Date __/__/____

THEORY OF COMPUTATION

Automata theory (also known as Theory of computation) is a theoritical branch of computer science and Mathematics, which mainly deals with the logic of computation with respect to simple machines, referred to as automata.

Applications of Theory of computation -

- Traffic lights.
- Lifts and elevators.
- Marketing.
- Compilers.
- cloud computing.

Date __/__/____

1. Mathematical Preliminaries

Sets – A set is a collection of well defined objects. The objects of a set are taken as distinct only.

$A = \{1, 2, 3, 4, 5, \ldots\}$
$B = \{\}$ or $B = \emptyset$

Relations – Relations are defined from a non-empty set A to non-empty set B is a subset of $A \times B$, where the elements of the ordered pair have a fixed property or relation.

Types of Relations :-

1.) Universal Relation – A relation defined from A to B is said to be an universal relation if

$\boxed{R = A \times B}$

2.) Empty Relation – A relation defined from A to B is said to be an empty relation if

$\boxed{R = \emptyset}$

Date __/__/____

3.) Identity Relation - A relation defined on set B is said to be an identity relation if each and every elements of set B is related to itself only.

$B = \{1, 2, 3, 4\}$
$R = \{(1,1), (2,2), (3,3), (4,4)\}$

4.) Inverse Relation - Inverse relation is seen when a set has elements which are inverse pairs of another set.

$A = \{(a,b), (c,d)\}$
$R^{-1} = \{(b,a), (d,c)\}$

5.) Reflexive Relation - A relation is defined on a set B is said to be reflexive, if ~~$(a,a) \in R \; \forall \; a \in A$~~ $(b,b) \in R \; \forall \; b \in B$.

$B = \{1, 2, 3, 4\}$
$R = \{(1,1), (2,2), (3,3), (4,4), (4,1)\}$

6.) Symmetric Relation - A relation defined on a set B is said to be symmetric relation if for every ordered pair (a,b) in R, there must exist an ordered pair (b,a) on relation $R \; \forall \; a, b \in B$.
$R = \{(1,2), (2,1), (3,4), (4,3)\}$

Page No.

Date __/__/____

7.) Transitive Relation - For transitive relation, if $(x, y) \in R$, $(y, z) \in R$, then $(x, z) \in R$.

8.) Equivalence Relation - If a relation is reflexive, symmetric and transitive at the same time it is known as equivalence relation.

Domain - The set of all first elements of the ordered pair in the relation is called the domain of the relation.

Range - The set of all second elements of the ordered pair in the relation is called the range of the relation.

Functions - A relation defined from a non-empty set A to a non-empty set B, is said to be a function if and only if each and every element of set A has only one unique image in set B.

Types of Functions :-

1.) Identity function - $y = x$

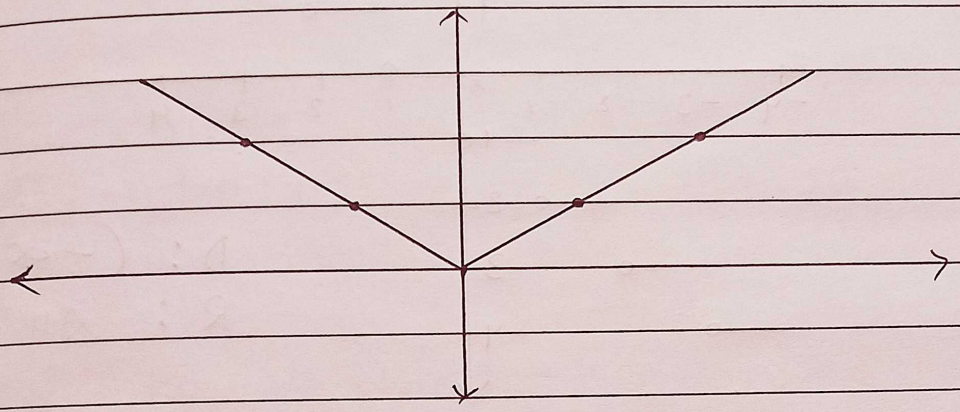
2.) Constant Function - $y = c$

Page No.

