
5. Design an automaton for the language L = {a, aa, , aaa, aaaa, …....}, where input symbol, Σ ={a,b}.
6. In the context of formal languages, what is the difference between a symbol, an alphabet, a string, and a language? Provide examples for each.

2.0.Introduction:

- Initial State (q_0): The state where the system starts when the computation begins.
- Final States (F): A subset of the states that represent the accepting or goal states.

The transition system(T) is formally defined as a 5-tuple: $T=(Q,\Sigma,\delta,q_0,F)$

Where:

- Q is a set where the elements are the states.
- Σ is the set of input symbols (alphabet).
- δ is the transition function.
- q_0 is the initial state.
- F is the set of final (or accepting) states.

Consider a simple transition system with:

- States $Q=\{q_0,q_1,q_2\}$
- Alphabet $\Sigma=\{0,1\}$
- Initial state = q_0
- Final state $F=\{q_2\}$

Transition function δ: $\delta(q_0,0)= q_1$

$\delta(q_1,1)= q_2$

$\delta(q_2,0)= q_2$

$\delta(q_2,1)= q_2$

This transition system accepts strings that reach the state q_2 by reading a sequence of 0s and 1s.

2.2.Properties of transition function: (Concept of totality, determinism and non-determinism)

The transition function δ plays a central role in defining the behavior of an automaton. It maps a state and an input symbol to a new state.

Formally, the transition function is defined as: $\delta:Q\times\Sigma\rightarrow Q$. This means that for every state $q\in Q$ and every symbol $a\in\Sigma$, there is a state $q'\in Q$ such that $\delta(q, a)=q'$.

In deterministic automata, the transition function is total (or complete) and deterministic. The word deterministic here means that for each state and input symbol, there is exactly one resulting state.

In non-deterministic automata, the transition function may be non-deterministic, allowing for multiple possible transitions for a given state and input symbol.

Deterministic vs. Non-Deterministic Transition Systems:

- Deterministic Transition Systems (DTS):
  - For each state and input symbol, there is exactly one possible transition.
- Non-Deterministic Transition Systems (NTTS):
  - For some state and input symbol, there may be multiple possible next states.

Transition Function in DFAs:

2.3. Summary:

i) **A** transition system is a formal model that describes the states and transitions of a system.

ii) A transition system is formally defined as a 5-tuple:$T = (Q, \Sigma, \delta, q_0, F)$

iii) The transition function δ defines the system's behavior by mapping a state and input symbol to a new state.

2.4.Check your progress:

1. Define the transition system. What are its components? Explain each with an example.
2. Define transition function and give an example of it.
3. Consider a DFA with states $\{q_0,q_1,q_2\}$, alphabet $\{0,1\}$, initial state q_0, and final state q_2. Define the transition function for a machine that accepts strings ending in "01".
4. Explain the difference between deterministic and non-deterministic transition systems. Give an example of each.
5. Given the following transition function for an NFA, trace the computation for the input string "0110".

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

6. Discuss the application of transition systems in modeling software systems. Provide an example of how it can be used to verify a system.

MODULE I- Theory of Automata (Unit 3: Acceptability of a string by a finite automaton)

1. Deterministic Finite Automata (DFA)
2. Non-Deterministic Finite Automata (NFA)

3.1. Concepts of symbol, alphabet, string, language:

Symbol: Symbols are basic building blocks, which can be any character/token (ex. Cat, mouse, any type of symbol etc.). In English language we called them as letters.

Alphabet: Alphabet is the finite non-empty set of symbols. Every language has its own alphabets.In theory of computation, the symbol Σ is used for depicting alphabet. e.x.: Σ = {0, 1}. For English Σ = {a, b, c,……z}. In English also alphabet is a set of letters; in general we called them as alphabet.

String: A string is a finite sequence of symbols. e.x.: Σ = {a, b}, String: aaabb, aa,bb,c and so on.(In English we call them as words)

Language: A language is a set of strings (In natural language (set of words (predefined) and grammar) we apply this model from words to sentence).

E.x.: Σ ={a,b}, and L is a language which is defined as L= { a, aa, ab, aaa, aba, aab, ab, ……..}

If $\Sigma = \{a, b\}$ then: $\Sigma^0 = \{ \varepsilon \}$

$\Sigma^1 = \{ a, b \}$

$\Sigma^2 = \Sigma . \Sigma = \{ a, b \} . \{ a, b \} = \{aa, ab, ba, bb\}$

Similarly, $\Sigma^3 = \{ aaa, aab, aba, abb, baa, bab, bba, bbb \}$

Thus, Σ^n defines the set of all the strings from the alphabet Σ of length exactly n.

$\Sigma^n = \{ W \mid |W|=n$; using the symbols from the alphabet Σ

We may have languages that have infinite number of words, so it is not possible for us to list them. We have to use some framework for this, which can somehow represent the same language. Using two methods a language is represented.

- By a grammar that generates a language.
- By a machine that accepts a language.

3.2. Kleene closure, positive closure :

Kleene closure: Given a set of symbols Σ, the notation Σ* represents the collection of strings formed by concatenating zero or more symbols from Σ, with strings of any length allowed. In general, it refers to any string composed solely of symbols from Σ.

$$\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \ldots\ldots\ldots\ldots \cup \Sigma^\infty$$

Positive closure: If Σ represents a set of symbols, then Σ^+ is used to denote the set of strings formed by concatenating one or more symbols from Σ, of any length. Essentially, it refers to any string of any length that consists solely of symbols from Σ.

$$\Sigma^+ = \Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \ldots\ldots\ldots\ldots \cup \Sigma^\infty$$

3.3. Acceptability of a string by a deterministic finite automaton:

A deterministic finite automaton (DFA) is defined by 5-tuple (Q, Σ, δ, S,F), where:

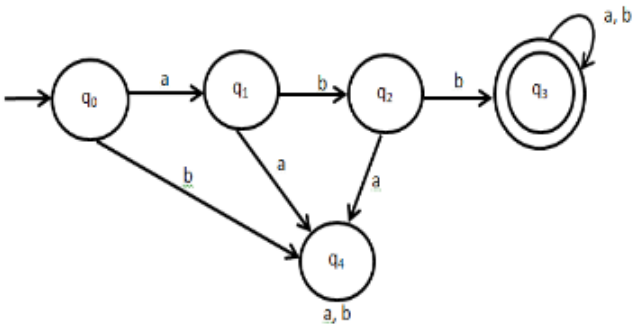
- Q: It is the non-empty finite set of states in the finite control.
- :
  δ: Q X Σ ->Q
- q_0: It is starting state, one of the states in Q.
- F: It is the non-empty set of final states accepting states from the set belonging to Q.

Some examples of contructing DFA:

Q1. Design a DFA which is minimal and which accepts strings over the alphabet Σ = {a, b}, where every accepted string 'w' starts with substring s= 'abb'.

Solution: The language accepted by the DFA, L = {abb, abba, abbb,……}

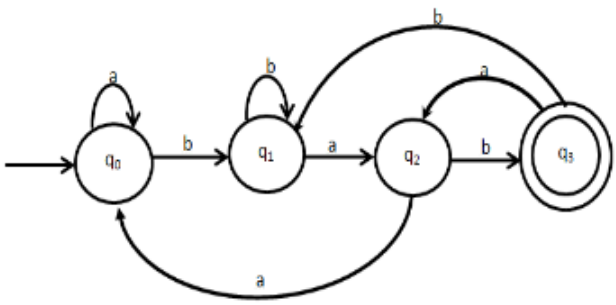
The number of strings in the set L is infinite and there is only one condition imposed which is "the string should start with "abb".



Q2. Design a DFA which is minimal and which accepts strings over the alphabet Σ ={a, b}, where every accepted string 'w' ends with substring s= 'bab'.

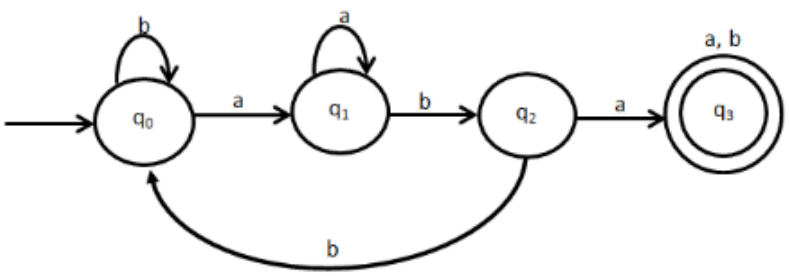
Solution: The language accepted by the DFA, L = { bab, abab, bbab,……}

The number of strings in the set L is infinite and there is only one condition imposed which is "the string should end with "bab".



Q3. Design a DFA which is minimal and which acceptsl strings over the alphabet Σ = {a, b}, where every accepted string 'w' contains substring s= 'aba'.

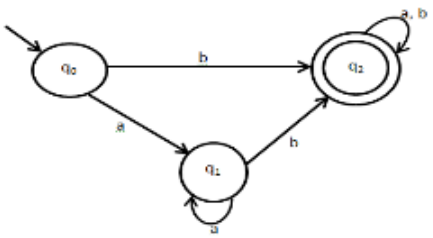
Solution:



The best knowledge about DFA can be only understood by designing a number of DFAs.

Representation of a finite automaton by the transition table:

It is a two dimensional table where number of columns is equal to the number of input alphabets and number of rows is equal to the number of states. The figure given below shows the transition table for the finite automaton on the left hand side.



It is

δ		Σ	
		a	b
Q	q_0	q_1	q_2
	q_1	q_1	q_2
	q_2	q_2	q_2

Transition ID- denoted by δ $(q_i, a) = q_j$ which means that the machine changes its state from state q_i to q_j if it encounters input symbol a.

Language accepted by DFA: A string is accepted by a DFA iff the DFA starting at the initial state ends in an accepting state (any of the final states) after reading the string wholly.

A string S is accepted by a DFA $(Q, \Sigma, \delta, q_0, F)$, iff $\delta^*(q_0, S) \in F$

The language L accepted by DFA is

{S | S ∈ Σ * and $\delta^*(q_0, S) \in F$}

A string S′ is not accepted by a DFA (Q, Σ, δ, q_0, F), iff

$\delta^*(q_0, S') \notin F$

The language L′ not accepted by DFA (Complement of accepted language L) is

{S | S ∈ Σ * and $\delta^*(q_0, S) \notin F$}

Example: Let us consider the DFA shown below in the diagram 1.3. From the DFA, the acceptable strings can be derived.

