USN

Internal Assessment Test II –December 2025

Sub:	**Theory Of Computation**			Sub Code:	BCS503	Branch:	ISE
Date:	**01/12/2025**	Duration:	90 min’s	Max Marks: 50	Sem/Sec:	V A, B & C	OBE

	<u>Answer any FIVE FULL Questions (Give Definition wherever is necessary)</u>	MARKS	CO	RBT
1.	Design a PDA for the language that accepts the string with na(w)<nb(w), where wϵ(a+b)* and show the instantaneous description of the PDA on input abbab	10	CO3	L3

Scheme

Language explanation & idea (2 Marks)
PDA construction (2 Marks)
Transition (3 Marks)
IDs Constriction (3 Marks)

Solution

The PDA keeps track of the difference between the number of a’s and b’s. If b’s exceed a’s, the PDA must accept. The stack stores unmatched symbols.

Stack symbols: A (unmatched 'a'), B (unmatched 'b'), Z_0 (bottom marker)

Transitions

For symbol 'a':
$\delta(q, a, B) \rightarrow (q, \varepsilon)$ (pop B)
$\delta(q, a, A) \rightarrow (q, AA)$ (push A)
$\delta(q, a, Z_0) \rightarrow (q, AZ_0)$ (push A)

For symbol 'b':
$\delta(q, b, A) \rightarrow (q, \varepsilon)$ (pop A)
$\delta(q, b, B) \rightarrow (q, BB)$ (push B)
$\delta(q, b, Z_0) \rightarrow (q, BZ_0)$ (push B)

Acceptance:
$\delta(q, \varepsilon, B) \rightarrow (q\_acc, B)$ (ε-move to accept when at least one B exists)

2. Instantaneous Descriptions (IDs) for abbab

$(q, abbab, Z_0)$
$(q, bbab, AZ_0)$ — read a, push A
(q, bab, Z_0) — read b, pop A
(q, ab, BZ_0) — read b, push B
(q, b, Z_0) — read a, pop B
(q, ε, BZ_0) — read b, push B
$(q\_acc, \varepsilon, BZ_0)$ — ε-move, ACCEPT

	3. Conclusion String abbab has $n_a = 2$, n_b = 3. Since n_a < n_b, the PDA accepts the string, matching final configuration.			
2.	Illustrate whether the grammar is Ambiguous or not? $S \rightarrow aB \mid bA$ $A \rightarrow a \mid aS \mid bAA$ $B \rightarrow b \mid bS \mid aBB$ For the string "aaabbabbba" . Draw LMD or RMD **Scheme** Statement of grammar and target string (1 Marks) First correct leftmost derivation (LMD-1) (3 Marks) Second distinct correct leftmost derivation (LMD-2) (3 Marks) Parse trees / explanation of difference and conclusion. (3 Marks) Solution String positions for clarity: a a a b b a b b b a (length 10). **Leftmost derivation 1 (LMD-1)** 1. $S \Rightarrow aB$ 2. $\Rightarrow a\ aBB$(apply $B \rightarrow aBB$to leftmost B) 3. $\Rightarrow a\ a\ aBBB$(apply $B \rightarrow aBB$to leftmost B) 4. $\Rightarrow a\ a\ a\ bBB$(apply leftmost $B \rightarrow b$) 5. $\Rightarrow a\ a\ a\ b\ bB$(apply next $B \rightarrow b$) 6. $\Rightarrow a\ a\ a\ b\ b\ aBB$(apply remaining $B \rightarrow aBB$) 7. $\Rightarrow a\ a\ a\ b\ b\ a\ bB$(apply leftmost of these $B \rightarrow b$) 8. $\Rightarrow a\ a\ a\ b\ b\ a\ b\ bS$(apply last $B \rightarrow bS$) 9. $\Rightarrow a\ a\ a\ b\ b\ a\ b\ b\ bA$(apply $S \rightarrow bA$) 10. $\Rightarrow a\ a\ a\ b\ b\ a\ b\ b\ b\ a$(apply $A \rightarrow a$) Hence LMD-1 yields aaabbabbba.	10	CO3	L3
	Leftmost derivation 2 (LMD-2) — different choice earlier 1. $S \Rightarrow aB$ 2. $\Rightarrow a\ aBB$(apply $B \rightarrow aBB$) 3. $\Rightarrow a\ a\ aBBB$(apply $B \rightarrow aBB$) 4. $\Rightarrow a\ a\ a\ bSBB$(apply leftmost $B \rightarrow bS$**instead of** $B \rightarrow b$) 5. $\Rightarrow a\ a\ a\ b\ bABB$(apply $S \rightarrow bA$) 6. $\Rightarrow a\ a\ a\ b\ b\ aBB$(apply $A \rightarrow a$) 7. $\Rightarrow a\ a\ a\ b\ b\ a\ bB$(apply leftmost $B \rightarrow b$) 8. $\Rightarrow a\ a\ a\ b\ b\ a\ b\ bS$(apply last $B \rightarrow bS$) 9. $\Rightarrow a\ a\ a\ b\ b\ a\ b\ b\ bA$(apply $S \rightarrow bA$) 10. $\Rightarrow a\ a\ a\ b\ b\ a\ b\ b\ b\ a$(apply $A \rightarrow a$) LMD-2 also yields aaabbabbba. The **order and grouping of expansions** differ from LMD-1 (compare step 4–6 in each derivation), so the parse trees are different.			