Date __/__/____

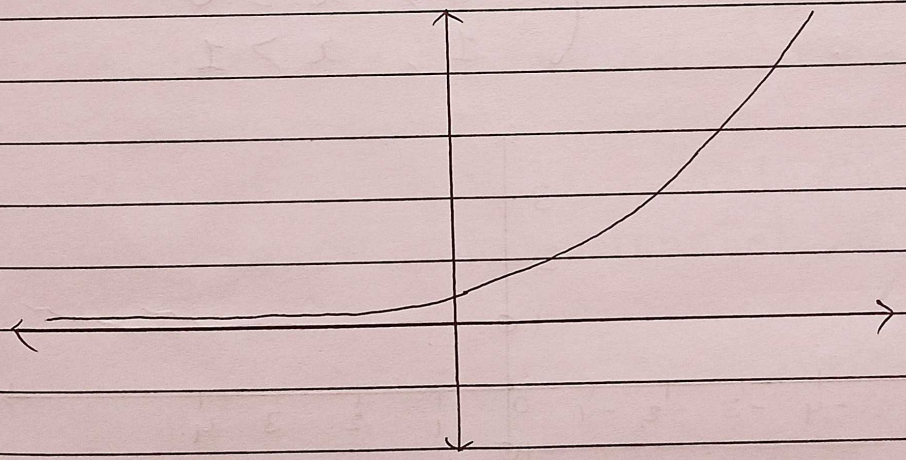
3.) Modulus Function - $y = |x|$

D: $(-\infty, \infty)$ R: $[0, \infty)$



4.) Exponential Function - $y = 5^x$

D: $(-\infty, \infty)$ R: $(0, \infty)$



5.) Greatest Integer Function - $y = [x]$

$2 = [2.5]$ $3 = [3]$

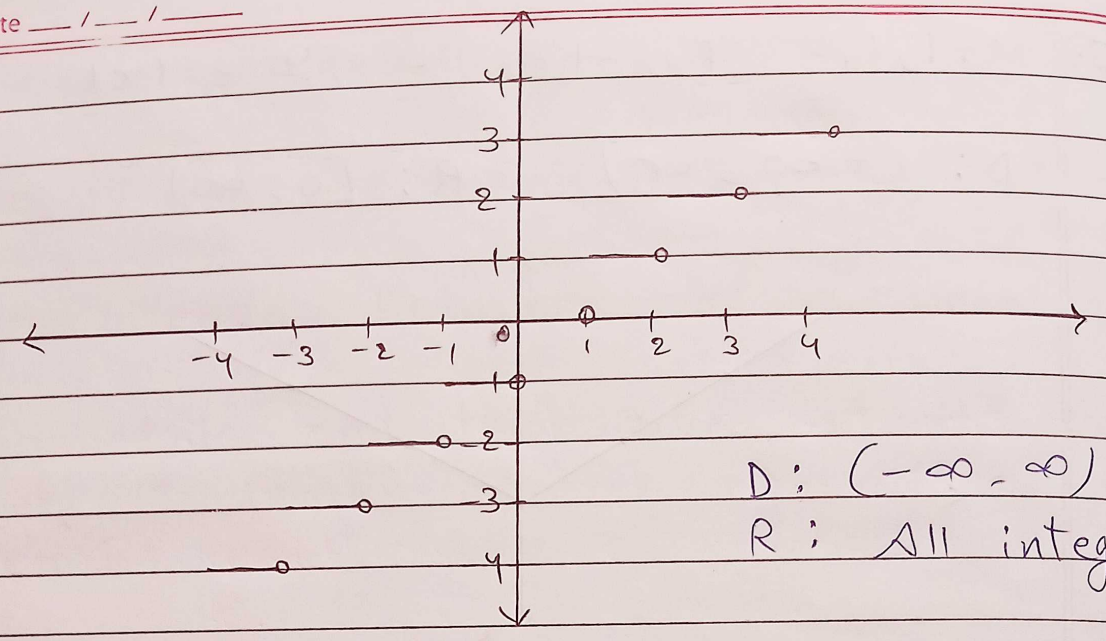
$-1 = [-0.5]$ $0 = [0.1]$

$-1 = [-1]$ $1 = [1]$

$-2 = [-1.5]$ $1 = [1.5]$

$-2 = [-2]$ $2 = [2]$

Date __/__/____

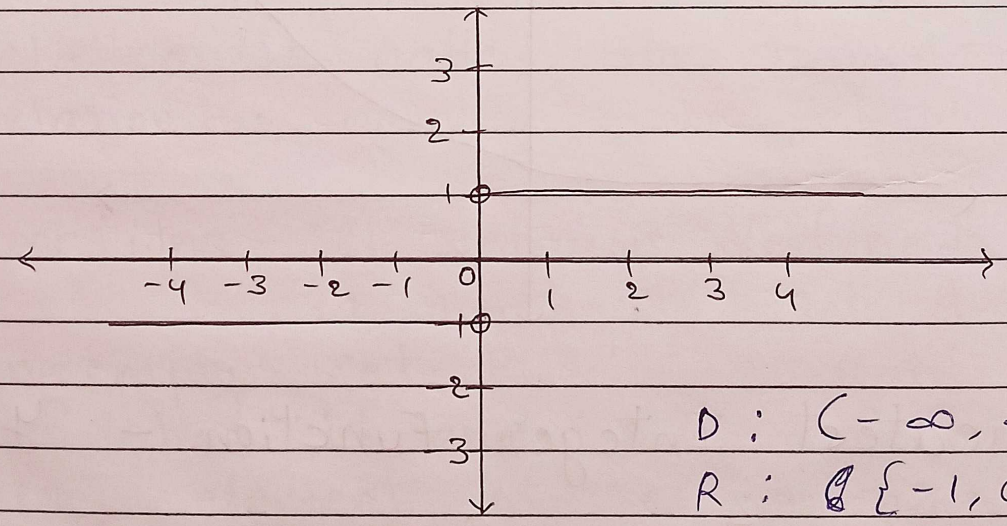


D: $(-\infty, \infty)$

R: All integers

6.) Signum Function -

$$y = \frac{|x|}{x} = \begin{cases} -1, & x < 1 \\ 0, & x = 0 \\ 1, & x > 1 \end{cases}$$



D: $(-\infty, \infty)$

R: $\{-1, 0, 1\}$

7.) Polynomial Functions -

$y = x^2$

$y = x^3 + x^2 + 1$

$y = x^4 + x^3 + 2$

Date __/__/____

Strings - A string is a finite sequence of symbols selected from some alphabet. It is generally denoted as w.

- For example for alphabet $\Sigma = \{0, 1\}$
  $w = 010101$ is a string
- Length of a string is denoted as $|w|$ and is defined as the number of positions for the symbol in the string. For the above example length is 6.
- The empty string is the string with zero occurrence of symbols. This string is represented as ϵ or λ.
  $|w| = 0$
- The set of strings, including the empty string, over an alphabet Σ is denoted by Σ^*.

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

$\Sigma^* = \{\epsilon, 0, 1, 01, 10, 00, 11, 101, 1010101, \ldots\}$

Graph and Trees - Pre Notes.

Principle of Induction - It is a way of proving that P(n) is true for all integers $n \geq a$.

It works in two steps : (a) [Base case]

Page No.

Date __/__/____

Prove that P(a) is true.

(b) [Inductive step] Assume that P(k) is true for some integer $k \geq a$, and use this to prove that P(k+1) is true.

Example : Show that $1+3+5+\ldots+(2n-1)=n^2$

step 1 : Result is true for $n=1$

i.e. $1=(1)^2$ (True)

step 2 : Assume that result is true for $n=k$

$1+3+5+\ldots+(2k-1)=k^2$

step 3 : check for $n=k+1$

i.e. $1+3+5+\ldots+(2k-1)+(2(k+1)-1)=(k+1)^2$

using step 2 result, we get

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

L.H.S and R.H.S are same.

So the result is true for $n=k+1$

By P.M.I this statement is true for all natural numbers n.

Page No.

Date __/__/____

2. Propositions and Predicates

Proposition or statements – Pre Notes.

Propositional connectives – Pre Notes.

Well-formed formulae – Pre Notes.

Tautology – If the outputs of all inputs in a truth table are true is called Tautology.

Predicates – In logic, a predicate is a symbol which represents a property or a relation.

A predicate is an expression of one or more variables defined on some specific domain.

Example:
- Let $E(x, y)$ denote "$x = y$"
- Let $X(a, b, c)$ denote "$a + b + c = 0$"
- Let $M(x, y)$ denote "x is married to y"

Quantifiers – The variable of predicates is quantified by quantifiers.

Types of Quantifiers in Predicate logic:-

1.) Universal Quantifier – Universal Quantifier states that the

Date __/__/____

statements within its scope are true for every value of the specific variable. It is denoted by the symbol $\forall$.

Example :- "Man is mortal"

$\forall x\, P(x)$ where $P(x)$ is the predicate which denotes x is mortal and the universe of discourse is all men.

2.) Existential Quantifier - Existential Quantifier states that the statements within its scope are true for some values of the specific variable. It is denoted by the symbol $\exists$.

Example :- "Some people are dishonest"

$\exists x\, P(x)$ where $P(x)$ is the predicate which denotes x is dishonest and the universe of discourse is some people.

Date __/__/____

3. Theory of Automata

Automata theory is the study of abstract machines and automata, as well as the computational problems that can be solved using them.

Finite Automata – Finite Automata is an abstract ~~machine.~~ computing device. It is a mathematical model of a system with discrete inputs, outputs, states and set of transitions from state to state that occurs on input symbols from alphabet Σ.

It's representation –

1) Graphical
2) Tabular
3) Mathematical

→ A finite automata is a 5-tuples, they are
$M = (Q, \Sigma, \delta, q_0, F)$
where,
Q: is a finite set called the states
Σ: is a finite set called the alphabets
δ: $Q \times \Sigma \rightarrow Q$ is the transition function
$q_0 \in Q$ is the initial state
$F \subseteq Q$ is called the final state

Page No.

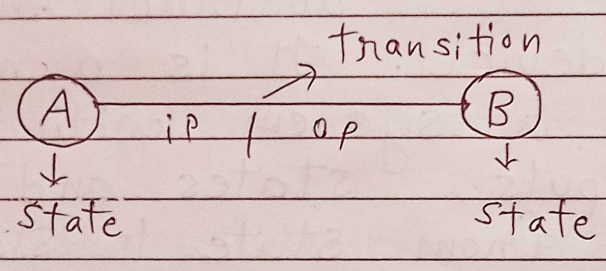
Date __/__/____

Transition system - It is directed labelled graph in which

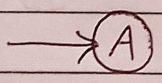
(i) Each vertex (or node) represent states.

(ii) Directed edge between states represent transition.

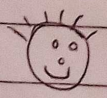
(iii) Each directed edge is labelled with i/p & o/p.



1.) Initial state - A state that has incoming arrow without any state previous to it.



2.) Final state - A state that is enclosed within double circle.



Note :- Transition system is also known as Transition Graph.

Page No.

Date __/__/____

Acceptability of a String - A string is said to be accepted if there is dedicated path from start state to final state with given string.

Acceptability of a string by Finite Automata -

A string ω is accepted by a finite automaton, $M = (Q, \Sigma, \delta, q_0, F)$, if $\delta(q_0, \omega) = F$, where q_0 is initial state and F is the final state. For every input symbol of string it takes one transition and starts with initial state and reaches final state by reading the complete string.