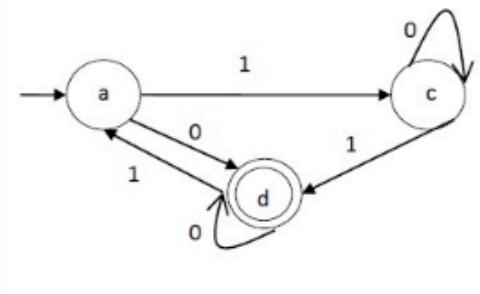


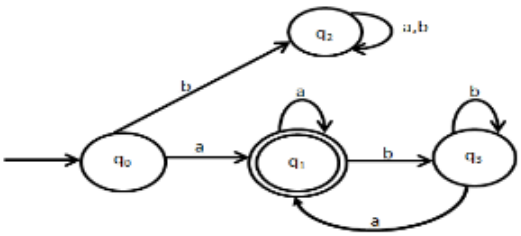
Diagram: 1.3

Strings accepted by the above DFA: {0, 00, 11, 010, 101, ...}

Strings not accepted by the above DFA: {1, 011, 111, ...}

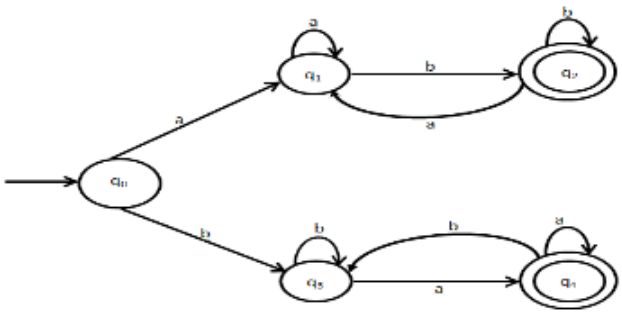
Example: Design a DFA which is minimal and which accepts strings over the alphabet Σ ={a, b} such that every accepted string start and end with 'a'.

Solution:



Example: Design a DFA which is minimal and which accepts strings over the alphabet Σ ={a, b} such that every accepted string start and end with different symbol.

Solution:



3.4.Summary:

i) Symbols are basic building blocks.

ii) An alphabet is a finite non-empty set of symbols.

iii) String is a sequence of symbols and the sequence is finite.

iv) A language is a set of strings.

3.5.Check your progress:

1. Define a Finite Automaton. Explain its components and how it accepts a string.
2. What is the difference between a DFA and an NFA? Provide an example to demonstrate how the transition functions in a DFA and NFA differ when processing the same input string.
3. Consider the DFA with the following transitions :

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

   -where $Q=\{q_0,q_1,q_2\}$, $\Sigma=\{0,1\}$, initial state is q_0 and final state is q_2

   Is the string "101" accepted by this DFA?

4. Explain the concept of Kleene Closure and Positive Closure in the context of a finite automaton.
5. Construct a DFA that accepts strings over the alphabet {0,1} that contain an even number of 0's. Provide the transition table and explain the working of the automaton.
6. Can a DFA be converted to NFA. Justify your answer.

MODULE I- Theory of Automata (Unit 4: Non-deterministic finite state machine (NFA)

4.0.Introduction, definition and designing of NFA:

In non-deterministic finite automata, rather than prescribing a unique move for each and every input symbol, a set of possible moves are allowed. Non-deterministic finite automata are not implemented. The only reason for which we study non-deterministic finite automata is that they are easy to design and can be easily converted into deterministic finite automata. So, if the situation demands a complex DFA, first we will go for designing a NFA (which is quite easy) and then convert it into the equivalent DFA. After that, we will check whether the equivalent DFA is minimum DFA or not; if it is not a minimum DFA then we will minimize it to achieve efficiency.

Definition of NFA: A non-deterministic finite automaton (NFA) is defined by 5-tuple:

$(Q, \Sigma, \delta, q_0, F)$, where:

- Q: A non-empty finite set of states in the finite control.
- Σ: A set of non-empty finite input symbols.
- δ: δ is a function that takes a state from Q and an input symbol, then produces a subset of Q as the result.. It is defined as:

  $\delta: Q \times \Sigma \rightarrow 2^Q$
- q_0: Initial state of NFA and member of Q.
- F: A non-empty set of final states and member of Q.

Important points:

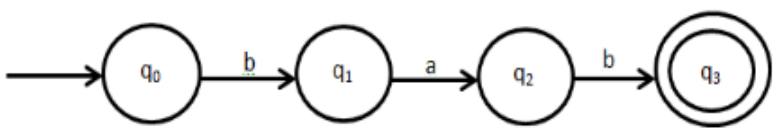
- The language accepting capability of both DFA and NFA are same , thus both are equivalent.
- Every NFA can be converted into a equivalent DFA,
- In NFA, it is not necessary to have transition on each of the input symbols in a state. Thus it need not to be a complete system

- In NFA, a single state can have multiple transitions on same input.
- Both of DFA and NFA recognize only regular languages.
- Null transition is possible in case of NFA and such special NFA are called Null-NFA.

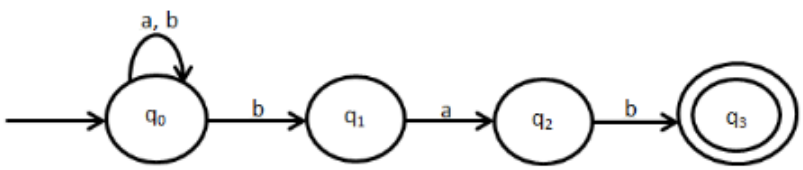
Designing NFA: The designing of NFA can be understood by working on some examples as shown below:

Example 1: Design a NFA which accepts the strings that can be generated over the alphabet Σ = (a, b), where every accepted string 'w' end with substring 's', where s = 'bab'.

Solution: Step1: Start with the acceptance of the smallest possible string, i.e. "bab".



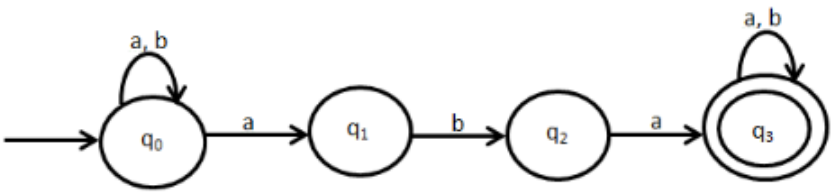
Step2: As there is no condition to satisfy on the starting of the starting of the string (only ending needs to be 'bab'), put a self-loop on the initital state for inputs a and b.



(In case of NFA, it is not necessary that the valid string is accepted in each move. It can be seen in the above NFA that an input string "bab" can have self loop on the initial state which will not be accepted. But, there should be at least one accepted move among all possible moves. No invalid string can be accepted in NFA.This is the non determinism.)

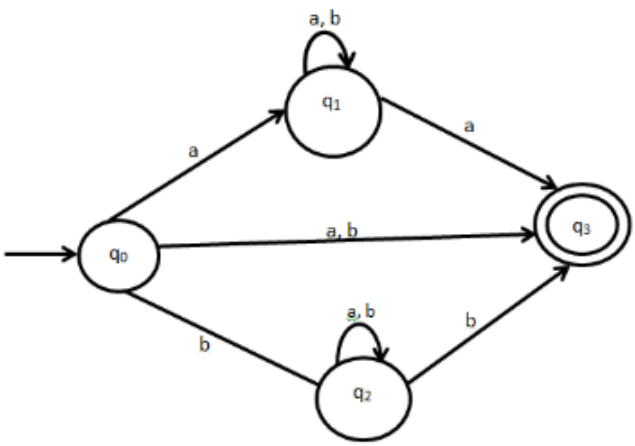
Example 2: Design a NFA which accepts strings over the alphabet Σ = (a, b), where every accepted string 'w' contains substring 's', where s = 'aba'.

Solution:



Example 2: Design a NFA that accepts all strings over the alphabet Σ = (a, b) such that every accepted string start and end with same symbol.

Solution:



Important points:

- Accepting power of DFA = Accepting power of NFA
- There is no concept of dead state in NFA
- The process of conversion from an NFA to DFA is called subset construction.

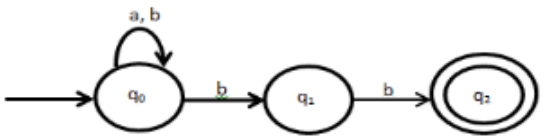
4.1. Conversion from NFA to DFA:

Let us do this with an example.

Example: Design a DFA which accepts strings over the alphabet Σ = (a, b) with the condition that every accepted string ending with "bb".

Solution:

Step1: Design the NFA.



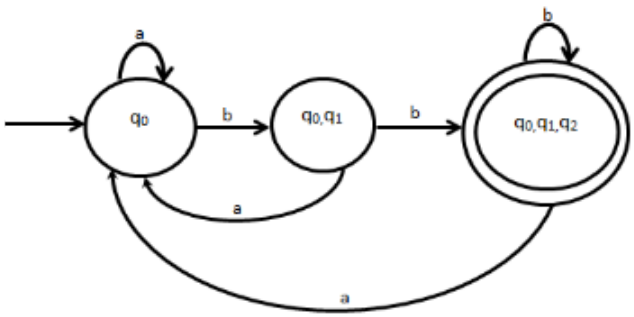
Step2: Make the transition for the NFA:

Step 3: Make the transition table for the equivalent DFA using the following rules:

- Input symbols and and the initial state will be same as in the NFA.
- First row for the intial moves are same as in the NFA.
- If there is any multiple move(as here in the NFA: q_0,q_1 are multiple moves.) consider it as a new single state inside curly brackets.
- If any multiple move is being considered as single state, then make new transition with the respected combined moves from the NFA's transition table.
- Every subset of states that contain the final state of the NFA is a final state in the DFA.

	a	B
→q_0	q_0	$\{q_0,q_1\}$
$\{q_0,q_1\}$	q_0	$\{q_0,q_1,q_2\}$
(q_2)	q_0	$\{q_0,q_1,q_2\}$

Step 4:



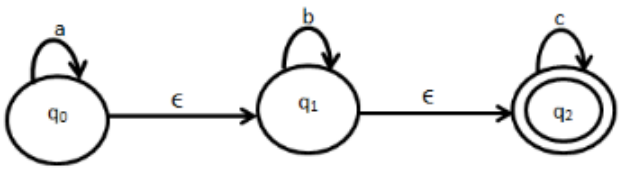
***Important Point**: If a NFA has 'n' states and we convert the NFA into equivalent DFA which has 'm' states then the relationship between n and m is: **$1<=m<=2^n$**

4.2: NFA with Epsilon moves (ϵ - NFA)

An automaton that has null transition is called null –NFA or ϵ - NFA, i.e. we allow a transition on null or empty string.

ϵ - NFA is a 5 tuple defined by 5-tuple (Q, Σ, δ, q_0, F), where:

- Q: A non-empty finite set of states in the finite control.
- Σ: A set of non-empty finite input symbols.
- δ is a transition function. δ:(Q X (Σ U)) -> 2^Q
- q_0: Initial state of NFA and member of Q.
- F: A non-empty set of final states and member of Q.



The language accepted by this ϵ - NFA is : L = { $a^n b^m c^q$ | n,m,q >=0}

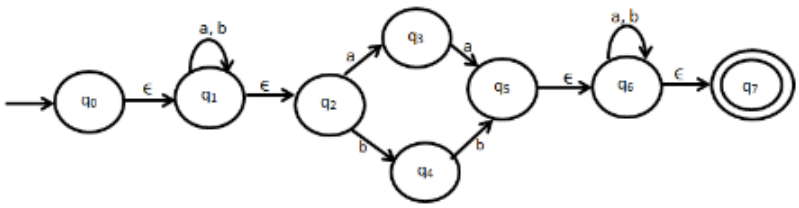
To effort needed to design a ϵ - NFA is much less than that to design a DFA or NFA.