	We prove this by producing **two different leftmost derivations** (LMDs) that both yield aaabbabbba but have different production structure → two distinct parse trees.			
3.	Simplify the given grammar remove all the useless symbols, ϵ-productions and Unit productions from the grammar: S→ABCa/Bd ,A→BC/b , B→b/ ε ,C→c/ ε , D→d **Scheme** Remove useless/unreachable symbols: identify and remove D (3 Mark), Eliminate ε-productions correctly (3 Marks) Eliminate unit productions (3 Marks) Final Grammer (1 Marks) **Solution** (Terminals: a,b,c,d — nonterminals: S,A,B,C,D) **Step 1 — remove useless symbols (unreachable / non-generating)** 1. Check reachability from start S: S uses A,B,C and B and d (terminal). So A,B,C are reachable. D is **not used** anywhere (only D → d), so D is **unreachable** → remove D. 2. Check generating (can produce terminals): o B generates b (or ε) → generating. o C generates c (or ε) → generating. o A can produce BC and b → since B and C generate, A is generating. o S produces strings containing terminals → generating. All remaining nonterminals are generating. So after removing unreachable D, grammar stays: S → ABCa | Bd A → BC | b B → b | ε C → c | ε **Step 2 — eliminate ε-productions** Find nullable nonterminals (those that can derive ε): • B → ε so B nullable. • C → ε so C nullable. • A → BC and both B,C nullable ⇒ A **nullable** as well. (S is not nullable because both S-productions end with terminal a or d.) Eliminate ε-productions by adding alternatives that omit nullable symbols (but *do not* add S → ε because S is not nullable). **Expand S → ABCa** Generate all variants by optionally omitting each of A, B, C (since each nullable) — keep the terminal a: • ABCa (original) • ABa (omit C) • ACa (omit B)	10	CO4	L3

	• BCa (omit A) • Aa (omit B,C) • Ba (omit A,C) • Ca (omit A,B) • a (omit A,B,C) **Expand $S \rightarrow Bd$** B nullable ⇒ also $S \rightarrow d$. So S alternatives become: $S \rightarrow ABCa \mid ABa \mid ACa \mid BCa \mid Aa \mid Ba \mid Ca \mid a \mid Bd \mid d$ **Expand $A \rightarrow BC$** B and C nullable ⇒ add $A \rightarrow B \mid C \mid \varepsilon$. (Also $A \rightarrow b$ already exists.) So temporarily (before deleting ε rules) A alternatives: $A \rightarrow BC \mid B \mid C \mid \varepsilon \mid b$ B and C already $b \mid \varepsilon$, $c \mid \varepsilon$. Now **remove all ε-productions** $B \rightarrow \varepsilon$, $C \rightarrow \varepsilon$, $A \rightarrow \varepsilon$ (we have already accounted for their effect by adding the alternative forms above). After removing the explicit ε rules, the grammar becomes: $S \rightarrow ABCa \mid ABa \mid ACa \mid BCa \mid Aa \mid Ba \mid Ca \mid a \mid Bd \mid d$ $A \rightarrow BC \mid B \mid C \mid b$ $B \rightarrow b$ $C \rightarrow c$ (At this stage there are **no ε-productions**.) **Step 3 — eliminate unit productions** A *unit production* has the form $X \rightarrow Y$ where Y is a single nonterminal. In our grammar the unit productions are: $A \rightarrow B$ $A \rightarrow C$ (There are no other single-nonterminal right-hand sides.) Compute what B and C produce (except unit productions): • $B \rightarrow b$ so replace $A \rightarrow B$ by $A \rightarrow b$ (but $A \rightarrow b$ is already present). • $C \rightarrow c$ so replace $A \rightarrow C$ by $A \rightarrow c$. After adding those, remove the unit productions $A \rightarrow B$ and $A \rightarrow C$. The grammar becomes: $S \rightarrow ABCa \mid ABa \mid ACa \mid BCa \mid Aa \mid Ba \mid Ca \mid a \mid Bd \mid d$ $A \rightarrow BC \mid b \mid c$ $B \rightarrow b$ $C \rightarrow c$ Now there are **no unit productions**.			
4	Define CNF. Convert the following grammar into CNF. S --> 0A | 1B A --> 0AA | 1S | 1 B --> 1BB | 0S | 0 **Scheme**	10	CO4	L2