Deterministic Finite Automaton (DFA) -

A finite automata is said to be deterministic, if corresponding to an input symbol, there is single resultant state i.e. there is only one transition.
A deterministic finite automata is set of five tuples and represented as,
$M = \{Q, \Sigma, \delta, q_0, F\}$

Q: A non empty finite set of states present in the finite control ($q_0, q_1, q_2, \ldots$).
Σ: A non empty finite set of inputs symbols.
δ: It is a transition function that ~~makes~~

Page No.

Date __/__/____

takes two arguments, a state and an input symbol, it returns a single state.

q_0 : It is starting state, one of the state in Q.

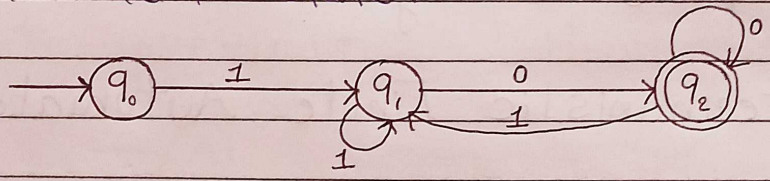
F: It is non-empty set of final states/ accepting states from the set belonging to Q.

Example :

Q.) Design a FA with $\Sigma = \{0, 1\}$ accepts those string which starts with 1 and ends with 0.

=> Solution:

The FA will have a start state q_0 from which only the edge with input 1 will go to the next state.



Non-Deterministic Finite Automaton (NFA) -

A Finite Automata (FA) is said to be non deterministic, if there is more than one possible transition from one state on the same input symbol.

A NFA is also set of five tuples and represented as,

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

Q: A set of non empty finite states.

Page No.

Date __/__/____

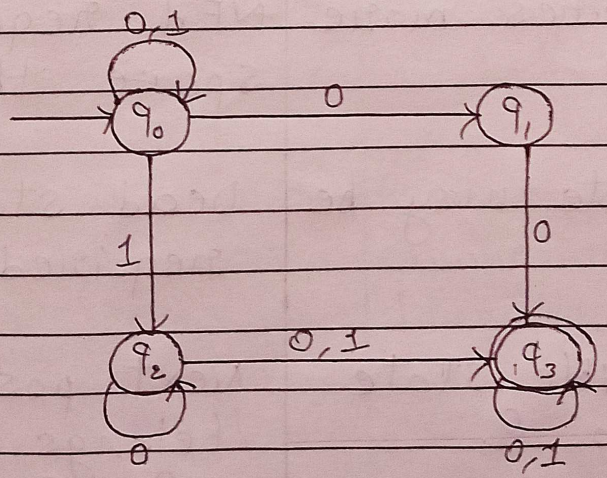
Σ: A set of non empty finite input symbols.
δ: It is a transition function that takes a state from Q and an input symbol from and returns a subset of Q.
q_0: Initial state of NFA and member of Q.
F: A non empty set of final states and member of Q.

Example:

Q.) Design a NFA for the transition table as given below:

Present state	0	1
$\rightarrow q_0$	q_0, q_1	q_0, q_2
q_1	q_3	ε
q_2	q_2, q_3	q_3
$\rightarrow q_3$	q_3	q_3

⇒ Solution:
The transition diagram can be drawn by using the mapping function as given in the table.



Page No.

Date __/__/____

Difference between DFA and NFA :-

	DFA	NFA
(i)	DFA cannot use Empty string transition.	NFA can use Empty string transition.
(ii)	DFA can be understood as one machine.	NFA can be understood as multiple little machines computing at the same time.
(iii)	In DFA, the next possible state is distinctly set.	In NFA, each pair of state and input symbol can have many possible next states.
(iv)	DFA is more difficult to construct.	NFA is easier to construct.
(v)	All DFA are NFA.	Not all NFA are DFA
(vi)	DFA requires more space.	NFA requires less space than DFA.
(vii)	Dead state may be required.	Dead state is not required.
(viii)	next possible state belongs to Q.	Next possible state belongs to power set of Q.

Page No.

Date __/__/____

4. Formal Languages

In automata theory, a formal language is a set of strings of symbols drawn from a finite alphabet.
A formal language can be specified either by a set of rules (such as regular expressions or a context-free grammar) that generates the language, or by a formal machine that accepts the language.

Chomsky classification of Languages :-

A/c to Noam chomosky, there are four types of grammars -

Type 0, Type 1, Type 2, Type 3.

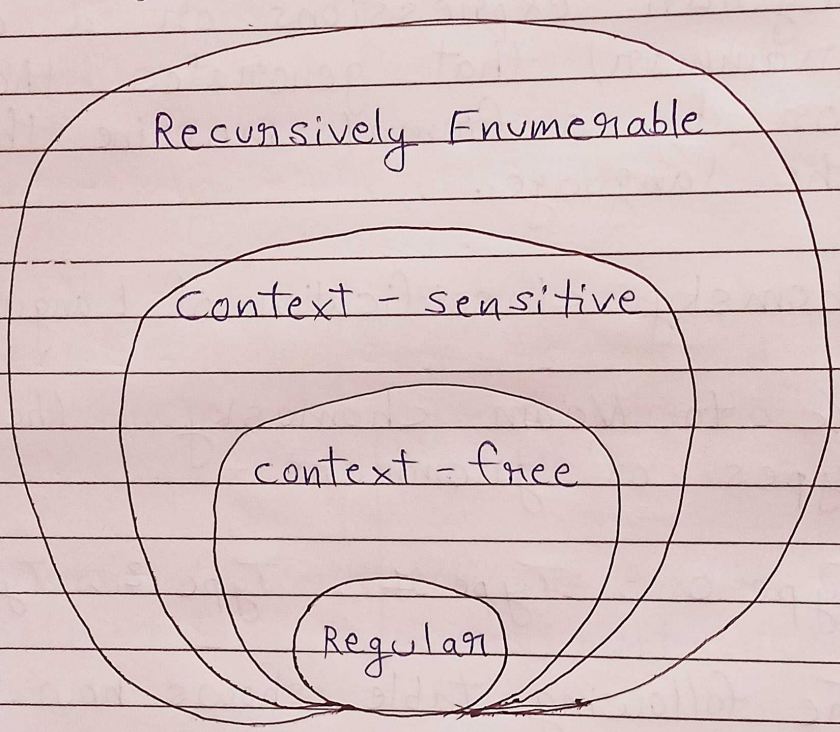
→ The following table shows how they differ from each other -

Grammar Type	Grammar Accepted	Language Accepted	Automaton
Type 0	unrestricted grammar	Recursively enumerable language	Turing Machine
Type 1	context-sensitive grammar	context-sensitive language	Linear bounded automaton

Date __/__/____

Type 2	context-free grammar	context-free language	Pushdown automaton
Type 3	Regular grammar	Regular language	Finite state automaton

→ The following illustration shows the scope of each type of grammar -



1.) Type - 0 Grammar - It generates recursively enumerable languages. The productions have no restrictions. They are any phase structure grammar including all formal grammars.

They generate the languages that are recognized by a Turing machine. The production can be in the form of $\alpha \rightarrow \beta$

Date __/__/____

example: $S \rightarrow ACaB$
$Bc \rightarrow acB$
$CB \rightarrow DB$
$aD \rightarrow Db$

2.) Type - 1 Grammar - It generates context-sensitive languages.
The production must be in the form
$\alpha A \beta \rightarrow \alpha \gamma \beta$
The strings α and β may be empty, but γ must be non-empty.
The languages generated by these grammars are recognized by a linear bounded automaton.

example: $AB \rightarrow AbBc$
$A \rightarrow bcA$
$B \rightarrow b$

3.) Type - 2 Grammar - It generates context-free languages. The production must be in the form $A \rightarrow \gamma$.
These languages generated by these grammars are be recognized by a non-deterministic pushdown automaton.

example: $S \rightarrow Xa$
$X \rightarrow a$
$X \rightarrow aX$
$X \rightarrow abc$
$X \rightarrow \varepsilon$

Date __/__/____

4.) Type -3 Grammar - It generates regular language. The productions must be in the form $X \to a$ or $X \to aY$. These languages generated by these grammars are be recognized by a Finite state automaton.