Here, null closure(q_0)= q_0,q_1,q_2 ; null closure(q_1)= q_1,q_2 ; null closure(q_2)= q_2 .

Equivalence between Null NFA to NFA: The following points need to be considered while converting an Null NFA to NFA.

- There will be no change in the initial state.
- No change in the total number of states
- There may be a change in the number of final states.
- All the states will get the status of the final state in the resulting NFA whose ϵ - closure contains at least one final state in the initial ϵ - Closure.

Let us understatnd the above points by doing an example.



We are going to construct the equivalent NFA of the above ϵ - NFA.

	A	B
->q_0	q_1, q_2, q_3	q_1, q_2, q_4
q_1	q_1, q_2, q_3	q_1, q_2, q_4
q_2	q_3	q_4
q_3	q_5,q_6,q_4	ø
q_4	ø	q_5,q_6,q_7
	q_6,q_7	q_6,q_7
	q_6,q_7	q_6,q_7
	ø	ø

4.3.Summary:

i) In non-deterministic finite automata, rather than prescribing a unique move for each and every input symbol, a set of possible moves are allowed.

ii) If a NFA has 'n' states and we convert the NFA into equivalent DFA which has 'm' states then the relationship between n and m is: **$1 <= m <= 2^n$**

iii) Null closure of a set Q is defined as a set of all the states, which are at zero distance from the state Q

4.4.Check your Progress:

1. Define a Non-deterministic Finite Automaton (NFA) and explain the differences between NFA and DFA in terms of the transition function.
2. Explain the concept of epsilon (ϵ)-transition in an NFA.
   How does an NFA with epsilon transitions (ϵ-NFA) differ from a regular NFA?
   Illustrate with an example of an epsilon transition and explain its role in automaton behavior.
3. Design an NFA which accepts strings over the alphabet Σ={a,b}} that end with the substring "bab".
   Describe your construction process step by step and provide the transition diagram or table.

4.Design a NFA which accepts strings over the alphabet Σ = (a, b) such that every accepted string start and end with different symbol.

5. Design an NFA which accepts strings over the alphabet Σ={a,b} containing the substring "aba".

6. Convert the following NFA into a DFA:

Q={q_0,q_1}, Σ={0,1}, initial state is q_0 and final state is q_1. The transion function is defined as given below:

δ(q0,a)={q0,q1}, δ(q0,b)={q1}, δ(q1,a)={q1}, δ(q1,b)={q0}

7. Design an epsilon-NFA (ϵ-NFA) which accepts strings over the alphabet Σ={a,b} where each accepted string has at least one "a" followed by a "b".

MODULE I- Theory of Automata (Unit 5: Mealy and Moore model)

5.0.Introduction:

Both Moore and Mealy machine are special cases of DFA. Moore and Mealy machine produces outputs rather than accepting language. There is no concept of dead states and final states in case of Moore and Mealy machine and both are equivalent in case of power.

5.1. Definition and designing of Mealy and Moore machine:

Figure : Moore Machine

The transition diagram for the Moore machine shown above is given below:

Present State	Next State δ		Output λ
	a = 0	a = 1	
→ q_0	q_3	q_1	0
q_1	q_1	q_2	1
q_2	q_2	q_3	0
q_3	q_3	q_0	0

If the length of i/p string in Moore machine is n, then length of o/p string will be n+1. Moore machine gives response to empty string ϵ .

Mealy Machine: Mealy Machines are a type of finite state machine where the output symbol is determined by the transition. It can be represented by six components: $(Q, q_0, \sum, \Delta, \delta, \lambda)$, where:

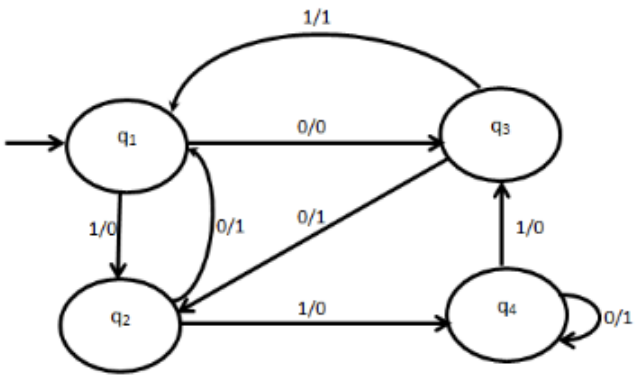


Figure : Mealy Machine

The transition diagram for the Moore machine shown above is given below:

Present State	Next State			
	a=0		a=1	
	State	Output	State	Output
→q_1	q_3	0	q_2	0
q_2	q_1	1	q_4	0
q_3	q_2	1	q_1	1
q_4	q_4	1	q_3	0

5.2.Summary:

i) Both Moore and Mealy machine are special cases of DFA.

ii) Moore Machines are finite state automata in which the output depends only on the present state.

iii) Mealy Machines are finite state automata where the output symbol depends on the transition.

5.3. Check your progress:

MODULE I- Theory of Automata (Unit 6: Minimization of Finite Automata)

6.0.Introduction:

The process of elimination of states whose presence or absence does not affect the language accepting capability of deterministic finite automata is called minimization of automata and the result is minimal deterministic finite automata or commonly known as minimal finite automat or MFA. The MFA for a particular language is always unique.

It is sometimes difficult to design a minimal DFA directly so, a better approach is to first design the DFA and then minimize it.

6.1.Definition of dead state, unreachable state and equal state :

Based on productivity, the states of DFA can be mainly classified in two types:

Productive states: State is said to be productive if it adds any accepting power to the machine that is its presence and absence effect the language accepting capability of the machine.

Non-productive states: These states do not add anything to the language accepting power to the machine.Non-productive states are further devided into three types:

- Dead State: It is basically created to make the system complete, can be defined as a state from which there is no transition possible to the final state. In a DFA there can be more than one dead state but logically always one dead state is sufficient to complete the functionality.
- Unreachable State: It is the state which can not be reached starting from initial state by parsing any input string.
- Equal State: These are those states that behave in same manner on each and every input string. That is for any string w where $w \epsilon \Sigma^*$, either both of the states will go to final state or both will go to non-final state.

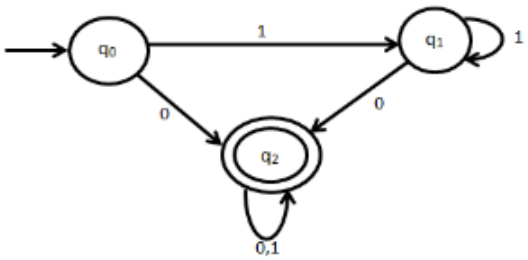
  More formally, two states q_1 and q_2 are equivalent ($q_1 \simeq q_2$) if both $\delta(q_1,x)$ and $\delta(q_2,x)$ are final states or both of them are non-final states for all $x \epsilon \Sigma^*$.If q_1 and q_2 are k-equivalent for all $k \geq 0$, then they are k-equivalent.

6.2.Procedure for minimization:

Steps to be followed:

- For this first of all, group all the non-final states in one set and all final states in another set.
- Now, on both the sets individually check, whether any of the underlying elements (states) of that particular set are behaving in the same way, that is, are they having same transition(to same set) on each input alphabet.
- If the answer is yes, then these two states are equal, otherwise not.

Example: Minimize the DFA :



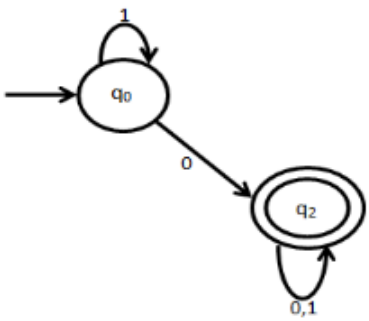
Solution: Set of non-final states: $\{q_0, q_1\}$ ------(i)

Set of final states: $\{q_2\}$-------(ii)

In set (i), behavior of q_0, q_1 is same on input symbol 0 as both move to final state q_2.

Behavior of q_0, q_1 is same on input symbol 1 as both move to states which are elements of the same set where q_0, q_1 belong to.

Thus $q_0 \simeq q_1$ and we can eliminate q_1.The minimized DFA is shown below:



6.3.Summary:

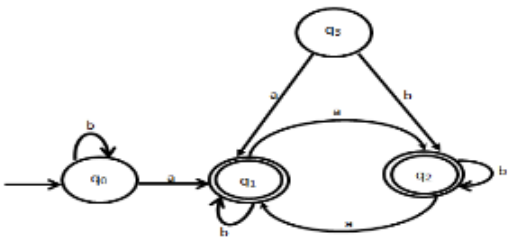
i) Productive states: State is said to be productive if it adds any accepting power to the machine that is its presence and absence effect the language accepting capability of the machine

ii) Dead state is basically created to make the system complete.

iii)Unreachable states can be eliminated.

6.4.Check your progress:

1) Minimize the DFA shown below:



2. Explain what minimization of a DFA is and why it is important.

1. What is the difference between productive and non-productive states in a DFA?
2. Explain the concept of a dead state in the context of DFA minimization.
3. What are unreachable states, and how can they be identified in a DFA?
4. Define equal states and explain how the equal states are identified in the minimization process. How does the behavior of two equal states affect the DFA's language acceptance?

MODULE II- Formal Languages, Regular Sets and Regular Grammars

(Unit 7: Definition of formal languages)

7.0. Introduction:

In the theory of computation, a formal language is a set of strings of symbols that are defined by rules and used to describe aspects of computation. Formal languages are fundamental to understanding how computers work and process information.

7.1. Definition of formal language, Terminal symbols and non-terminal symbols :

In theory of computation, a formal language is a set of strings (strings are composed of symbols) that may be generated by a formal grammer.

E.x.: Σ ={a,b}, and L is a language which is defined as L= { a, aa, ab, aaa, aba, aab, ab, ……..}

Grammar: Grammar is a finite set of formal rules that are generating syntactically correct sentences.

A grammar is formally defined as a set of four components: G = (V, T, P, S), where:

- G is a grammar, which consists of a set of production rules. It is used to generate the strings of a language.
- T is the final set of terminal symbols. It is denoted by lower case letters.
- V is the final set of non-terminal symbols. It is denoted by capital letters.

- P is a set of production rules, which is used for replacing non-terminal symbols (on the left side of production) in a string with other terminals (on the right side of production).
- S is the start symbol used to derive the string.

Grammar is composed of two basic elements: Terminal symbol and non-terminal symbol.

Terminal Symbols - Terminal symbols are the parts of the sentences generated using grammar and are denoted using small case letters like a, b, c etc.

Non-Terminal Symbols - Non-Terminal Symbols take part in the generation of the sentence but are not the component of the sentence. These types of symbols are also called auxiliary symbols and variable.These are represented with capital letters like A,B,C etc.