CNF Definition (2 Marks)
Write down the original grammar and observe there are no ε-productions or unit productions initially (brief statement) (1 Marks)
Introduce terminal nonterminal X0 → 0, X1 → 1 and correctly show replacement in all mixed/right-hand sides (2 Marks)
Binarize long RHS: introduce Y1 → A A, Y2 → B B and show the replaced productions A → X0 Y1 and B → X1 Y2 (3 Marks)
Present the final CNF grammar (all productions listed), and state there are no ε or unit productions remaining (2 Marks)

Solution
Chomsky Normal Form (CNF): A context-free grammar is in CNF if every production is of one of these two forms:

- $A \rightarrow BC$where $B, C$are non-start nonterminals, or
- $A \rightarrow a$where $a$is a terminal.

Step 1 — Introduce nonterminal for terminals that appear in mixed/right-hand sides
CNF allows terminals only in productions of the form $A \rightarrow a$. For right-hand sides longer than one symbol we must not have raw terminals. Introduce:
X0 → 0
X1 → 1
Now replace terminal symbols in mixed productions by these new nonterminal.
Replace:

- S → 0A | 1B ⇒ S → X0 A | X1 B
- A → 0AA | 1S | 1 ⇒ A → X0 A A | X1 S | 1
- B → 1BB | 0S | 0 ⇒ B → X1 B B | X0 S | 0

(We keep the original single-terminal productions A → 1 and B → 0 as they are allowed in CNF.)

Step 2 — Binarize RHS with length > 2
CNF requires RHS length ≤ 2 (and if 2, both must be nonterminal). The only long productions now are A → X0 A A and B → X1 B B (length 3). We introduce new helper nonterminal to make them binary.
Introduce:
Y1 → A A
Y2 → B B
Replace:

- A → X0 A A ⇒ A → X0 Y1
- B → X1 B B ⇒ B → X1 Y2

Now all productions are either length 2 (two nonterminal) or length 1 (a terminal).

Step 3 — List the final CNF grammar
All productions now conform to CNF (either two nonterminal or one terminal).
Final CNF grammar
Nonterminal: S, A, B, X0, X1, Y1, Y2
Terminals: 0, 1

	Productions: S → X0 A S → X1 B A → X0 Y1 A → X1 S A → 1 B → X1 Y2 B → X0 S B → 0 Y1 → A A Y2 → B B X0 → 0 X1 → 1 Check: • Every production is either $N \to NN$or $N \to t$. • No ε-productions, no unit productions remain. • Start symbol S appears on RHS in some rules — that is allowed; we did **not** introduce a new start symbol because there were no ε-productions requiring it.			
5	Construct the TM to Accept the language $a^n b^n c^n$ n>=1, show the sequence of moves for the string “aabbcc”, Give Transition diagram and Transition function **Scheme** Definition / Idea of TM (1 Marks) States & Alphabets (2 Marks) Transition Function & Diagram (7 Marks) **Solution** The TM works by repeatedly matching one a, one b, and one c and marking them so they won’t be reused: 1. Scan from the left to find the leftmost unmarked a. Replace it by a marker X. 2. Move right to find the leftmost unmarked b. Replace it by Y. If no b found (but expected), reject. 3. Move right to find the leftmost unmarked c. Replace it by Z. If no c found, reject. 4. Return to the left end of the tape and repeat (go back to step 1). 5. If in step 1 there is no unmarked a left (only X,Y,Z and blanks remain), go to a verification state and check that there are no unmarked b or c. If verified, accept; otherwise reject. Markers: X marks an a, Y marks a b, Z marks a c. Blank symbol is □ **Alphabet & states**	10	CO5	L3

