USN | | | | | | | | | |

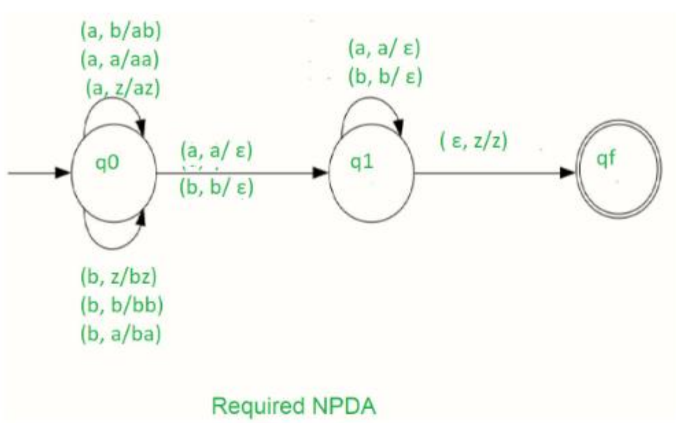
Internal Assessment Test 2–December2024

Sub:	**Theory of Computation**				Sub Code:	BCS503	Branch:	AIML/CSE AIML
Date:	**15/12/2024**	Duration:	90min's	Max Marks: 50	Sem/Sec:	V/A,B CSEAIML(A)		OBE
Answer any FIVE FULL Questions						MARKS	CO	RBT
1.(a)	Construct PDA that aacepts the language L={ ww^R | w∈{a+b}* and w^R is the reversal of w}. Write ID for the string aabbaa.					6	CO3	L2, L3

Where, q0 = Initial state

qf = Final state

ϵ = indicates pop operation



Let's create the ID for the string aabbaaaabbaaaabbaa.

1. **Initial ID (Before any steps):**
(q0,aabbaa,Z0)(q_0, aabbaa, Z_0)(q0,aabbaa,Z0)
   - State: q0q_0q0
   - Input: aabbaaaabbaaaabbaa
   - Stack: [Z0][Z_0][Z0]
2. **After reading the first aaa:**
(q0,abbaa,[a,Z0])(q_0, abbaa, [a, Z_0])(q0,abbaa,[a,Z0])
   - State: q0q_0q0
   - Input: abbaaabbaaabbaa
   - Stack: [a,Z0][a, Z_0][a,Z0]
3. **After reading the second aaa:**
(q0,bbaa,[a,a,Z0])(q_0, bbaa, [a, a, Z_0])(q0,bbaa,[a,a,Z0])
   - State: q0q_0q0
   - Input: bbaabbaabbaa
   - Stack: [a,a,Z0][a, a, Z_0][a,a,Z0]
4. **After reading the first bbb:**
(q0,baa,[a,a,b,Z0])(q_0, baa, [a, a, b, Z_0])(q0,baa,[a,a,b,Z0])
   - State: q0q_0q0
   - Input: baabaabaa
   - Stack: [a,a,b,Z0][a, a, b, Z_0][a,a,b,Z0]
5. **After reading the second bbb:**
(q0,aa,[a,a,b,b,Z0])(q_0, aa, [a, a, b, b, Z_0])(q0,aa,[a,a,b,b,Z0])
   - State: q0q_0q0
   - Input: aaaaaa
   - Stack: [a,a,b,b,Z0][a, a, b, b, Z_0][a,a,b,b,Z0]