Example 1: Consider a grammar G = (V , T , P , S) Where,

V = { S , A , B } ⇒ Non-Terminal symbols

T = { a , b } ⇒ Terminal symbols

Production rules P = { S → ABa , A → BB , B → ab , AA → b }

S = { S } ⇒ Start symbol

Example 2 Consider a grammar G=(V,T,P,S) Where,

V= {S, A, B} ⇒ non terminal symbols

T = { 0,1} ⇒ terminal symbols

Production rules P = { S→A1B, A→0A| ε, B→0B| 1B| ε }

{S} ⇒ start symbol.

7.2 Check your Progress:

1. What is the difference between a string and a language in formal language theory?
2. Define grammer in the context of automata theory.
3. Design a grammer which can generate the language, L={ab,abbbbb,abaa,aba,abbb…..}
4. Why are natural languages not suitable for machine communication, and how do machine languages differ in terms of complexity?
5. In the context of formal languages, what is the difference between a symbol, an alphabet, a string, and a language? Provide examples for each.
6. In the context of formal languages, what is the difference between a symbol, an alphabet, a string, and a language? Provide examples for each.

MODULE II- Formal Languages, Regular Sets and Regular Grammars(Unit-8 Chomsky Hierarchy)

8.0.Introduction:

The Chomsky hierarchy of languages is a classification system that categorizes formal grammars based on their generative power. It was proposed by Noaman Chomsky, a renowned linguist and computer scientist, in the 1950s. The hierarchy consists of four levels, each representing a different class of formal languages. These levels are known as Type-3 (Regular), Type-2 (Context-Free), Type-1 (Context-Sensitive), and Type-0 (Unrestricted).

8.1: Type-0, type-1.type-2, type3 grammers :

Type-0 grammar encompasses all formal grammars. The languages produced by type-0 grammars can be recognized by a Turing machine and are referred to as Recursively Enumerable languages. This type of grammar is also called an unrestricted grammar. In type-0 grammar, at least one of the variables must appear on the left side of a production.

For example: $Sab \longrightarrow ba$

Type-1 grammer: Type-1 grammar generates context-sensitive language. These languages are recognized by the Linear Bound Automata .

For a grammer to be type -1:

- It should be type – 0 grammer.
- The number of symbols on the left-hand side must not exceed the number of symbols on the right-hand side.

- The rule of the form A $\longrightarrow \epsilon$ is not allowed unless A is a start symbol.

For example: S $\longrightarrow$ AB
B $\longrightarrow$ b

Type-2 grammer: Type-2 grammar generates context-free language. These languages are recognized by the Push Down Automata .

For a grammer to be type -2:

- It should be type – 1 grammer.
- The left-hand side production can have only one variable.
- There is no restriction on the right hand side.

For example: S $\longrightarrow$ AB

B $\longrightarrow$ b

Type-3 grammer: Type-3 grammar generates regular language. These languages are recognized by the Finite Automata . Type-3 is the most restricted form of grammer. Type -3 or regular grammer is further devided into two categories: Left linear grammer and Right linear grammer.

For a grammer to be regular grammer :

- It should be type -2 grammer.
- Here position of Non-terminal in r.h.s is either on extreme left or on extreme right. Thus the production like A $\rightarrow$ aBa is not allowed in regular grammer.

Thus, regular grammer can be of two types: Left linear and Right linear.

Left linear grammar	Right linear grammar
All the non-terminals on the right-hand side exist at the rightmost place, i.e; right ends. Ex: A $\longrightarrow$ aB	All the non-terminals on the left-hand side exist at the leftmost place, i.e; left ends. Ex: A $\longrightarrow$ Ba

*** If we make production rules combining left linear and right linear, the grammer will be no longer regular.

8.1: Relations between language, grammer and Automata:

Grammar	Language	Automation that accepts	Production rules
Type-0	Recursively enumerable	Turing machine	No restriction
Type-1	Context-sensitive	Linear-bounded non-deterministic machine	$\alpha A\beta \rightarrow \alpha\gamma\beta$
Type-2	Context-free	Non-deterministic push down automata	$A \rightarrow \gamma$
Type-3	Regular	Finite state automata	$A \rightarrow \alpha B$ $A \rightarrow \alpha$

8.2.Summary:

i) The Chomsky hierarchy categorizes formal grammars into 4 types based on their generative power.

ii) Regular grammar are of two types: Left linear and Right linear.

8.3.Check your progress:

1. Explain the four types of grammars in the Chomsky Hierarchy and provide an example for each type.
2. What are the key features that distinguish Type-0 grammars from other types?
3. How do context-sensitive grammars differ from context-free grammars (Type-2) in terms of production rules and generative power.
4. Provide an example of a context-free grammar and describe the language it generates.
5. Explain the relationship between regular grammars and finite automata. Provide an example of a regular grammar.
6. Can a context-sensitive language ever be a context-free language? Justify your answer with an example.
7. What closure properties hold for languages of each type in the Chomsky Hierarchy?

MODULE II- Formal Languages, Regular Sets and Regular Grammars

(Unit 9: Regular Language and Regular Expressions)

9.0.Introduction:

Regular languages can be described using regular expressions. A regular expression is a notation that represents a set of strings, specifically those that belong to a regular language. These expressions are used to represent particular sets of strings, or languages, in an algebraic form.

We give formal recursive definition of regular expression over $\sum = \{a, b\}$ as follows:

Any terminal symbol, i.e. an element of $\sum$, ϵ and ø are regular expressions(primitive regular expression). A regular expression is valid iff it can be derived from a primitive regular expression by a finite number of applications of operators.

Regular language: Any set represented by a regular expression is called a regular Language. For example: $a, b \in \sum$ then, R =a denotes the language, L ={a}

R =a.b denotes the language, L ={ab}-------- Concatenation

R =a+b denotes the language, L ={a,b}--------Union

R =a^* denotes the language, L ={ ϵ ,a,aa,aaa…….}--------Kleene closure

R =a^+ denotes the language, L ={ a,aa,aaa…….}--------Positive closure

R =$(a+b)^*$ denotes $\{a,b\}^*$

Identities for Regular Expression:

Two regular expressions P and Q are equivalent(P=Q):

- If P and Q represent the same set of strings.

Each regular expression can generate only a single regular language, but a regular language can be produced by multiple regular expressions. In other words, two different regular expressions can define the same language.

Example: i) Check wheather the regular expressions R_1 and R_2 are equal or not: $R_1 = a^*$, $R_2 = a^* + (aa)^*$.

Solution: The language represented by both R_1 and R_2 is { ϵ,a,aa,aaa……..}.

So, R_1 and R_2 are equal.

Example: ii) Find out the regular language represented by the regular expression, R= {a}

Solution: L={a}

Example: iii) Find out the regular language represented by the regular expression, R= a+b

Solution: L={a,b}

Example: iv) Find out the regular language represented by the regular expression, R= a+b+c

Solution: L={a,b,c}

Example: v) Find out the regular language represented by the regular expression, R= a.b

Solution: L={ab}

Example: vi) Find out the regular language represented by the regular expression, R= (a.b+a)b

Solution: L={abb, ab}

Example: vii) Design a regular expression that represent a language 'L', where L={a} over the alphabet ∑ = {a, b}.

Solution: R=a

Example: viii) Design a regular expression that represent all strings over the alphabet ∑ = {a, b}, where every accepted string 'w' start with substring 'abb'.

Solution: R= abb(a+b)*

Example: ix) Design a regular expression that represent all strings over the alphabet ∑ = {a, b}, where every accepted string 'w' end with substring 'abb'.

Solution: R= (a+b)*abb

(Note: (a+b)* can generate any combination of a and b including ϵ)

Example: x) Design a regular expression that represent all strings over the alphabet ∑ = {a, b}, where every accepted string 'w' contains substring 'abb'.

Solution: R= (a+b)*abb(a+b)*

Example: xi).Design a regular expression that represent all strings over the alphabet ∑ = {a, b}, where every accepted string 'w' starts and ends with 'a'.

Solution: R= a+ a(a+b)*a

Example: xii).Design a regular expression that represent all strings over the alphabet ∑ = {a, b}, where every accepted string 'w' starts and ends with same symbol.

Solution: R= a+ a(a+b)*a + b+ b(a+b)*b

Example: xiii).Design a regular expression that represent all strings over the alphabet ∑ = {a, b}, where every accepted string 'w' starts and ends with different symbol.

Solution: R= a(a+b)*b + b+ b(a+b)*a

Example: xiv).Design a regular expression that represent all strings over the alphabet ∑ = {a, b}, where every accepted string 'w' is like w=SX, S=aaa/bbb

Solution: R= (aaa+bbb)(a+b)*

Algebric properties of regular expression: Let us consider R_1 , R_2 and R_3 are three regular expressions to understand the algebraic properties of regular expression.

1. Closure property: Regular expressions are closed w.r.t union, concatenation and Kleene closure. R_1+R_2, $R_1.R_2$, R_1^*, R_1^+ all these are regular expressions.

2. Associative property: Regular expression satisfy associative property w.r.t union and intersection.

$$(R_1+R_2)+ R_3 = R_1+(R_2+ R_3)$$
$$(R_1.R_2).R_3 = R_1.(R_2.R_3)$$

3. Identity property: Regular expression satisfy identity property under union and intersection.

$$R_1.\epsilon = R_1$$
$$R_1+\emptyset = R_1$$

4.Commutative property: Regular expressions are commutative w.r.t union but not with concatenation.

$$R_1+R_2 = R_2+ R_1$$
$$R_1.R_2 \neq R_2.R_1$$

5.Distributive property: Regular expression satisfy this property as follows:

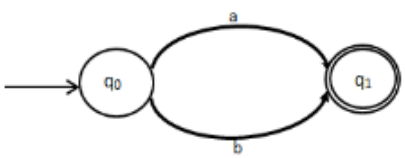
$$R_1+(R_2+ R_3) = R_1.R_2+ R_1.R_3$$
$$(R_1+R_2).R_3 = R_1.R_3+ R_2.R_3$$
$$R_1+(R_2.R_3) \neq (R_1+ R_2)+ (R_1+R_3)$$
$$(R_1.R_2)+R_3 \neq (R_1+ R_3).(R_2+R_3)$$

6.Idempotent property: Regular expressions satisfies idempotent property w.r.t. union but not with concatenation.

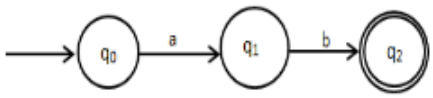
$$R_1+R_2 = R_1$$
$$R_1.R_1 \neq R_1$$

9.1.Finite Automata and Regular Expression:

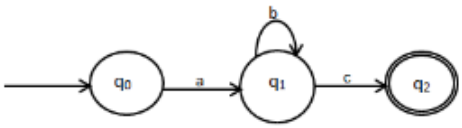
Since finite automata are capable of recognizing regular languages, and regular expressions also define regular languages, it is possible to transform a finite automaton into a regular expression. Let's examine this idea with the following examples.