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

Alphabet - An alphabet is any finite set of symbols.
(,), ∅, *, +
example: $\Sigma = \{a, b, c, d\}$ is an alphabet set where 'a', 'b', 'c' and 'd' are symbols.

Language - A language is a subset of Σ^* for some alphabet Σ. It can be finite or infinite.

example: If the language takes all possible strings of length 2 over $\Sigma = \{a, b\}$, then $L = \{ab, aa, ba, bb\}$

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

The formal definition of grammar is that it is defined as four tuples -

Page No.

Date __/__/____

$G = (V, T, P, S)$

- G : G is a grammar, which consists of a set of production rules. It is used to generate the strings of a language.
- T : T is a final set of terminal symbols. It is denoted by lower case letters.
- V : V is the final set of non-terminal symbols. It is denoted by capital letters.
- P : P is a set of production rules, which is used for replacing non-terminal symbols in a string with other terminals.
- S : S is the start symbol used to derive the string.

Grammar is composed of two elements :-

1.) Terminal Symbols - They are the components of the sentences that are generated using grammar and are denoted using small case letters like a, b, c etc.

2.) Non-Terminal Symbols - They take part in the generation of the sentence but are not the component of the sentence. These type of symbols

Date __/__/____

are also called Auxiliary Symbols and variables. They are represented using a capital letter like A, B, C etc.

example :

Consider a grammar
$G = (V, T, P, S)$
where,
$V = \{S, A, B\}$ => Non-Terminal Symbols
$T = \{a, b\}$ => Terminal symbols
$P = \{S \to ABa, A \to BB, B \to ab, AA \to B\}$
=> Production rules
$S = \{S\}$ => Start Symbol

Operations on Languages -

Some of the operations on languages are as follows -

- Union
- Intersection
- Difference
- Concatenation
- Kleen * closure

1) Union - If $L1$ and $L2$ are two regular languages, their union $L1 \cup L2$ will also be regular.
eg: $L1 = \{an \mid n > 0\}$ and $L2 = \{bn \mid n > 0\}$
$L3 = L1 \cup L2 = \{an \cup bn \mid n > 0\}$ is also regular.

Page No.

Date __/__/____

2.) Intersection - If L_1 and L_2 are two regular languages, their intersection $L_1 \cap L_2$ will also be regular.

eg: $L_1 = \{a^n \mid n>0\}$ and L_2

eg: $L_1 = \{a^m b^n \mid n>0 \text{ and } m>0\}$ and $L_2 = \{a^m b^n \cup b^n a^m \mid n>0 \text{ and } m>0\}$

$L_3 = L_1 \cap L_2 = \{a^m b^n \mid n>0 \text{ and } m>0\}$ is also regular.

3.) Concatenation - If L_1 and L_2 are two regular languages, their concatenation $L_1 . L_2$ will also be regular.

eg: $L_1 = \{a^m \mid m>0\}$ and $L_2 = \{b^n \mid n>0\}$

$L_3 = L_1 . L_2 = \{a^m . b^n \mid m>0 \text{ and } n>0\}$ is also regular.

4.) Kleen* closure - If L_1 is a regular language, its kleen closure L_1^* will also be regular.

eg: $L_1 = (a \cup b)$, $L_1^* = (a \cup b)^*$

$a^* = \wedge, a, aa, aaa, \ldots$

$(a,b)^* = \{\wedge, a, b, aa, aaa, \ldots, bb, bbb, \ldots\}$

$(a+b) = \{a, b\}$

$(a+b)^* = \{\wedge, a, aa, aaa, \ldots, b, bb, bbb, \ldots\}$

- $\emptyset \rightarrow$ empty set
- $\wedge \rightarrow$ empty string

Page No.

Date __/__/____

5.) Complement – If $L(G)$ is a regular language, its complement $L'(G)$ will also be regular. Complement of a language can be found by subtracting strings which are in $L(G)$ from all possible strings.

eg: $L(G) = \{a^n \mid n > 3\}$
$L'(G) = \{a^n \mid n <= 3\}$

Note :- Two regular expressions are equivalent, if languages generated by them are same. For example, $(a+b^*)^*$ and $(a+b)^*$ generate the same language. Every string which is generated by $(a+b^*)^*$ is also generated by $(a+b)^*$ and vice versa.

Date __/__/____

5. Regular Set and Regular Grammar

Regular Expressions – The language accepted by finite automata can be easily described by simple expressions called Regular Expressions. It is the most effective way to represent any language.

→ A regular expression can be recursively defined as follows –

- ε is a Regular Expression indicates the language containing an empty string. ($L(\varepsilon) = \{\varepsilon\}$)

- φ is a Regular Expression denoting an empty language. ($L(\varphi) = \{\}$)

- X is a Regular Expression where $L = \{x\}$

- If X is a Regular Expression denoting the language $L(X)$ and Y is a Regular Expression denoting the language $L(Y)$, then :

→ X+Y is a Regular Expression corresponding to the language $L(X) \cup L(Y)$ where $L(X+Y) = L(X) \cup L(Y)$.

→ X.Y is a Regular Expression corresponding

Date __/__/____

to the language $L(x) \cdot L(y)$ where $L(x \cdot y) = L(x) \cdot L(y)$

→ R^* is a Regular Expression corresponding to the language $L(R^*)$ where $L(R^*) = (L(R))^*$

If we apply any of the rules several times from 1 to 5, they are Regular Expressions.

examples:-

Regular Expressions	Regular Set
$(0+10^*)$	$L = \{0, 1, 10, 100, 1000, 10000, \ldots\}$
$(0^* 10^*)$	$L = \{1, 01, 10, 010, 0010, \ldots\}$
$(0+\varepsilon)(1+\varepsilon)$	$L = \{\varepsilon, 0, 1, 01\}$
$(a+b)^*$	$L = \{\varepsilon, a, b, aa, ab, bb, ba, aaa, \ldots\}$
$(a+b)^*abb$	$L = \{abb, aabb, babb, aaabb, ababb, \ldots\}$
$(11)^*$	$L = \{\varepsilon, 11, 1111, 111111, \ldots\}$
$(aa)^*(bb)^*b$	$L = \{b, aab, aabbb, aabbbbb, aaaab, aaaabbb, \ldots\}$
$(aa+ab+ba+bb)^*$	$L = \{aa, ab, ba, bb, aaab, aaba, \ldots\}$

Date __/__/____

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

Pumping Lemma for regular sets :-

Theorem -

Let L be a regular language. Then there exists a constant 'c' such that for every string w in L -

$|w| \geq c$

we can break w into three strings, $w = xyz$, such that -

$|y| > 0$

$|xy| \leq c$

$\forall\ k \geq 0$, the string xy^kz is also in L.

* Applications of Pumping Lemma -

Pumping Lemma is to be applied to show that certain languages are not regular. It should never be used to show a language is regular.

→ If L is regular, it satisfies Pumping Lemma.

→ If L does not satisfy Pumping Lemma, it is non-regular.

Date __/__/____

* Method to prove that a language L is not regular -

- At first we have to assume that L is regular.
- So, the pumping lemma should hold for L.
- use the pumping lemma to obtain a contradiction -

→ select w such that $|w| \geq c$
→ select y such that $|y| \geq 1$
→ select x such that $|xy| \leq c$
→ Assign the remaining string to z.
→ select k such that the resulting string is not in L.
Hence L is not regular.

Q.) Prove that $L = \{a^i b^i \mid i \geq 0\}$ is not regular.
⇒ solution:
At first, we assume that L is regular and n is the number of states.

Let $w = a^n b^n$, Thus $|w| = 2n > n$.

By pumping lemma, let $w = xyz$, where $|xy| \leq n$.

Let $x = a^p$, $y = a^q$ and $z = a^r b^n$, where $p+q+r = n$, $p \neq 0$, $q \neq 0$, $r \neq 0$. Thus $|y| \neq 0$.