	6. **Transition to q1q_1q1 (start reading second half):** (q1,aa,[a,a,b,b,Z0])(q_1, aa, [a, a, b, b, Z_0])(q1,aa,[a,a,b,b,Z0]) o State: q1q_1q1 o Input: aaaaaa o Stack: [a,a,b,b,Z0][a, a, b, b, Z_0][a,a,b,b,Z0] 7. **After reading the first aaa of second half:** (q1,a,[a,b,b,Z0])(q_1, a, [a, b, b, Z_0])(q1,a,[a,b,b,Z0]) o State: q1q_1q1 o Input: aaa o Stack: [a,b,b,Z0][a, b, b, Z_0][a,b,b,Z0] 8. **After reading the second aaa:** (q1,ϵ,[b,b,Z0])(q_1, \epsilon, [b, b, Z_0])(q1,ϵ,[b,b,Z0]) o State: q1q_1q1 o Input: ϵ\epsilonϵ o Stack: [b,b,Z0][b, b, Z_0][b,b,Z0] 9. **After reading the first bbb:** (q1,ϵ,[b,Z0])(q_1, \epsilon, [b, Z_0])(q1,ϵ,[b,Z0]) o State: q1q_1q1 o Input: ϵ\epsilonϵ o Stack: [b,Z0][b, Z_0][b,Z0] 10. **After reading the second bbb:** (qf,ϵ,[Z0])(q_f, \epsilon, [Z_0])(qf,ϵ,[Z0]) o State: qfq_fqf o Input: ϵ\epsilonϵ o Stack: [Z0][Z_0][Z0] Thus, the final ID shows that the PDA has accepted the string.			
1.(b)	Convert the following CFG to PDA and give the procedure for the same. S→aABB|aAA A→aBB|a B→bBB|A C→a □ **Initialization**: Start with the initial stack symbol Z0Z_0Z0. δ(q0,ϵ,Z0)=(q0,S)\delta(q_0, \epsilon, Z_0) = (q_0, S)δ(q0,ϵ,Z0)=(q0,S) □ **For S→aABBS \to aABBS→aABB**: δ(q0,ϵ,S)=(q0,aABB)\delta(q_0, \epsilon, S) = (q_0, aABB)δ(q0,ϵ,S)=(q0,aABB) □ **For S→aAAS \to aAAS→aAA**: δ(q0,ϵ,S)=(q0,aAA)\delta(q_0, \epsilon, S) = (q_0, aAA)δ(q0,ϵ,S)=(q0,aAA) □ **For A→aBBA \to aBBA→aBB**: δ(q0,ϵ,A)=(q0,aBB)\delta(q_0, \epsilon, A) = (q_0, aBB)δ(q0,ϵ,A)=(q0,aBB) □ **For A→aA \to aA→a**: δ(q0,ϵ,A)=(q0,a)\delta(q_0, \epsilon, A) = (q_0, a)δ(q0,ϵ,A)=(q0,a) □ **For B→bBBB \to bBBB→bBB**: δ(q0,ϵ,B)=(q0,bBB)\delta(q_0, \epsilon, B) = (q_0, bBB)δ(q0,ϵ,B)=(q0,bBB) □ **For B→AB \to AB→A**: δ(q0,ϵ,B)=(q0,A)\delta(q_0, \epsilon, B) = (q_0, A)δ(q0,ϵ,B)=(q0,A) □ **For C→aC \to aC→a**: δ(q0,ϵ,C)=(q0,a)\delta(q_0, \epsilon, C) = (q_0, a)δ(q0,ϵ,C)=(q0,a)	4	CO3	L3
2	Define CFG and CNF. Convert the given CFG to CNF S→ABC|BaB A→aA|BaC|aaa B→bBb|a|D C→CA|AC D→ϵ	10	CO3	L1, L3

Context-Free Grammar (CFG)

A **Context-Free Grammar (CFG)** is a formal grammar where every production rule is of the form:

A→γA \to \gammaA→γ

where AAA is a non-terminal symbol and γ\gammaγ is a string of terminals and/or non-terminals. In other words, the left-hand side of every production consists of a single non-terminal, and the right-hand side can be a string of terminals, non-terminals, or an empty string (denoted ϵ\epsilonϵ).

A CFG is used to define the syntax of programming languages, natural languages, and other formal languages.

Chomsky Normal Form (CNF)

A **Context-Free Grammar (CFG)** is in **Chomsky Normal Form (CNF)** if all of its production rules satisfy one of the following conditions:

1. **A → BC**, where AAA, BBB, and CCC are non-terminal symbols, and BBB and CCC are not the start symbol.
2. **A → a**, where AAA is a non-terminal symbol, and aaa is a terminal symbol.
3. **A → \epsilon**, where AAA is the start symbol and the production is allowed only if the language generated by the grammar includes the empty string.

In CNF, all productions must either have two non-terminals on the right-hand side or a single terminal symbol.