Regular Expression	Corresponding Finite Automata
1. a + b	

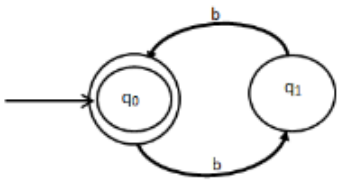
2. a.b



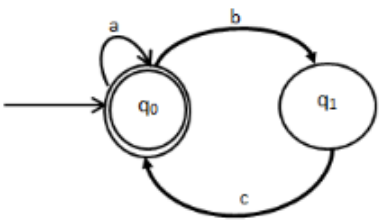
3. a. $b^*.c$



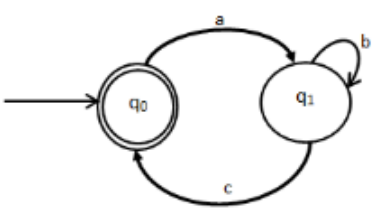
4. b^*



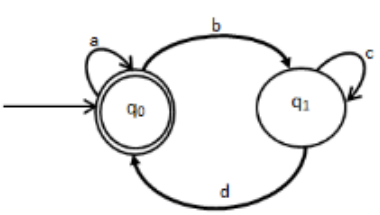
5. $(a + bc)^*$



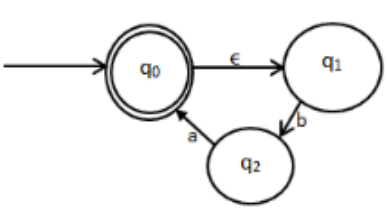
6. $(a(b + ca)^*c)^*$



7. $(a+bc^*d)^*$

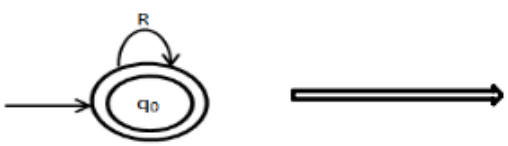
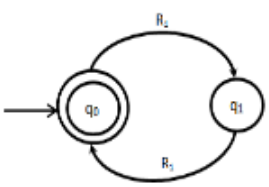
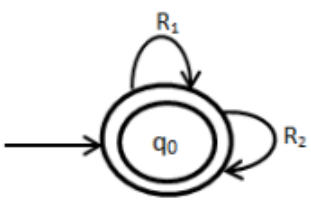


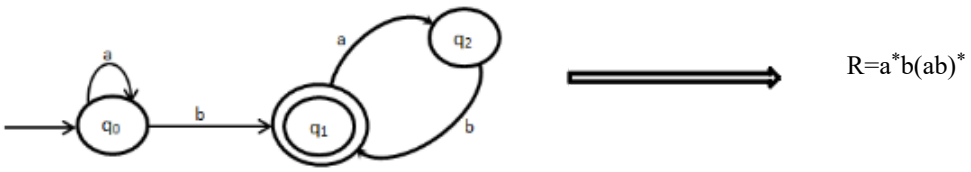
8. $(\epsilon.b.a)^* = (ba)^*$



9.2.Conversion from to Regular Expression to Finite Automata:

Let us understand the conversion from to Regular Expression to Finite Automata with the following examples.

Finite Automata	Corresponding Regular Expression
	R^*
	$(R_1.R_2)^*$
	$(R_1+R_2)^*$



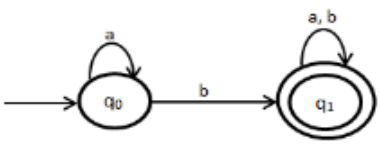
$R=a^*b(ab)^*$

ARDEN'S Theorem: It is a mechanism for constructiing regular expression from a DFA (it is not applicable to NFA or ϵ - NFA). Here the following steps are used:

- For every individual state of the DFA, write an expression for every incoming and outgoing input alphabet.
- Now apply Arden's theorem as follows:
  - I. If P is free from NULL, then equation R=Q+RP has unique solution, $R=QP^*$
  - II. If P contains NULL, then equation R=Q+RP has infinitely many solution.

Example of using Arden's theorem:

Question: Consider the DFA given below and convert it into regular expression using Arden's theorem.



Solution: $\delta(q_0,a)= q_0$

$\delta(q_0,b)= q_1$

$\delta(q_1,a)= q_1$

$\delta(q_1,b)= q_1$

We write equation for each state: $M = \epsilon + Ma$ ----------(i)

$N = Mb + N(a+b)$ ----------(ii)

Comparing equation (i) with the equation R=Q+RP : Q = ϵ , R = M , P=a

Thus, from Ardon's theorem, $M= \epsilon a^*$ $M= a^*$

Putting the value of M in equation (ii): $N = a^*b + N(a+b)$ ----------(iii)

Comparing equation (iii) with the equation R=Q+RP : $Q = a^*b$, R = N , P=a +b

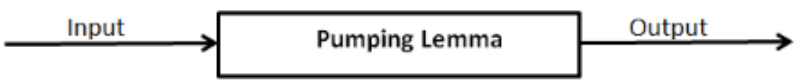
Thus, from Ardon's theorem, $N= a^*b\ (a+b)^*$. N is the final state. So the regular expression represented by the given DFA is : $a^*b\ (a+b)^*$.

9.3.Pumping lemma**:**

The pumping lemma is a technique used to prove that a language is not regular. It operates through a proof by contradiction: here we assume that the language is regular and then we prove that this assumption leads to a contradiction, proving the language is not regular.

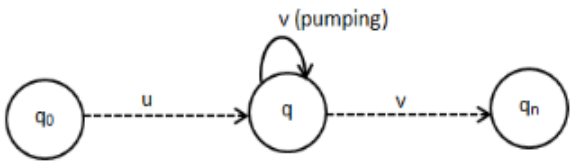
A regular language will always satisfy the pumping lemma. However, if even one string produced by pumping is not part of the set of strings defined by the language L, then L is certainly not regular. Conversely, the fact that the pumping lemma holds does not necessarily mean the language is regular.

For the pumping lemma, the input and output both are non-regular languages.



Formal definition of pumping lemma: For any regular language L, there exists an integer n, such that for $z \in L$ with $|z| \geq n$, there exists $u,v,w \in \Sigma^*$, such that z=uvw and :

- $|uv| \leq n$
- $|v| \leq n$
- For all $i \geq 0 : uv^iw \in L$



In simple words, if a string v is 'pumped' (pumped means inputing v any number of times), the resultant string still remains in the language L.

Let us apply pumping lemma to proof that the language L= $\{a^mb^n \mid m=n\}$ is not regular.

Here L = { ab, aabb, aaabbb,……}

For the time being, let us assume L as a regular language

Let us also consider a sample string $z \in L$. Now, in the given language $z = a^kb^k$.

For i=1: $uv^iw = a^kb^k = a^{k-1}ab^k$ ($u = a^{k-1}$, $v = a$, $w = b^k$)

For i=2: $uv^iw = a^{k+1}b^k = a^{k-1}a^2b^k$ ($u = a^{k-1}$, $v = a^2$, $w = b^k$)

Now, we can see that $a^{k+1}b^k$ (for i=2) does not belong to the language L= $\{a^mb^n \mid m=n\}$ because in L the powers of a and b are equal. So, the definition of pumping lemma is not fulfilled here and our assumption of considering L as a regular language is not true. Thus L is not a regular language.

(***The basic idea of applying pumping lemma to prove the irregularity of a language depends on the fact how the string is divided into three parts (u, v and w))

9.4.Regular sets and regular grammars

Languages typically consist of an infinite number of strings, making it impractical to list every possible string to represent the language. As a result, we use grammar, a mathematical model similar to automata, to represent the language. Grammar essentially acts as a generator for the language.

A grammar is defined as a 4-tuple (V_N, $\sum$, P, S) where:

- V_N is a finite, nonempty set of elements called nonterminals.
- $\sum$ is a finite, nonempty set of elements called terminals, and $V_N \cap \sum = \emptyset$.

- S is a special nonterminal ($S \in V_N$), known as the start symbol, and every grammar has exactly one start symbol.
- P is a finite set of rules where each rule has the form $\alpha \rightarrow \beta$, with α and β being strings formed from $V_N \cup \sum$, and α must contain at least one symbol from V_N. P's elements are referred to as production rules.

The concept of defining a language using grammer is, starting from a start symbol using the production rules of the grammer any time , deriving the string. During derivation, each time, a production rule is used as its LHS is replaced by its RHS, all the intermediate stages (strings) are called sentential forms. The language formed by the grammer consists of distinct strings that can be generated in this manner.

$L(G)=\{w|w \in \sum^*, S \xrightarrow{*} w\}$ [$\xrightarrow{*}$ means taking any number of productions S ends upon w]

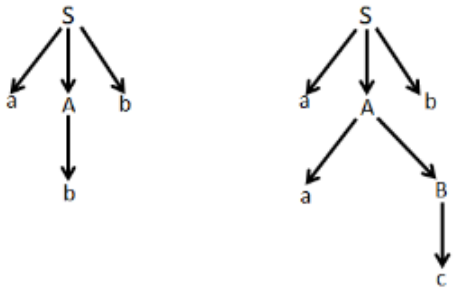
L(G) is the set of all terminal strings which can be derived from the start symbol S. Two grammers G_1 and G_2 are equivalent if $L(G_1)=L(G_2)$.

Let us solve the following questions to make some understandings about grammer:

Question 1: Identify the language of the following grammer.

$S \longrightarrow aAb$
$A \longrightarrow aB / b$
$B \longrightarrow c$

Solution: Let us try to build trees where root is the start symbol and leafs are terminals. We can build two trees here as shown below:

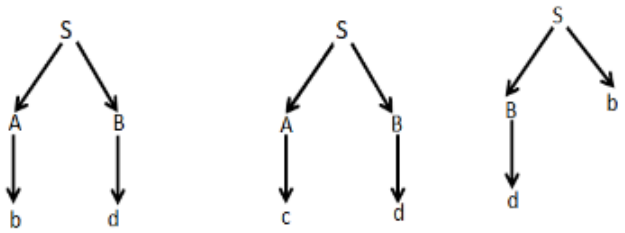


We can conclude that the given grammer generates the language, L ={abb, aacb}.

Question 2: Identify the language of the following grammer.

$S \longrightarrow AB/Bb$
$A \longrightarrow b/c$
$B \longrightarrow d$

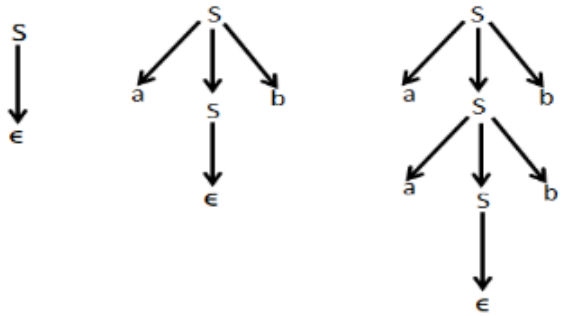
Solution: We can build three different trees where root is the start symbol and leafs are terminals.



We can conclude that the given grammer generates the language, L ={bd, cd, db}.

Question 3 : Identify the language of the grammer: $S \longrightarrow aSb/\epsilon$.

Solution: We can build three different trees where root is the start symbol and leafs are terminals.