Date __/__/____

Let $k=2$. Then $xy^2z = a^p a^{2q} a^r b^n$.

Number of as $= (p+2q+r) = (p+q+r)+q = n+q$

Hence, $xy^2z = a^{n+q} b^n$, Since $q \neq 0$, xy^2z is not of the form $a^n b^n$.

Thus, xy^2z is not in L. Hence L is not regular.

closure properties of regular set -

Any set that represents the value of the Regular Expression is called Regular set.

→ The closure of a regular set is regular.

Proof -
If $L = \{a, aaa, aaaaa, \ldots\}$ (strings of odd length excluding NULL)

i.e., $RE(L) = a(aa)^*$
$L^* = \{a, aa, aaa, aaaa, aaaaa, \ldots\}$
(strings of all lengths excluding NULL)

$RE(L^*) = a(a)^*$

Hence proved.

Date __/__/____

Regular set - Any set that represents the value of the Regular Expression is called Regular set.

eg: $L = \{\varepsilon, 11, 1111, 111111, \ldots\}$
Here L is language set of even number of 1's.

Regular Grammar - A grammar is regular if it has rules of form $A \to a$ or $A \to aB$ or $A \to \varepsilon$ where ε is a special symbol called NULL.

example: $G = \{a, b\}$, $V = \{S\}$ and $P = \{S \to aS, S \to bS, S \to \}$
is a regular grammar and it generates all the strings consisting of a's and b's including the empty string.

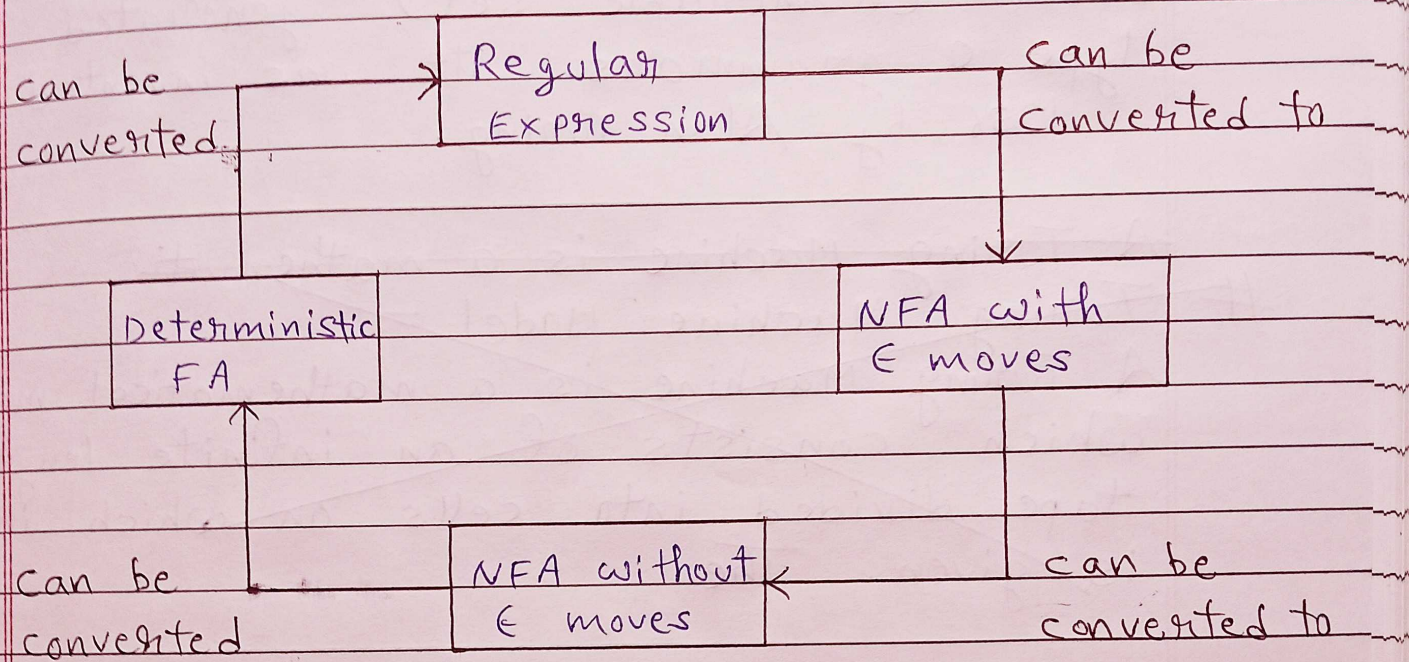
Regular Languages - A language is regular if it can be expressed in terms of regular expression.
Every finite set represents a regular language.
example: $\{a, b\}^*$ i.e. $L = \{aa, ab, ba, bb\}$
Given an expression of non-regular language but the value of parameter is bounded by some constant i.e $|L| = \leq z$, then the language is regular. (finite comparision)

Page No.

Date __/__/____

5. Regular set and Regular grammar

Relationship between Finite Automata (FA) and Regular Expression (RE) –



The above figure explains that it is easy to convert

- RE to NFA with epsilon (∈) moves.
- NFA with epsilon moves to without epsilon moves.
- NFA without epsilon moves to DFA.
- DFA can be converted easily to RE.

Date __/__/____

6. Context-free languages

Context-Free Language (CFL) is a language which is generated by a context-free grammar or type-2 grammar according to chomsky classification and gets accepted by a Pushdown Automata.

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

Derivation Trees - Derivation tree is a graphical representation for the derivation of the given production rules of the context free grammar (CFG).

It is a way to show how the derivation can be done to obtain some string from a given set of production rules. It is also called as the Parse Tree.

The Parse tree follows the precedence of operators.

The deepest subtree is traversed first. So, the operator in the parent node has less precedence over the operator in the subtree.

Date __/__/____

Properties of Derivation Trees -

The root node is always a node indicating the start symbols.

The derivation is read from left to right.

The leaf node is always the terminal node.

The interior nodes are always the non-terminal nodes.

* Example:

The Production rules for the derivation tree are as follows -

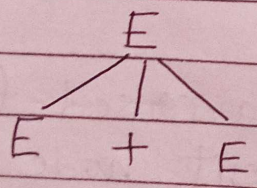
E = E + E

E = E * E

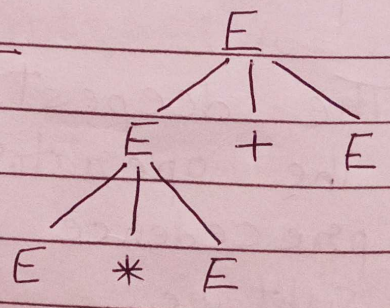
E = a|b|c

Here, Let the input be a*b+c

Step 1

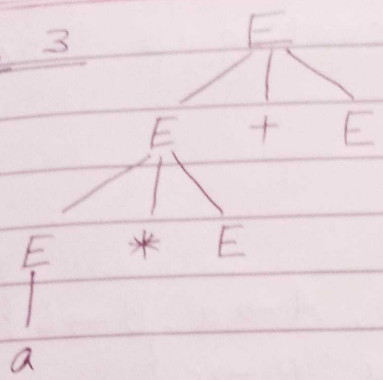


Step 2

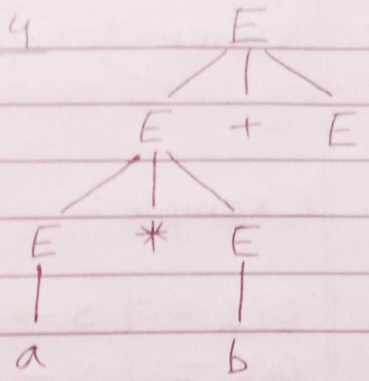


Date __/__/____

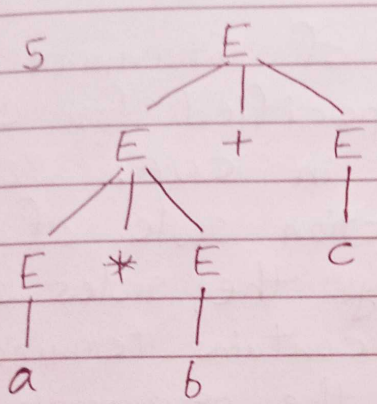
step 3



step 4



step 5



* Types of Derivation Tree

- Left most derivation tree
- Right most derivation tree
- Mixed derivation tree

Chomsky's Normal Form of CFG -

CNF stands for Chomsky normal form. A CFG (context-free grammar) is in CNF if all production rules satisfies one of the following conditions:

- Start symbol generating ε. eg: $A \rightarrow \varepsilon$.
- A non-terminal generating two non-terminals. eg: $S \rightarrow AB$.