Converting CFG to CNF

To convert a CFG to CNF, the following steps are typically followed:

1. **Eliminate ε-productions** (productions of the form A→ϵA \to \epsilonA→ϵ, except for the start symbol if it can derive the empty string).
2. **Eliminate unit productions** (productions of the form A→BA \to BA→B, where both AAA and BBB are non-terminals).
3. **Eliminate useless symbols** (symbols that do not derive any terminal string).
4. **Convert all productions into binary form** (i.e., ensure that every production is either of the form A→BCA \to BCA→BC or A→aA \to aA→a, where aaa is a terminal).

3(a) State and prove pumping Lemma for context free languages. | 5 | CO3 | L1, L2

The **Pumping Lemma for Context-Free Languages** is a property that all context-free languages must satisfy. It is used primarily to prove that certain languages are **not** context-free by showing they do not meet this property.

Statement of the Pumping Lemma for CFLs:

If a language LLL is context-free, then there exists a pumping length ppp such that any string w∈Lw \in Lw∈L with |w|≥p|w| \geq p|w|≥p can be decomposed into five parts:

w=uvxyzw = uvxyzw=uvxyz

such that:

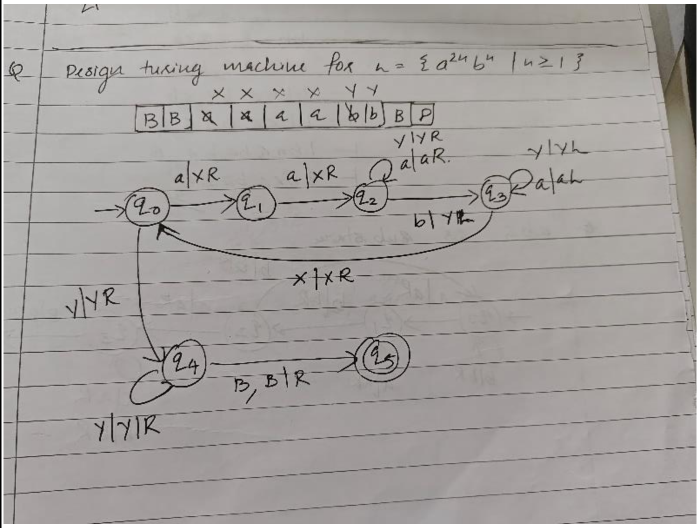
1. |vxy|≤p|vxy| \leq p|vxy|≤p (The middle portion is of limited length.)
2. |vy|≥1|vy| \geq 1|vy|≥1 (The repeated parts are not empty.)
3. For all k≥0k \geq 0k≥0, the string u(vk)x(yk)z∈Lu(v^k)x(y^k)z \in Lu(vk)x(yk)z∈L. (The string remains in the language when the repeating parts are pumped.)

Proof of the Pumping Lemma for CFLs:

1. Consider a context-free language LLL and its corresponding context-free

<table>
<tr><td></td><td>
grammar GGG.
2. Suppose GGG has nnn non-terminal symbols.
3. Let p=2np = 2^np=2n. This is the pumping length.
4. Consider any string w∈Lw \in Lw∈L with |w|≥p|w| \geq p|w|≥p. The parse tree for www has a height of at least n+1n+1n+1 due to the pigeonhole principle, since there are more symbols in the string than non-terminals in the grammar.
5. By the pigeonhole principle, some non-terminal symbol must appear more than once along a path from the root to a leaf in the parse tree.
6. Let this repeated non-terminal be AAA. Consider the substring generated by AAA on the first and second occurrences.
7. Decompose w=uvxyzw = uvxyzw=uvxyz, where:
o vvv and yyy are the substrings derived from repeated occurrences of AAA.
o xxx is the part between these substrings.
8. Since the grammar is context-free, replacing the repeated occurrences of AAA by producing more or fewer copies still generates valid strings in LLL.
9. Thus, for any integer kkk, the string u(vk)x(yk)z∈Lu(v^k)x(y^k)z \in Lu(vk)x(yk)z∈L.
This proves the Pumping Lemma for CFLs.
</td><td></td><td></td><td></td></tr>
<tr><td>3.(b)</td><td>
Prove that language $L=\{a^nb^nc^n \mid n>=1\}$ is not context free.