We can conclude that the given grammer generates the language,

L ={ ϵ, ab, aabb……}. So, L can be written as, L ={$a^n b^n$| n ≥ 0}.

9.5.Conversion of regular grammar to finite automata:

The steps involved in converting a regular grammar into a finite automaton are as follows:

- Start from the first production.

- From every left alphabet (or variable) go to the symbol followed by it.
- Start state: It will be the first production state.
- Final state: Take those states which end up with terminals without further non-terminals.

Example: Design the finite automata for the grammer given below:

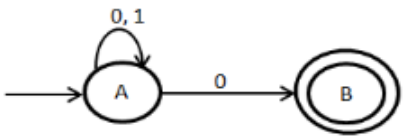
$A \longrightarrow 0A/1B/0B$

$B \longrightarrow \epsilon$

Solution: Start with variable A and use its production.

- For production $A \longrightarrow 0A$, this means after getting input symbol 0, the transition will remain in the same state.
- For production, $A \longrightarrow 1B$, this means after getting input symbol 1, the state transition will take place from State A to B.
- For production $A \longrightarrow 0B$, this means after getting input symbol 0, the state transition will be from State A to B.
- For production $B \longrightarrow \in$, this means there is no need for state transition. This means it would be the final state in the corresponding FA as RHS is terminal.

So the final NFA for the corresponding RLG is



The language represented by this NFA is the set of all strings that end with 0.

9.6. Conversion of finite automata to regular grammer:

i) Conversion to right linear grammer:

The steps for the conversion of finite automata (FA) to the right linear grammar are as follows –

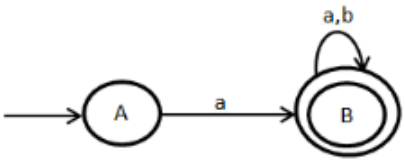
Step 1 − Begin the process from the start state.

Step 2 − Repeat the process for each state.

Step 3 − Write the production as the output followed by the state on which the transition is going.

Step 4 − And at last, add ϵ (epsilon) to end the derivation.

Example: Convert the finite automata given below to Right linear grammer.



Solution: Pick the start state A and output is on symbol ‘a’ going to state B

A → aB

Now we will pick state B and then we will go on each output

i.e., B → aB

B → bB

B → ε

Therefore the final right linear grammar is as follows −

A → aB

B → aB/bB/ε

ii) Conversion to left linear grammer:

The steps for the conversion of finite automata (FA) to the left linear grammar are as follows −

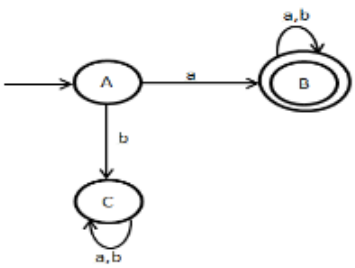
Step 1 − Take reverse of the finite automata

Step 2 − Write right linear grammar

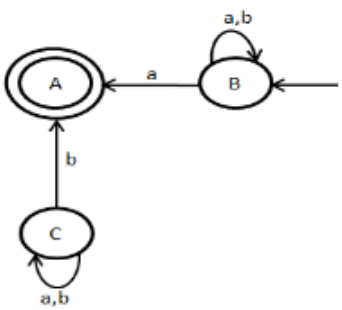
Step 3 − Then take reverse of the right linear grammar

Step 4 − And finally, you will get the left linear grammar

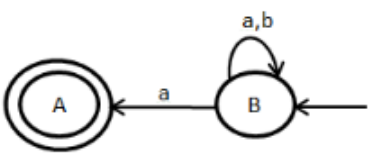
Example: Convert the finite automata given below to Right linear grammer.



Solution: Make final state as initial state and initial state as final state, as shown below –



Now remove the states which can not be reached from the starting state as shown below –



After eliminating the unreachable states, the transition diagram does not represent a deterministic finite automaton because state A has no output, while state B has two outputs for the symbol 'a'.

So the resultant diagram is a Non-deterministic finite automata (NFA).

First, generate the right linear grammar for the final transition diagram −

B→aA/aB/bB

A→ε

Now reverse the right linear grammar to generate left linear grammar −

B → Ba/Bb/Aa

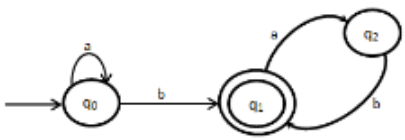
A→ε

9.7.Summary:

i) If P and Q are two regular expressions and they are equivalent (P=Q),if P and Q represent the same set of strings.

ii) Pumping lemma is used to prove the irregularity of a language.

1. Convert the finite automaton given below to regular expression:



Module-III Context Free Language (Unit-10: Context free languages and derivation tree)

10.0 Introduction:

The languages produced by type-2 grammars are referred to as context-free languages. The strings belonging to a context-free language are recognized by pushdown automata. Context-free languages play a crucial role in compilers, particularly during the parsing phase.

For a grammer to be type -2:

- It should be type – 1 grammer.
- The left-hand side production can have only one variable.
- There is no restriction on the right hand side.

For example:

$S \longrightarrow AB$
$B \longrightarrow b$

If all the productions are type-2 grammer then the grammer is called context free grammer. Before starting CFL, two terminologies on which we have done work already in type-3 grammers are defined below:

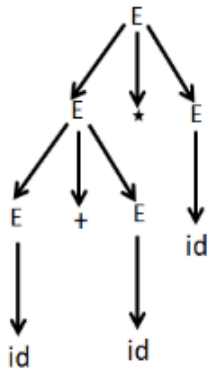
10.1. Derivation tree of Context free grammar.

Derivation: The process by which derivation of a string is done is known as derivation.

Parse Tree/Syntax Tree/Derivation Tree: When derivation is represented by a graph it is known as derivation tree.

Example: E ⟶ E + E / E * E / id

The corresponding derivation tree:

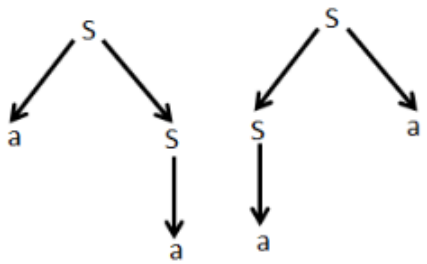


10.2.Left most derivation(LMD) and Right most derivation(RMD): The process of construction of parse tree or derivation tree by expanding the left most terminal is known as LMD and the construction of parse tree or derivation tree by expanding the right most terminal is known as RMD.

The parser tree in the above example is built using leftmost derivation.

10.3. Ambiguous grammer: The grammer CFG is said to be ambiguous if there exist more than one parse tree (LMD or RMD) for any string produced by the grammer. (There is no concept of ambiguity in regular grammer.)

For example, S ⟶ aS/Sa/a is an ambiguous context free grammer as we can produce the string "aa" with two different trees as shown below:



Unambiguous grammer: The grammer CFG is said to be unambiguous if there exists only one parse tree for every string produced by the grammer.

For example, S ⟶ aSb/ab is an unambiguous context free grammer as every string produced by the grammer has exactly one parse tree.

10.4.Summary:

i) The languages produced by type-2 grammers are known as context free languages.

ii) The grammer CFG is ambiguous if there exist more than one parse tree (LMD or RMD) for any string produced by the grammer.

iii)There is no concept of ambiguity in regular grammar.

Check progress:

1. What is parse tree?
2. Which one of the following statements is FALSE?

a) There exist context-free languages such that all the context-free grammars generating them are ambiguous.
b) An unambiguous context free grammar always has a unique parse tree for each string of the language generated by it.
c) Both deterministic and non-deterministic pushdown automata always accept the same set of languages.
d) A finite set of string from one alphabet is always a regular language.

3. Consider the context-free grammar G below
   S→aSb|XX→aX|Xb|a|b,
   where S and X are non-terminals, and a and b are terminal symbols. The starting non-terminal is S.Which one of the following statements is CORRECT?
   a) The language produced by G is (a+b)∗
   b) The language produced by G is a∗(a+b)b∗
   c) The language produced by G is a∗b∗(a+b)
   d) The language produced by G is not a regular language

4. what is the langauge generated by this grammar: S-->aS | aSbS | ε .
5. Select the correct statement: A context-free grammar is ambiguous if:

a) The grammar contains useless non-terminals.
b) The grammer produces more than one parse tree for some sentence.
c) Some production has two non terminals side by side on the right-hand side.
d) None of the above.

6. If L1 and L2 are context free languages and R a regular set, one of the languages below is not necessarily a context free language. Which one?
   a) L1L2
   b) L1∩L2
   c) L1∩R

d) L1∪L2

Module-III Context Free Language (Unit 11: Simplification of Context free grammer)

11.0.Introduction:

We need to simplify or minimize CFG to make it more efficient and compiler friendly.The process of deleting and eliminating useless symbols, unit production and null production is known as simplification of context free grammer.

11.1. Process of minimization :

The process of minimization by eliminating null production is shown below with an example.

Let the CFG is: $S \longrightarrow AbB$
$A \longrightarrow a / \varepsilon$
$B \longrightarrow b/ \varepsilon$

After the elimination of null productions, the CFG becomes: $S \longrightarrow AbB/Ab/Bb/b$
$A \longrightarrow a$
$B \longrightarrow b$

The process of minimization by eliminating unit production (e.x. $A \longrightarrow B$, where $|A|=|B|=1$) is shown below with an example:

Let the CFG is: $S \longrightarrow Aa$
$A \longrightarrow a /B$

B ⟶ d

After the elimination of unit productions, the CFG becomes: S ⟶ AbB/Ab/Bb/b
A ⟶ a/d

B ⟶ d is removed as has become useless now.

The minimization process by eliminating useless symbols is illustrated below with an example. Useless symbols are variables that do not participate in deriving any string. To minimize:

- Identify the variables that cannot be reached from the start symbol of the grammar and remove them, along with all their productions.
- Identify the variables that can be reached from the start symbol but do not lead to any terminal symbols,then remove them along with their productions.

Let the CFG is:

S ⟶ aAB/bA/aC
A ⟶ aB/b
B ⟶ aC/d
C ⟶ d

Here, C is not producing any terminal.

After the elimination of useless symbols, the CFG becomes:

S ⟶ aAB/bA
A ⟶ aB/b
B ⟶ d

11.2.Chomsky Normal Form(CNF): The grammer G is said to be in Chomsky Normal Form, if every production is in the form: A ⟶ BC/a , where B,C ∈ V_n and a ∈ Σ. To make a context free grammer compiler friendly, we transform into Chomsky normal form.

Let's understand the idea behind CNF with the following example.

We have a grammer: S ⟶ aSb/ab. Let us take two non-terminals A and B such that:

A ⟶ a and B ⟶ b

Now, the grammer becomes: S ⟶ ASB/AB, A ⟶ a and B ⟶ b .

Next, let us take a new non-terminals C such that: C ⟶ AS

Now, the grammer becomes: S ⟶ CB/AB, A ⟶ a , B ⟶ b and C ⟶ AS which is in CNF.

***Note:** If a context free grammer is in CNF, then for a derivation of string w, with length n, we need exactly 2n – 1 production. |W|=n, number of sentential forms will be 2n-1.