	• Input alphabet: $\{a, b, c\}$
	• Tape alphabet: $\{a, b, c, X, Y, Z, \square\}$
	• States: $\{q_0, q_1, q_2, q_3, q_5, q_{accept}, q_{reject}\}$
	  ○ q_0: find leftmost unmarked a (or decide none remain)
	  ○ q_1: after marking a (X) — find an unmarked b and mark Y
	  ○ q_2: after marking b (Y) — find an unmarked c and mark Z
	  ○ q_3: return to left end (go to beginning)
	  ○ q_5: final verification (ensure no unmarked a/b/c remain)
	  ○ q_{accept}, q_{reject}: halting states
	Head initially on the leftmost input symbol, start state q_0.
	Transition function (compact table notation)
	We write δ(q, symbol) = (q', write, move) with moves R or L. Any missing entry goes to q_{reject}(or can be considered undefined).
	State q_0— search for leftmost unmarked a
	δ(q0, a) = (q1, X, R) // found unmarked a: mark it and search b
	δ(q0, X) = (q0, X, R) // skip marked a
	δ(q0, Y) = (q0, Y, R) // skip marked b
	δ(q0, Z) = (q0, Z, R) // skip marked c
	δ(q0, b) = (q5, b, R) // no more a's (b encountered) -> verify remaining
	δ(q0, c) = (q5, c, R) // no more a's (c encountered) -> verify
	δ(q0, □) = (q5, □, R) // reached blank -> verify
	State q_1— search right for first unmarked b
	δ(q1, a) = (q1, a, R) // skip any a's (they might be present before b's)
	δ(q1, X) = (q1, X, R) // skip marked a
	δ(q1, Y) = (q1, Y, R) // skip marked b
	δ(q1, Z) = (q1, Z, R) // skip marked c
	δ(q1, b) = (q2, Y, R) // found an unmarked b -> mark it and search c
	δ(q1, c) = (q_{reject}, c, R) // c before b -> mismatch -> reject
	δ(q1, □) = (q_{reject}, □, R) // end without finding b -> reject
	State q_2— search right for first unmarked c
	δ(q2, a) = (q2, a, R) // skip
	δ(q2, X) = (q2, X, R)
	δ(q2, Y) = (q2, Y, R)
	δ(q2, Z) = (q2, Z, R)
	δ(q2, b) = (q2, b, R) // skip b's (marked/unmarked)
	δ(q2, c) = (q3, Z, L) // found unmarked c -> mark and move left to return
	δ(q2, □) = (q_{reject}, □, R) // no c found -> reject
	State q_3— move left back to left end (blank) then switch to q_0
	δ(q3, a) = (q3, a, L)
	δ(q3, b) = (q3, b, L)
	δ(q3, c) = (q3, c, L)
	δ(q3, X) = (q3, X, L)
	δ(q3, Y) = (q3, Y, L)
	δ(q3, Z) = (q3, Z, L)
	δ(q3, □) = (q0, □, R) // reached left blank: move right to first symbol and go q0
	State q_5— verification: ensure no unmarked a/b/c remain			

δ(q5, X) = (q5, X, R)
δ(q5, Y) = (q5, Y, R)
δ(q5, Z) = (q5, Z, R)
δ(q5, □) = (q_{accept}, □, R) // if only markers and blank -> accept
δ(q5, a) = (q_{reject}, a, R) // unmarked a found -> reject
δ(q5, b) = (q_{reject}, b, R) // unmarked b found -> reject
δ(q5, c) = (q_{reject}, c, R) // unmarked c found -> reject

Sequence of moves (instantaneous descriptions) for input aabbcc
We use notation: …Tape… with the current state shown **in-line** at the head position.
Blank is □. Start: head on leftmost symbol.
Initial tape (underscores show cells; we write compact):