1. Assume LLL is context-free.
2. Let the pumping length be ppp.
3. Consider the string w=apbpcpw = a^p b^p c^pw=apbpcp.
4. Decompose w=uvxyzw = uvxyzw=uvxyz, where |vxy|≤p|vxy| \leq p|vxy|≤p and |vy|≥1|vy| \geq 1|vy|≥1.
5. The substring vxyvxyvxy can contain symbols from at most two of the three blocks apa^pap, bpb^pbp, and cpc^pcp, because its length is at most ppp.
6. Pumping vvv and yyy will unbalance the counts of at least two symbols, causing the string to fall out of the language.
This contradicts the Pumping Lemma, so LLL is not context-free
</td><td>5</td><td>CO3</td><td>L3</td></tr>
<tr><td>4</td><td>Construct Turing Machine to accept the language $L=\{a^{2n}b^n \mid n>=1\}$.Give the transition table as well as Transition diagram of TM obtained.</td><td>10</td><td>CO4</td><td>L2,L3</td></tr>
</table>



5	Write a short notes on: i) Turing Machine and its working	10	CO4	L2

Turing Machine and Its Working

A **Turing Machine (TM)** is a theoretical computational model that defines an abstract machine capable of simulating any computer algorithm. It was introduced by **Alan Turing** in 1936 and serves as a foundation for the theory of computation.

Components of a Turing Machine:

1. **Tape:** An infinite strip of cells, each capable of holding a symbol from a finite alphabet.
2. **Head:** A read/write head that can move left or right along the tape.
3. **States:** A finite set of states, including a start state and one or more accepting or rejecting states.
4. **Alphabet:** A set of symbols, including a special blank symbol (usually denoted by '␣').
5. **Transition Function:** A set of rules that define how the machine transitions between states based on the current symbol and state.

Working of a Turing Machine:

1. **Initialization:** The machine starts in the initial state with the tape containing the input string, and the head positioned at the first symbol.
2. **Reading and Writing:**
   - The head reads the current symbol on the tape.
   - Based on the symbol and the current state, the machine consults its transition function.
3. **Transition:**
   - The machine may:
     - **Write** a new symbol on the current tape cell.
     - **Move** the head left or right by one cell.
     - **Change** to a new state.
4. **Halting:**
   - The machine halts when it reaches an accepting or rejecting state or if no transition is defined for the current configuration.

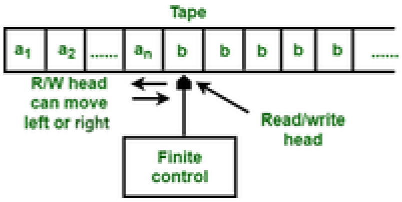


Figure: Turing Machine

ii) Multi tape Turing Machine

A **Multi-Tape Turing Machine** is an extension of the standard Turing Machine that uses multiple tapes, each with its own read/write head. It is used to improve computational efficiency and simplify the design of Turing Machines for complex problems.

Components of a Multi-Tape Turing Machine:

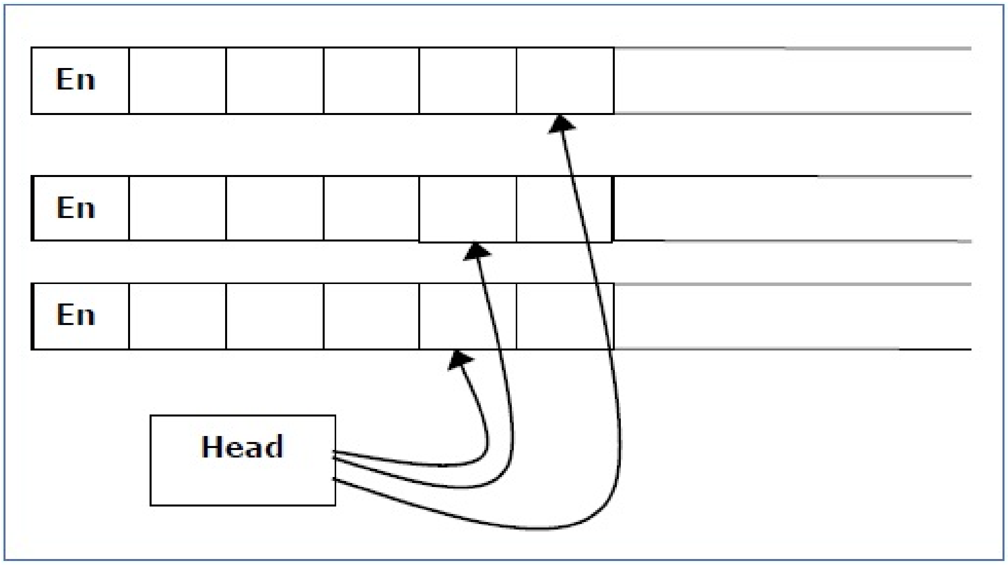
1. **Tapes:** There are multiple infinite tapes, each capable of storing symbols.
2. **Heads:** Each tape has its own read/write head.
3. **States:** A finite set of states, including start, accept, and reject states.
4. **Alphabet:** A set of symbols, including a blank symbol.
5. **Transition Function:** The machine reads symbols from all tapes simultaneously and defines transitions based on these symbols and the current state.

Working of a Multi-Tape Turing Machine:

1. **Initialization:** The first tape holds the input string, while other tapes are blank.
2. **Reading and Writing:**
   - The heads read symbols from all tapes simultaneously.
   - Based on the current state and symbols read, the machine:
     - Writes symbols on any or all tapes.
     - Moves each head independently (left, right, or stationary).
     - Changes its current state.
3. **Halting:** The machine halts when it reaches an accepting or rejecting state.

Advantages:

- **Faster Computation:** Simulates complex computations more efficiently.
- **Simplified Design:** Easier to design machines for tasks like copying, sorting, and parsing.
- **Theoretical Power:** Equivalent in computational power to a single-tape Turing Machine but more time-efficient.

<table>
<tr><td></td><td>
</td><td></td><td></td><td></td></tr>
<tr><td>6</td><td>Write a short notes on:
i) Recursive Language

A Recursive Language is a type of formal language in the Chomsky hierarchy that is decidable by a Turing Machine. This means there exists a Turing Machine that can always halt and determine whether a given input string belongs to the language.<hr>Definition:
A language L⊆Σ∗L \subseteq \Sigma^*L⊆Σ∗ is recursive if there exists a Turing Machine MMM such that:
1. MMM halts and accepts if w∈Lw \in Lw∈L.
2. MMM halts and rejects if w∉Lw \notin Lw∈/L.<hr>Key Characteristics:
1. Decidability: Recursive languages are also called decidable languages because their membership problem can be solved by a Turing Machine that always halts.
2. Closure Properties: Recursive languages are closed under:
o Union
o Intersection
o Complementation
o Concatenation
o Kleene star

ii) Universal Language

Universal Language
The Universal Language (L_U) refers to a formal language associated with the encoding of Turing Machines and their inputs. It represents the set of strings that encode a Turing Machine and an input string, where the Turing Machine accepts the input.<hr>Definition:
The Universal Language LUL_ULU is defined as:
LU={⟨M,w⟩∣M is a Turing Machine and M accepts w}L_U = \{ \langle M, w \rangle \mid M \text{ is a Turing Machine and } M \text{ accepts } w \}LU ={⟨M,w⟩∣M is a Turing Machine and M accepts w}</td><td>10</td><td>CO5</td><td>L2</td></tr>
</table>

Here, ⟨M,w⟩\langle M, w \rangle⟨M,w⟩ is an encoding of the Turing Machine MMM and the input string www.

Key Properties:

1. **Recursively Enumerable:**
    - The Universal Language is **recursively enumerable (RE)** but **not recursive.**
    - A Turing Machine can simulate MMM on www; if MMM accepts www, the universal Turing Machine also accepts the encoded pair.
2. **Non-Decidability:**
    - The Universal Language is **not decidable** because determining whether an arbitrary Turing Machine accepts an input is equivalent to solving the **Halting Problem**, which is undecidable.
3. **Closure Properties:**
    - It is closed under union and intersection but **not** under complementation.

Faculty Signature CCI Signature HOD Signature