11.3.Greiback Normal Form(GNF): The grammer G is said to be in Chomsky Normal Form, if every production is in the form: $A \longrightarrow a\alpha$, where $A \in V_n$, $a \in \Sigma$ and $\alpha \in V_n^*$. To make a context free grammer compiler friendly, we transform into Greiback normal form.

Let's understand the idea behind CNF with the following example.

We have a grammer: $S \longrightarrow$ aSb/ab. Let us take a non-terminal B such that: $B \longrightarrow$ b

Now, the grammer becomes: $S \longrightarrow$ aSB/ab and $B \longrightarrow$ b which is in GNF.

11.4.Summary:

i) We need to simplify or minimize CFG to make it more efficient and compiler friendly.

ii) The process of deleting and eliminating useless symbols, unit production and null production is known as simplification of context free grammer

11.5.Check your progress:

1. Simplify the CFG given below:

$S \longrightarrow AB$
$A \longrightarrow a / \varepsilon$
$B \longrightarrow b/ \varepsilon$

2. Identify the type of normal form which accepts the grammer given below:

AB->CD
A->BC or
A->B or
A->a

where A, B, C, D are non terminal symbols and a is a terminal symbol.

a) Greibach Normal Form
b) Chomsky Nrmal Form
c) Kuroda Normal Form
d) None of the mentioned

4. Given a grammar in GNF and a derivable string in the grammar with the length n, any _____will halt at depth n.
a) top-down parser
b) bottom-up parser

c) multitape turing machine
d) none of the mentioned

5. Every grammar in CNF is: a)regular b)context sensitive c)context free d)all of the mentioned.

6. Convert the following CFG into an equivalent CFG in CNF.

$A \rightarrow BAB|B|\varepsilon$, $B \rightarrow 00|\varepsilon$

7. If G is a context free grammar and w is a string of length l in L(G), how long is a derivation of w in G, if G is in CNF:

A. $2l$
B. $2l+1$
C. $2l-1$
D. l

Module-III Context Free Language (Unit-12: Pumping lemma for context free language)

12.0. Introduction:

The pumping lemma is a tool that can be used to construct a refutation by contradiction that a specific language is not context-free. Conversely, the pumping lemma does not suffice to guarantee that a language is context-free.

12.1. Applications of the Pumping Lemma:

The Pumping Lemma is useful for determining whether a grammar is context-free. Let's consider an example to illustrate how this is done.

Problem: Determine if the language $L = \{x^n y^n z^n \mid n \geq 1\}$ is context-free.

Solution: Assume L is a context-free language. According to the Pumping Lemma, L must satisfy its conditions. First, choose a pumping length n as per the lemma. Then, take a string z such as $0^n1^n2^n$ and break it into substrings uvwxy, where $|vwx| \leq n$ and $vx \neq \varepsilon$. This means vwx cannot contain both 0s and 2s, as the last 0 and the first 2 are at least n+1 positions apart. There are two possible cases:

- **Case 1:** If vwx does not contain any 2s, then vx consists only of 0s and 1s. This results in uwy having n 2s but fewer than n 0s or 1s, leading to a contradiction.
- **Case 2:** If vwx does not contain any 0s, this also leads to a contradiction.

Therefore, this proves that L is not a context-free language.

12.2.Various Properties of context free languages: (CFL):

Closure properties :

The CFLs are closed under some specific operation, closed means after doing that operation on a CFL the resultant language will also be a CFL .Some such operation are:

1. Union Operation
2. Concatenation
3. Kleene closure
4. Reversal operation
5. Homomorphism
6. Inverse Homomorphism
7. Substitution
8. init or prefix operation

9. Quotient with regular language
10. Cycle operation
11. Union with regular language
12. Intersection with regular language
13. Difference with regular language

CFLs are not closed under certain operations, meaning that applying these operations to a CFL can result in a language that is no longer context-free. Some such operations are:

1. Intersection
2. Complement
3. Subset
4. Superset
5. Infinite Union
6. Difference, Symmetric difference (xor, Nand, nor or any other operation which get reduced to intersection and complement)

Decision Properties:

1. Test for Membership: Decidable.
2. Test for Emptiness: Decidable
3. Test for finiteness: Decidable

Rest the decision properties are undecidable in context-free language.

Deterministic property :

The context-free language can be:

1. DCFL-Deterministic (which can be recognized by deterministic pushdown automata) context-free language
2. NDCFL-Non-deterministic (can't be recognized by DPDA but NPDA) context free language.

12.3 Check your progress:

Module-IV Pushdown Automata Turing Machines and Linear Bounded Automata

(Unit-13: Pushdown Automata)

13.0. Introduction:

We have previously covered finite automata. However, finite automata can only recognize regular languages. A Pushdown Automaton (PDA) is essentially a finite automaton enhanced with additional memory, called a stack, which allows it to recognize Context-Free Languages.

13.1.Definition of Pushdown Automata:

A PDA can be defined as :

- Q represents the collection of states.
- ∑ refers to the set of possible input symbols.
- Γ stands for the set of stack symbols, which can be added to or removed from the stack.
- q0 is the designated starting state.
- Z is the initial stack symbol, which is placed at the bottom of the stack initially.
- F is the set of accepting states.

13.2. Acceptance by PDA: PDA and CFL

Pushdown automata accepts context free language.To describe the change of states in a PDA while processing a string, we use the notation ⊢ (turnstile notation). The turnstile notation means one move.

⊢* sign represents a sequence of moves.

Eg- (p, b, T) ⊢ (q, w, α)

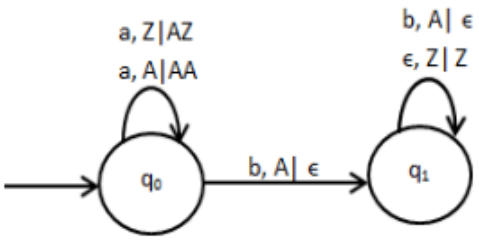
This means that during the transition from state p to state q, the input symbol 'b' is processed, and the stack's top symbol 'T' is replaced with a new string 'α'.

Example : Define the pushdown automata for language {anbn | n > 0}
Solution : M = where Q = { q0, q1 } and Σ = { a, b } and Γ = { A, Z } and δ is given by :
δ(q0, a, Z) = { (q0, AZ) }
δ(q0, a, A) = { (q0, AA) }
δ(q0, b, A) = { (q1, ∈) }
δ(q1, b, A) = { (q1, ∈) }
δ(q1, ∈, Z) = { (q1, ∈) }

Let us see how this automata works for aaabbb.

Row	State	Input	δ(Transition Function)	Stack(Leftmost symbol represents top of stack)	State after move
1	q0	aaabbb		Z	q0
2	q0	aaabbb	δ(q0,a,Z)={(q0,AZ)}	AZ	q0
3	q0	aaabbb	δ(q0,a,A)={(q0,AA)}	AAZ	q0
4	q0	aaabbb	δ(q0,a,A)={(q0,AA)}	AAAZ	q0
5	q0	aaabbb	δ(q0,b,A)={(q1, ϵ)}	AAZ	q1
6	q1	aaabbb	δ(q1,b,A)={(q1, ϵ)}	AZ	q1
7	q1	aaabbb	δ(q1,b,A)={(q1, ϵ)}	Z	q1
8	q1	ϵ	δ(q1,ϵ,Z)={(q1, ϵ)}	ϵ	q1

Push down automat state diagram:



Note:

1. A pushdown automaton is considered deterministic if, for each combination of input symbol and stack symbol, there is exactly one possible transition from a given state.
2. In contrast, a non-deterministic pushdown automaton can have multiple transitions from the same state for a specific input symbol and stack symbol.

3.Converting a non-deterministic pushdown automaton into a deterministic one is not always possible.

4. Non-deterministic pushdown automata are more powerful in terms of expressive capability than deterministic ones. Some languages can be recognized by non-deterministic PDAs but not by deterministic PDAs, which will be discussed in further detail in the next article.

5. Pushdown automata can be designed with either empty stack acceptance or final state acceptance, and it is possible to convert between these two types of acceptance mechanisms.

Pushdown Automata Acceptance by Final State:

We have previously covered Pushdown Automata (PDA) and its acceptance by an empty stack. Now, we will explore how a PDA can accept a context-free language (CFL) by reaching a final state.

Given a PDA P as: P = (Q, Σ, Γ, δ, q0, Z, F), the language accepted by P consists of all strings for which the PDA transitions from the initial state to a final state, regardless of the remaining symbols on the stack. This can be expressed as:

$L(P) = \{w \mid (q0, w, Z) \Rightarrow (qf, \varepsilon, s)\}$

Here, from start state q0 and stack symbol Z, the final state qf ε F is reached when input w is consumed. The stack can contain a string s which is irrelevant as the final state is reached and w will be accepted.

Example: Define the pushdown automata for language $\{a^n b^n \mid n > 0\}$ using final state.
Solution: M = where Q = {q0, q1, q2, q3} and ∑ = {a, b} and Γ = { A, Z } and F={q3} and δ is given by:

δ(q0, a, Z) = { (q1, AZ) }

δ(q1, a, A) = { (q1, AA) }

δ(q1, b, A) = { (q2, ε) }

δ(q2, b, A) = { (q2, ε) } δ(q2, ε, Z) = { (q3, Z) }

Let us see how this automaton works for aaabbb:

Row	State	Input	δ(Transition Function)	Stack(Leftmost symbol represents top of stack)	State after move
1	q0	aaabbb		Z	
2	q0	aaabbb	δ(q0,a,Z)={(q1,AZ)}	AZ	q1
3	q1	aaabbb	δ(q1,a,A)={(q1,AA)}	AAZ	q1
4	q1	aaabbb	δ(q1,a,A)={(q1,AA)}	AAAZ	q1
5	q1	aaabbb	δ(q1,b,A)={(q2, ϵ)}	AAZ	q2

6	q2	aaabbb	δ(q2,b,A)={(q2, ϵ)}	AZ	q2
7	q2	aaabbb	δ(q2,b,A)={(q2, ϵ)}	Z	q2
8	q2	ϵ	δ(q2,ϵ,Z)={(q3, ϵ)}	Z	q3

Explanation: Initially, the state of automata is q0 and symbol on the stack is Z and the input is aaabbb as shown in row 0. On reading a (shown in bold in row 1), the state will be changed to q1 and it will push symbol A on the stack. On next a (shown in row 2), it will push another symbol A on the stack and remain in state q1. After reading 3 a's, the stack will be AAAZ with A on the top. After reading b (as shown in row 4), it will pop A and move to state q2 and the stack will be AAZ. When all b's are read, the state will be q2 and the stack will be Z. In row 7, on input symbol ε and Z on the stack, it will move to q3. As the final state q3 has been reached after processing input, the string will be accepted. This type of acceptance is known *as acceptance by the final state.* Next, we will see how this automata works for aab:

Row	State	Input	δ(Transition Function)	Stack(Leftmost symbol represents top of stack)	State after move
1	q0	aab		Z	
2	q0	aab	δ(q0,a,Z)={(q1,AZ)}	AZ	q1
3	q1	aab	δ(q1,a,A)={(q1,AA)}	AAZ	q1
4	q1	aab	δ(q1,b,A)={(q2, ϵ)}	AZ	q2
5	q2	ϵ		AZ	

As we can see in row 4, the input has been processed and PDA is in state q2 which is a non-final state, the string aab will not be accepted.