1. q0 a a b b c c □ ...
   Apply δ(q0, a) = (q1, X, R)
2. X q1 a b b c c □ ...
   q1 sees a → skip: δ(q1, a) = (q1, a, R)
3. X a q1 b b c c □ ...
   q1 sees b → mark it: δ(q1, b) = (q2, Y, R)
4. X a Y q2 b c c □ ...
   q2 skips b: δ(q2, b) = (q2, b, R)
5. X a Y b q2 c c □ ...
   q2 sees c → mark: δ(q2, c) = (q3, Z, L)
6. X a Y b Z q3 c □ ... (moved left)

Now move q3 left back to left blank:

7. X a Y q3 b Z c □ ... (L)
8. X q3 a Y b Z c □ ... (L)
9. q3 X a Y b Z c □ ... (L)
10. □ q0 X a Y b Z c □ ... (on left blank → move R and go q0)

Now second pass:

11. X q0 X a Y b Z c □ ... (q0 sees X → skip R)
12. X X q0 a Y b Z c □ ...
13. q0 sees a → δ(q0,a)=(q1,X,R):
    X X q1 Y b Z c □ ... becomes after write:
    X X X q1 Y b Z c □ ... (second a marked)
14. q1 sees Y → skip: X X X Y q1 b Z c □ ...
15. q1 sees b → mark Y: X X X Y Y q2 Z c □ ... head on Z
16. q2 sees Z → skip R: X X X Y Y Z q2 c □ ...
17. q2 sees c → mark Z and L: X X X Y Y Z Z q3 □ ... head moved left
    18–22. q3 moves left to blank:
    ... eventually □ q0 X X X Y Y Z Z □ ... and then move R into q0 at first symbol

Now verification pass:

23. q0 skips X,X,X,Y,Y,Z,Z, reaches blank and transitions to q5:
    ... Z Z q0 □ ... → δ(q0,□)=(q5,□,R) ⇒ ... Z Z □ q5 ...
24. q5 sees □ and therefore accepts: δ(q5,□)=(q_accept,□,R)

Thus the machine **accepts** aabbcc.

	(That sequence shows the marks X,Y,Z replacing corresponding a,b,c and returns to start until all matched.)			
6	Illustrate the concept of Chomsky hierarchy with example **Scheme** Definition of Chomsky Hierarchy (1 Mark) Types (6 Marks) Hierarchy Diagram (3 Marks) **Solution** The **Chomsky Hierarchy** classifies grammars (and the languages they generate) into **four types** based on their generative power and restrictions on their production rules. **1. Type–0: Unrestricted Grammars (Recursively Enumerable Languages)** **Definition:** No restrictions on productions. Left-hand side can be any string with at least one variable. **Recognition Model:** **Turing Machine** **Example:** Grammar that generates $\{a^n b^n c^n \mid n \geq 1\}$ S → aSBC | abc CB → BC bB → bb cC → cc This language is **not context-free**, but Type-0 can generate it. **2. Type–1: Context-Sensitive Grammars (CSGs)** **Definition:** Productions are of the form $$\alpha A \beta \rightarrow \alpha \gamma \beta$$ where **length does not decrease**: $\mid \gamma \mid \geq 1$. **Recognition Model:** **Linear Bounded Automaton (LBA)** **Example:** Language $\{a^n b^n c^n \mid n \geq 1\}$can be generated by: S → aSBC | abc CB → BC bB → bb cC → cc (This grammar is length-increasing → context-sensitive.) **3. Type–2: Context-Free Grammars (CFGs)** **Definition:**	10	CO5	L3

Productions are of the form:
A → α
where **A** is a single nonterminal, **α** is any string of terminals/nonterminals.
Recognition Model:
Pushdown Automaton (PDA)
Example:
Language $\{a^n b^n \mid n \geq 0\}$
S → aSb | ε

4. Type–3: Regular Grammars (Regular Languages)
Definition:
Productions are of the form:

- Right-linear: **A → aB** or **A → a**
- Left-linear: **A → Ba** or **A → a**

Recognition Model:
Finite Automaton (FA)
Example:
Language of even number of 0's:
S → 0A | ε
A → 0S

Hierarchy Diagram
Regular ⊂ Context-Free ⊂ Context-Sensitive ⊂ Recursively Enumerable
Type–3 Type–2 Type–1 Type–0