Page No.

Date __/__/____

- A non-terminal generating a terminal.
eg: $S \rightarrow a$.

Example:

$G_1 = \{S \rightarrow AB, S \rightarrow c, A \rightarrow a, B \rightarrow b\}$
$G_2 = \{S \rightarrow aA, A \rightarrow a, B \rightarrow c\}$

The production rules of Grammar G1 satisfy the rules specified for CNF, so the grammar G1 is in CNF.
However, the production rule of Grammar G2 does not satisfy the rules specified for CNF as $S \rightarrow aZ$ contains terminal followed by non-terminal. So the grammar G2 is not in CNF.

* Steps for converting CFG into CNF -

Step 1 - Eliminate start symbol from the R.H.S. If the start symbol T is at the right hand side of ~~my~~ any production, create a new production as:
$S1 \rightarrow S$
where S1 is the new start symbol.

Step 2 - In the grammar, remove the null, unit and useless productions. You can refer to the simplification of CFG.

Step 3 - Eliminate terminals from the RHS

Page No.

Date __/__/____

of the production if they exist with other non-terminals or terminals. eg: production $S \rightarrow aA$ can be decomposed as:

$S \rightarrow RA$

$R \rightarrow a$

Step 4 - Eliminate RHS with more than two non-terminals. eg: $S \rightarrow ASB$ can be decomposed as:

$S \rightarrow RS$

$R \rightarrow AS$

Q.) Convert the given CFG to CNF. Consider the given grammar G1:

$S \rightarrow a \mid aA \mid B$

$A \rightarrow aBB \mid \varepsilon$

$B \rightarrow Aa \mid b$

⇒ Solution :-

Step 1 - We will create a new production $S1 \rightarrow S$, as the start symbol S appears on the RHS. The grammar will be:

$S1 \rightarrow S$

$S \rightarrow a \mid aA \mid B$

$A \rightarrow aBB \mid \varepsilon$

$B \rightarrow Aa \mid b$

Step 2 - As grammar G1 contains $A \rightarrow \varepsilon$ null production, its removal from the grammar yields:

Page No.

Date __/__/____

S1 → S
S → a | aA | B
A → aBB
B → Aa | b | a

Now as grammar G1 contains unit production S → B, its removal yield:

S1 → S
S → a | aA | Aa | b
A → aBB
B → Aa | b | a

Also remove the unit production S1 → S, its removal from the grammar yields:

S0 → a | aA | Aa | b
S → a | aA | Aa | b
A → aBB
B → Aa | b | a

Step 3 - In the production rule S0 → aA | Aa, S → aA | Aa, A → aBB and B → Aa, terminal a exists on RHS with non-terminals. So we will replace terminal a with X:

S0 → a | XA | AX | b
S → a | XA | AX | b
A → XBB
B → AX | b | a
X → a

Step 4 - In the production rule A → XBB, RHS has more than two symbols, removing it from grammar yield:

Date __/__/____

$S0 \rightarrow a \mid XA \mid AX \mid b$

$S \rightarrow a \mid XA \mid AX \mid b$

$A \rightarrow RB$

$B \rightarrow AX \mid b \mid a$

$X \rightarrow a$

$R \rightarrow XB$

Hence for the given grammar, this is the required CNF.

Date __/__/____

7. Pushdown Automata

A pushdown automaton is a way to implement a context-free grammar in a similar way we design DFA for a regular grammar.
A DFA can remember a finite amount of information, but a PDA can remember an infinite amount of information.

* Basically a pushdown automaton is - "Finite state machine" + "a stack"

* A pushdown automaton has three components -

- an input tape,
- a control unit, and
- a stack with infinite size

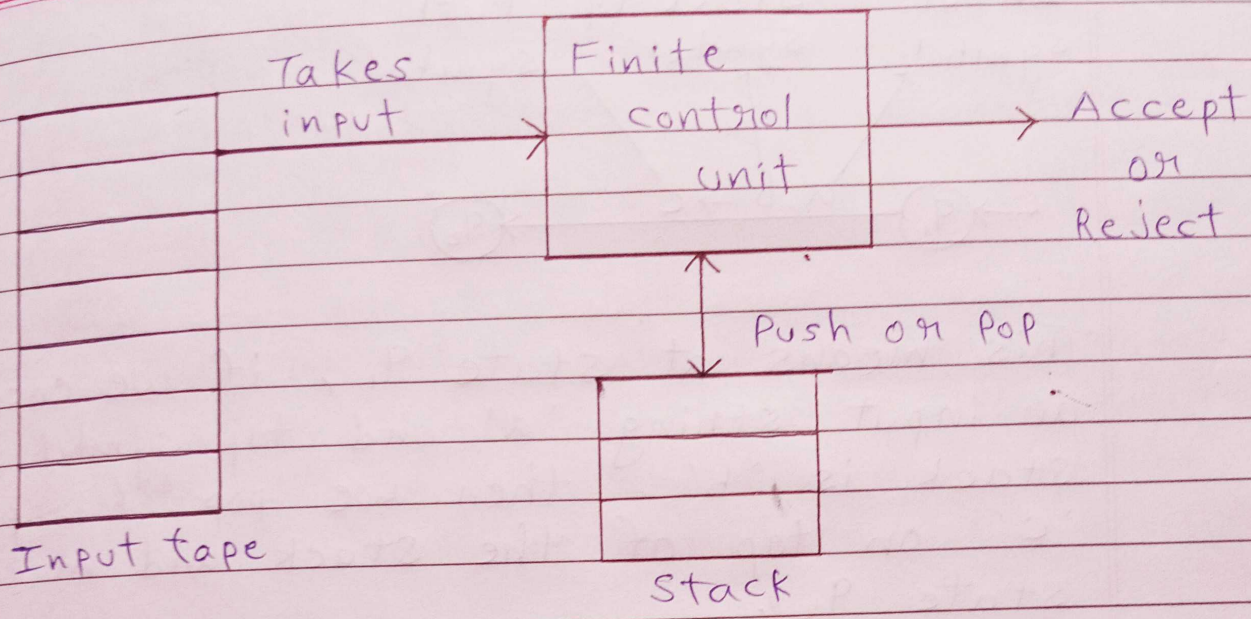
* The stack head scans the top symbol of the stack.

* A stack does two operations -

- Push - A new symbol added at the top.
- Pop - The top symbol is read and removed.

* A PDA may or may not read an input symbol, but it has to read the top of the stack in every transition.

Date __/__/____

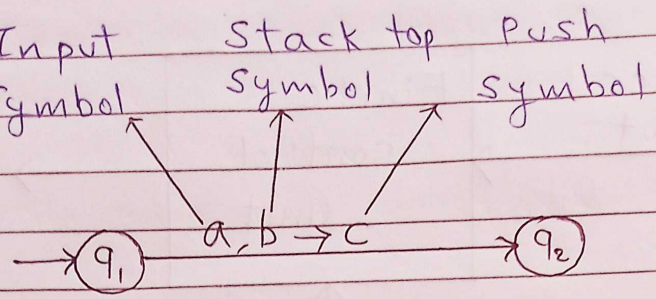


* A PDA can be formally described as a 7-tuple $(Q, \Sigma, S, \delta, q_0, I, F)$ –

- Q is the finite number of states
- Σ is input alphabet
- S is stack symbols
- δ is the transition function : $Q \times (\Sigma \cup \{\varepsilon\}) \times S \times Q \times S^*$
- q_0 is the initial state ($q_0 \in Q$)
- I is the initial stack top symbol ($I \in S$)
- F is a set of accepting states ($F \in Q$)

* The following diagram shows a transition in a PDA from a state q_1 to state q_2, labeled as ~~a, b~~ a, b→c –

Date __/__/____



This means at state q_1, if we encounter an input string 'a' and top symbol of the stack is 'b', then we pop 'b', push 'c' on top of the stack and move to state q_2.