13.3.Parsing and pushdown automata:

Parsing is done to derive a string using the production rules of a formal grammar. Parsing is widely used to check the acceptability of a string . Compilers are used to check syntactical correctness of a string. A parser takes the inputs and a parse tree is built.

A parser can be of two types −

Top-Down Parser − Top-down parsing starts from the top with the start-symbol and derives a string using a parse tree.
Bottom-Up Parser − Bottom-up parsing starts from the bottom with the string and comes to the start symbol using a parse tree.

12.3.Summary:

i) . Pushdown Automata is a type of finite automata with additional memory called stack.

ii) Acceptance of context free language can be proved by two ways: acceptance by empty stack and acceptance by final states.

12.4.Check your progress:

1. What is a Pushdown Automaton (PDA)? How does it differ from a Finite Automaton (FA) in terms of capabilities and structure?
2. List and explain the components of a Pushdown Automaton. How does the stack in a PDA contribute to its computational power?
3. Explain the two modes of acceptance for a Pushdown Automaton: acceptance by final state and acceptance by empty stack. How do they differ, and are they equivalent in terms of the languages they accept?
4. How is a Pushdown Automaton related to Context-Free Languages (CFLs)?
5. What are some languages that cannot be recognized by a Pushdown Automaton? Give an example of a language that is not context-free and explain why a PDA cannot recognize it.
6. Construct PDA for $L = \{a^{(2*m)}c^{(4*n)}d^{n}b^{m} \mid m,n \geq 0\}$

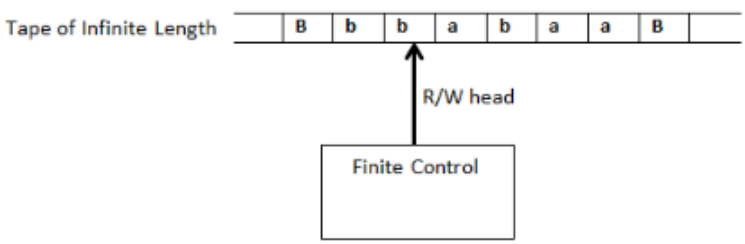
Module-IV Pushdown Automata Turing Machines and Linear Bounded Automata

(Unit-14: Turing Machine)

14.0.Introduction:

Turing machine is an important tool for studying the limits of computation and for understanding the foundations of computer science. It provides a simple yet powerful model of computation that has been widely used in research and has had a profound impact on our understanding of algorithms and computation. Turing Machines are used to accept Recursive Enumerable Languages (generated by Type-0 Grammar).

14.1. Turing machine model:



Block Diagram of Turing Machine

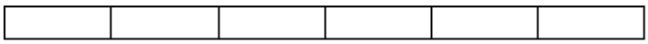
The different components of the Turing Machine are:

1. A two way tape of infinite length.
2. The header can read from the tape as well as erase and write into the tape .It can move both to the left and right .
3. The finite control unit which governs the machine's transition.

14.2. Representation of Turing machine:

14.3. Language acceptability by Turing machine: The Turing Machine is used to accept language generated by the Type-0 Grammar which is known as Recursive Enumerable Languages**.** Type-0 grammars include all formal grammars.

14.4.



By the move δ(q3, B) = halt, the transition will stop and the string is accepted.

Note:

- In a non-deterministic Turing Machine (TM), there may be multiple possible transitions for a given state and tape symbol, but this does not enhance the computational capability of the machine.

- Any non-deterministic TM can be transformed into a deterministic TM.
- In a multi-tape Turing Machine, there are multiple tapes and corresponding head pointers, yet this configuration does not increase the machine's computational power.
- A multi-tape TM can always be converted into an equivalent single-tape TM.

14.5. Universal Turing Machine and other modifications**:**

The single tape Turing machine is the most common type of TM. But with time, computer scientist were trying to modify this single tape Turing machine in search of more computational power. But these modified versions were failed attempts to increase the computational power of TM, only speed up in terms of processing could be achieved with these versions.

On the basis of determinism of the next transition to a state on a particular input, Turing Machines are devided into two types: Deterministic turing machine and non-deterministic Turing machine.In non-deterministic Turing Machine,there may be more than one possible move for a given state and tape symbol, but and non-deterministic Turing machine does not add any power.

Multi R/W head points TM: It does not add any computational power to the TM.

i) Multi-dimensional TM: It does not add any computational power to the TM.

ii) TM with one way infinite tape: It does not add any computational power to the TM.

Universal Turing Machine

TM is a hypothetical model of a digital computer. Digital computers are of two types based on their architecture: General Purpose Computer and Fixed Programme Computer. The concept of stored programme was implemented in general purpose computer which means the running of stored programme. The TM that we have designed sub-unit 14.4 is actually a fixed progamme computer. Universal Turing machine is an attempt to simulate the concept of stored programme.

	Codes	a	b	c	B	

Figure- Block diagram of UTM

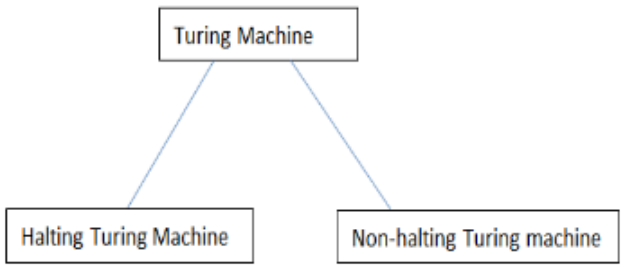
In Universal Turing machine,by reading the code, the machine knows how to behave on a state and there is no need of finite control unit. For example, to add two binary numbers and to detect a language we can write codes and store on the tape. By doing this, we have skipped the process different TM designings (which are like fixed programme computer.).

14.6. Halting problem of Turing machine**:**

In Finite automata and Pushdown automata, we have the concept of halting in two cases:

i)if the automaton halts on final state, it means than the string is accepted.
ii)if the automaton halts on non-final states, it means that the string is rejected.

In Turing machine, along with these two possibilities, there is a third possibility-the machine may go into infinite loop.

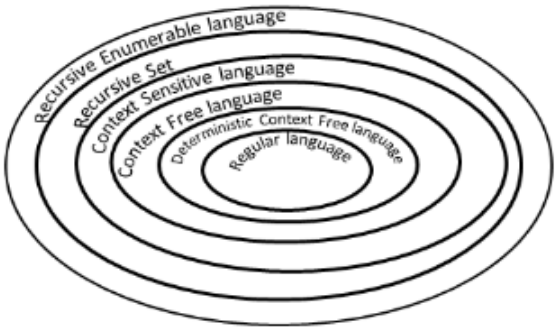


In case of halting Turing Machine it is guaranteed that the machine will halt. If it halts on final state then the string is accepted and if halts on none-final state then the string is rejected.

. Therefore, broadly recursively enumerable languages are categorized as of two types:

i) Recursive Set: The language L , which is accepted by TM, is said to be recursive set, where for all 'w' that belong to L, the machine will go to final halt, and for all 'w' that do not belong to L, the machine will go to none- final halt. Here the TM gives guarantee of halting. Hence membership property and non-membership property is decidable here. So this language is also called Turing decidable language.

ii) Recursively Enumerable Set: The language L , which is accepted by TM, is said to be recursively enumerable set, where for all 'w' that belong to L, the machine will go to final halt, and for all 'w' that do not belong to L, the machine will either go to non- final halt or infinite loop. Hence non-



membership property is not decidable, only membership property is decidable here. This language is also called Turing recognizable language.

Formally, the halting problem is as follows:

"Given a description of a computer program P and its input I, is there an algorithm that can decide whether P will eventually halt (terminate) or run forever on input I?"

14.7.Summary:

i) Turing machine is the threshold for modern day's computers.

ii) Turing machine may go into infinite loop.

iii) Recursive set language is also called Turing decidable language.

iv) Recursively Enumerable Set language is also called Turing recognizable language.

14.8.Check your progress:

1.What is a Turing Machine (TM)? Describe its basic components.

2.Explain the halting problem of Turing Machine.

3.Can a Turing Machine simulate other types of machines, such as a Finite Automaton or a Pushdown Automaton? Explain how a Turing Machine can perform such simulations.

4.What is a Universal Turing Machine (UTM)? How does it differ from a regular Turing Machine, and what is its significance in computation theory?

5.Provide a brief explanation of why the Halting Problem is undecidable.

Module-IV Pushdown Automata Turing Machines and Linear Bounded Automata

(Unit-15: Linear Bounded Automata)

15.0.Introduction:

Linear bounded automata(LBA) is considered as a restricted form of Turing Machine. It is a bounded automaton is a multi-track non-deterministic Turing machine with a tape of some bounded finite length. Linear bounded automata accepts context sensitive language.

LBA can be defined with eight tuple as: M = (Q , T , E , q0 , ML , MR , S , F), where,
Q -> A finite set of transition states
T -> Tape alphabet
E -> Input alphabet
q_0 -> Initial state
M_L -> Left bound of tape
M_R -> Right bound of tape
S -> Transition Function
F -> A finite set of final states

15.1. Model of LBA :

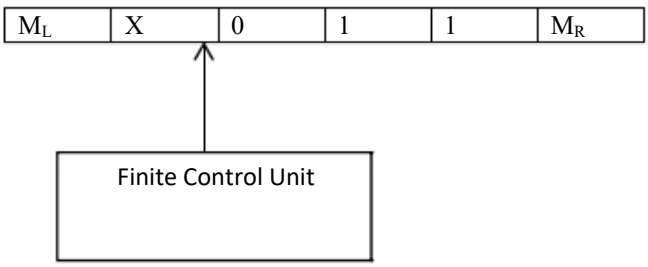


Figure- Block diagram of LBA

Examples:
Languages that form LBA with tape as shown above,

- $L = \{a^{n!} \mid n >= 0\}$

- L = {wn | w from {a, b}+, n >= 1}
- L = {wwwR | w from {a, b}+}

15.2.Summary:

i) Linear bounded automata(LBA) is a restricted form of Turing Machine.
ii) Linear bounded automata accepts context sensitive language.
iii) The length of tape in linear bounded automata is finite.

15.3.Check your progress:

1. How does a linear bounded automata differ from a Turing Machine?
2. Describe the components of a Linear Bounded Automaton. How does the use of a tape of limited size affect the computational power of an LBA?
3. Can every language accepted by an LBA be accepted by a Turing Machine?
4. Can a Linear Bounded Automaton simulate a Turing Machine? If so, under what conditions? Explain the limitations and strengths of LBAs compared to Turing Machines.
5. Provide an example of a language that can be accepted by a Linear Bounded Automaton but not by a finite automaton.
6. Linear Bounded Automaton is a:

   (a) Finite Automaton

   (b) Turing Machine

   (c) Push down Automaton

   (d) None of the mentioned