* Terminologies Related to PDA -

→ Instantaneous Description - The instantaneous description (ID) of a PDA is represented by a triplet (q, w, s) where

- q is the state
- w is ~~uncom~~ unconsumed input
- s is the stack contents

→ Turnstile Notation - The 'turnstile' notation is used for connecting pairs of ID's that represent one or many moves of a PDA. The process of transition is denoted by the turnstile symbol "⊢".

Date __/__/____

* Consider a PDA $(Q, \Sigma, S, \delta, q_0, I, F)$. A transition can be mathematically represented by the following turnstile notation –

$(p, aw, T\beta) \vdash (q, w, \alpha b)$

This implies that while taking a transition from state p to state q, the input symbol 'a' is consumed, and top of the stack 'T' is replaced by a new string 'α'.

Note :- If we want zero or moves of a PDA, we have to use the symbol $(\vdash^*)$ for it.

Pushdown Automata Acceptance :-

There are two different ways to define PDA acceptability.

1) Final State Acceptability – In final state acceptability, a PDA accepts a string when, after reading the entire string, the PDA is in a final state. From the starting state, we can ~~move~~ make moves that end up in a final state with any stack values. The stack values are irrelevant as long as we end up in a final state.

For a PDA $(Q, \Sigma, S, \delta, q_0, I, F)$ the

Date __/__/____

language accepted by the set of final states F is -

$L(PDA) = \{w \mid (q_0, w, I) \vdash^* (q, \varepsilon, x), q \in F\}$

for any input stack string x.

2.) Empty Stack Acceptability - Here a PDA accepts a string when, after reading the entire string, the PDA has emptied its stack.

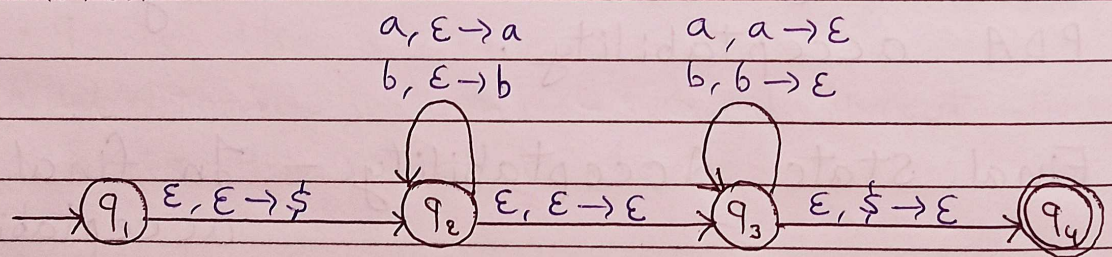
For a PDA $(Q, \Sigma, S, \delta, q_0, I, F)$ the language accepted by the empty stack is -

$L(PDA) = \{w \mid (q_0, w, I) \vdash^* (q, \varepsilon, \varepsilon), q \in Q\}$

Example: Construct a PDA that accepts

$L = \{WW^R \mid W = (a+b)^*\}$

Solution :-



Initially we put a special symbol '$' into the empty stack. At state q_2, the w is being read. In state q_3, each 0 or 1 is popped when it matches the input. If any other input is given, the PDA will go to a dead state. when we reach that special symbol '$', we go to the accepting state q_4.

Page No.

Date __/__/____

PDA & Context-free Grammar –

If a grammar G is context-free, we can build an equivalent nondeterministic PDA which accepts the language that is produced by the context-free grammar G. A parser can be built for the grammar G.

Also, if P is a pushdown automaton, an equivalent context-free grammar G can be constructed where
$L(G) = L(P)$

* Algorithm to find PDA corresponding to a given CFG–

Input – A CFG, $G = (V, T, P, S)$
output – Equivalent PDA, $P = (Q, \Sigma, S, \delta, q_0, I, F)$

Step 1 – Convert the productions of the CFG into GNF.

Step 2 – The PDA will have only one state {q}.

Step 3 – The start symbol of CFG will be the start symbol in the PDA.

Step 4 – All non-terminals of the CFG will be the stack symbols of the PDA and all the terminals of the CFG will be the input symbols of the PDA.

Date __/__/____

step 5 - For each production in the form $A \to aX$ where a is terminal and A, X are combination of terminal and non-terminals, make a transition $\delta(q, a, A)$.

Q.) construct a PDA from the following CFG. $G = (\{S, X\}, \{a, b\}, P, S)$ where the productions are -
$S \to XS \mid \varepsilon$, $A \to aXb \mid Ab \mid ab$

=) solution :-
Let the equivalent PDA,
$P = (\{q\}, \{a, b\}, \{a, b, X, S\}, \delta, q, S)$
where δ -
$\delta(q, \varepsilon, S) = \{(q, XS), (q, \varepsilon)\}$
$\delta(q, \varepsilon, X) = \{(q, aXb), (q, Xb), (q, ab)\}$
$\delta(q, a, a) = \{(q, \varepsilon)\}$
$\delta(q, 1, 1) = \{(q, \varepsilon)\}$

* Algorithm to find CFG corresponding to a given PDA -

Input - A CFG, $G = (V, T, P, S)$
Output - Equivalent PDA, $P = (Q, \Sigma, S, \delta, q_0, I, F)$ such that the non-terminals of the grammar G will be $\{X_{wx} \mid w, x \in Q\}$ and the start state will be $A_{q_0,F}$.

Step 1 - For Every $w, x, y, z \in Q$, $m \in S$ and $a, b \in \Sigma$, if $\delta(w, a, \varepsilon)$ contains (y, m) and (z, b, m) contains

Page No.

Date __/__/____

(x, ε), add the production rule $X_{wx} \rightarrow aX_{yz}b$ in grammar G.

Step 2 - For every $w, x, y, z \in Q$, add the production rule $X_{wx} \rightarrow X_{wy}X_{yx}$ in grammar G.

Step 3 - For $w \in Q$, add the production rule $X_{ww} \rightarrow \varepsilon$ in grammar G.

Parsing & PDA -

Parsing is used to derive a string using the production rules of a grammar. It is used to check the acceptability of a string. Compiler is used to check whether or not a string is syntactically and builds a parse tree.

* 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.

Date __/__/____

* Design of Top-Down Parser -

For Top-Down parsing, a PDA has the following four types of transitions -

- Pop the non-terminal on the left hand side of the production at the top of the stack and push its right-hand side string.
- If the top symbol of the stack matches with the input symbol being read, pop it.
- Push the start symbol 'S' into the stack.
- If the input string is fully read and the stack is empty, go to the final state 'F'.

Example:

Design a top-down parser for the expression "x+y*z" for the grammar G with the following production rules -

P: S → S+X | X, X → X*Y | Y, Y → (S) | id

=> Solution:

If the PDA is $(Q, \Sigma, S, \delta, q_0, I, F)$, then the top-down parsing is -

Page No.

Date __/__/____

$(X+Y*Z, I) \vdash (X+Y*Z, SI) \vdash (X+Y*Z, S+XI) \vdash (X+Y*Z, X+XI)$

$\vdash (X+Y*Z, Y+XI) \vdash (X+Y*Z, X+XI) \vdash (+Y*Z, +XI) \vdash (Y*Z, XI)$

$\vdash (Y*Z, X+YI) \vdash (Y*Z, Y*YI) \vdash (*Z, *YI) \vdash (Z, YI) \vdash (Z, ZI) \vdash (\varepsilon, I)$

* Design of a Bottom-Up Parser -

For bottom-up parsing, a PDA has the following four types of transitions -

- Push the current input symbol into the stack.
- Replace the right hand side of a production at the top of the stack with its left hand side.
- If the top of the stack element matches with the current input symbol, pop it.
- If the input string is fully read and only if the start symbol 'S' remains in the stack, pop it and go to the final state 'F'.

Date ___ / ___ / _____

Example:

Design a top-down parser for the expression "x+y*z" for the grammar G with the following production rules -

P: S → S+X | X, X → X*Y | Y, Y → (S) | id

=> solution:

If the PDA is (Q, Σ, S, δ, q_0, I, F), then the bottom-up parsing is -

(x + y*z, I) ⊢ (+y*z, xI) ⊢ (+y*z, YI)
⊢ (+y*z, xI) ⊢ (+y*z, SI)

⊢ (y*z, +SI) ⊢ (*z, y+SI) ⊢ (*z, Y+SI) ⊢
(*z, X+SI) ⊢ (z, *X+SI)

⊢ (ε, z*X+SI) ⊢ (ε, Y*X+SI) ⊢ (ε, X+SI)
⊢ (ε, SI)

Date __/__/____

8. Turing Machine and Linear bounded Automata

A Turing Machine is an accepting device which accepts the languages (recursively ~~enumer~~ enumerable set) generated by type 0 grammars. It was invented in 1936 by Alan Turing.

~~A Turing Machine is a mathemati~~

~~Turing~~ Machine Model –

A Turing Machine is a mathematical model which consists of an infinite length tape divided into cells on which input is given. The

A Turing Machine is a mathematical model which consists of an infinite length of tape divided into cells on which input is given. It consists of a head which reads the input tape. A state register stores the state of the TM. After reading an input symbol, it is replaced with another symbol, its internal state is changed, and it moves from one cell to the right or left. If the TM reaches the final state, the input string is accepted, otherwise rejected.

Date __/__/____

* A TM can be formally described as a 7-tuple $(Q, X, \Sigma, \delta, q_0, B, F)$ where -

- Q is a finite set of states
- X is the tape alphabet.
- Σ is the input alphabet.
- δ is a transition function; $\delta : Q \times X \rightarrow Q \times X \times \{Left\_shift, Right\_shift\}$.
- q_0 is the initial state.
- B is the blank symbol.
- F is the set of final states.

* Comparision with the previous automaton -

The following table shows a comparision of how a TM differs from FA and ~~PA~~ Pushdown Automata.

Machine	stack Data structure	Deterministic
Finite Automaton	N.A	Yes
Push down Automaton	Last In First out (LIFO)	No
Turing Machine	Infinite tape	Yes

Date __/__/____

* Example of Turing machine -

Turing Machine $M = (Q, X, \Sigma, \delta, q_0, B, F)$ with

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

$X = \{a, b\}$

$\Sigma = \{1\}$

$q_0 = \{q_0\}$

B = blank symbol

$F = \{q_f\}$

δ is given by -

Tape alphabet symbol	Present state 'q0'	Present state 'q1'	Present state 'q2'
a	$1 R q_1$	$1 L q_0$	$1 L q_f$
b	$1 L q_2$	$1 R q_1$	$1 R q_f$

Here the transition $1 R q_1$ implies that the write symbol is 1, the tape moves right and the next state is q_1. Similarly the transition $1 L q_2$ implies that the write symbol is 1, the tape moves left and the next state is q_2.

* Time and Space Complexity of a TM -

For a TM, the time complexity refers to the measure of the number of times the tape moves when the machine is initialized for some input symbols and the space complexity is the number

Page No.

Date __/__/____

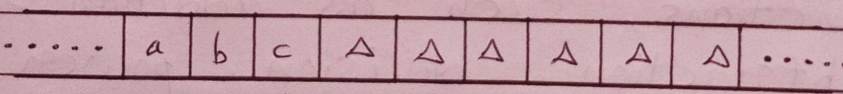
of cells of the tape written.

- Time complexity all reasonable functions - $T(n) = O(n \log n)$
- TM's space complexity - $S(n) = O(n)$

Turing Machine Model -

The TM can be modelled with the help of the following representation.

- The input tape is having an infinite number of cells, each cell containing one input symbol and thus the input string can be placed on tape. The empty tape is filled by blank characters.

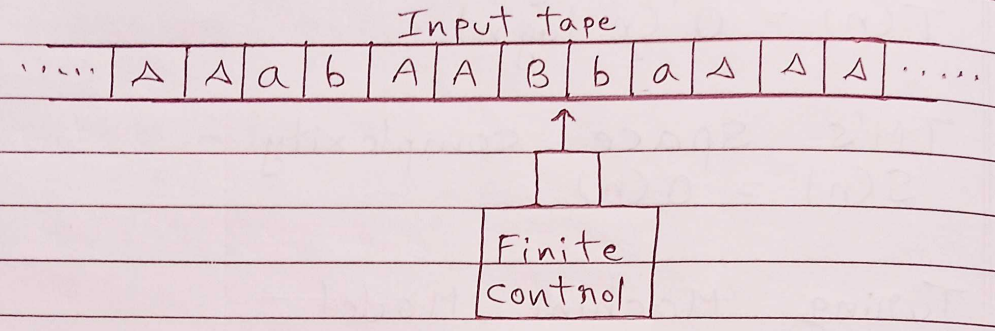


Input tape

- The finite control and the tape head which is responsible for reading the current input symbol. The tape head can move to left to right.
- A finite set of states through which machine has to undergo.
- Finite set of symbols called external

Date __/__/____

symbols which are used in building the logic of turing machine.



Language accepted by TM –

The TM accepts all the language even though they are recursively enumerable. Recursive means repeating the same set of rules for any number of times and enumerable means a list of elements.

The TM also accepts the computable functions, such as addition, multiplication, subtraction, division, power function, and many more.

Example: Construct a turing machine which accepts the language of aba over $\Sigma = \{a, b\}$

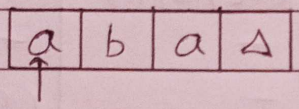
⇒ Solution: we will assume that one input tape the string 'aba' is placed like this:

a	b	a	Δ	.....

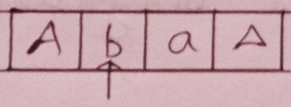
Date __/__/____

The tape head will read out the sequence up to the Δ characters. If the tape head is readout 'aba' string then TM will half after reading Δ.

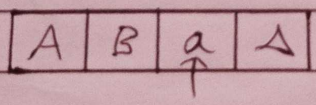
Now, we will see how this turing machine will work for aba. Initially, state is q_0 and head points to 'a' as:



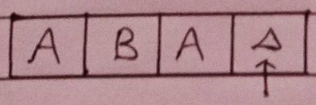
The move will be $\delta(q_0, a) = \delta(q_1, A, R)$ which means it will go to state q_1, replaced 'a' by 'A' and head will move to right as:



The move will be $\delta(q_1, b) = \delta(q_2, B, R)$ which means it will go to state q_2, replaced 'b' by 'B' and head will move to right as:



The move will be $\delta(q_2, a) = \delta(q_3, A, R)$ which means it will go to state q_3, replaced 'a' by 'A' and head will move to right as:



The move $\delta(q_3, \Delta) = (q_4, \Delta, S)$ which means it will go to state q_4 which is the HALT state and HALT state is always an

Page No.

Date __/__/____

accept state for any TM.

* The same TM can be represented by Transition Table :

States	a	b	Δ
q_0	(q_1, A, R)	–	–
q_1	–	(q_2, B, R)	–
q_2	(q_3, A, R)	–	–
q_3	–	–	(q_4, Δ, S)
q_4	–	–	–

* The same TM can be represented by Transition Diagram :