Indian Association for the Cultivation of Science

(Deemed to be University under *de novo* Category)

Master's/Integrated Master's-PhD Program/ Integrated Bachelor's-Master's Program/PhD Course

Mid-Semester (Sem-III) Examination-Autumn 2025

Subject: **Theory of Computation II** *Subject Code:* **COM 5108**‘

Full Marks: **25** *Time Allotted:* **2 hours**

Q 1. Answer five (5) questions with brief justifications. Only ‘*Yes*’ or ‘*No*’ will not be sufficient. **[5 × 2 = 10]**

(a) Is it true that $\binom{n}{30}$ is of $O(n^k)$ for some k?

Ans. $\binom{n}{30} = \frac{n\cdot(n-1)\cdots(n-29)}{30!} = O(n^{30})$.

(b) Let $\mathcal{P}\mathbb{N}_{cofin} = \{A \subseteq \mathbb{N}_0 : \ \mathbb{N}_0 \setminus A \text{ is finite}\}$, the collection of *cofinite* (complement is finite) subsets of $\mathbb{N}_0 = \{0, 1, 2, \cdots\}$.
Is $\mathcal{P}\mathbb{N}_{cofin}$ countably infinite?

Ans. There is a bijection $f : \mathcal{P}\mathbb{N}_{fin} \to \mathbb{N}_0$, $\emptyset \mapsto 0$ and $\{a_1, a_2, \cdots, a_k\} \mapsto 2^{a_1} + 2^{a_2} + \cdots + 2^{a_k}$. The bijection $g : \mathcal{P}\mathbb{N}_{cofin} \to \mathbb{N}_0$ is, $g(A) = f(\mathbb{N}_0 \setminus A)$. So it is countably infinite.

(c) Let L_1 and L_2 be two *Context-free languages* (CFLs) over Σ.
$L = L_1L_2 = \{x \in \Sigma^* : \ x = uv, \ u \in L_1 \wedge v \in L_2\}$. Is L a CFL?

Ans. Let the CFGs of L_1 and L_2 be $G_1 = (\Sigma, N_1, S_1, R_1)$ and $G_2 = (\Sigma, N_2, S_2, R_2)$. We assume that $N_1 \cap N_2 = \emptyset$. The CFG for L is $G = (\Sigma, N, S, N)$ where $S \notin N_1 \cup N_2$ and $N = \{S \to S_1S_2\} \cup N_1 \cup N_2$.

(d) Let L_1 and L_2 be two decidable languages over Σ.
$L = L_1L_2 = \{x \in \Sigma^* : \ x = uv, \ u \in L_1 \wedge v \in L_2\}$. Is L decidable?

Ans. Let the Turing machines M_1 and M_2 be decide the language L_1 and L_2 respectively. We design the following NTM to decide L.

N: input x

(a) Nondeterministically split $x = uv$.

(b) Copy u and v on two different tapes.

(c) Simulate M_1 on u and M_2 on v.

(d) If both M_1 accepts u and M_2 accepts v, then *accept*.

(e) If there is no split for which both the simulation accepts, then *reject*.

(e) $L_{1000} = \{< M, x >: M$ is a Turing machine that does not halt on input x within 1000 steps $\}$. Is L_{1000} decidable?

Ans. Following is the decider for L_{1000}.
D_{1000} : input $< M, x >$

(a) Simulate M on the input for at most 1000 steps.

(b) If M halts within this, *reject*.

(c) Else *accept*.

(f) A Boolean clause has 10 literals, $C = (l_1 \vee l_2 \vee \cdots \vee l_{10})$. How many clauses in the equivalent 3CNF formula ϕ will be there so that C is satisfiable if and only if ϕ is satisfiable? How many new Boolean variables are introduced?

Ans. The first and the last clauses have two literals from C. Other clauses have one literal from C. So the total number of clauses are $10 - 4 + 2 = 8$. Every clause except the last one has a new literal. So there are 7 new variables.

(g) Let $CLIQUE_{30} = \{< G >: G$ is an undirected graph and it has a clique of size 30 $\}$. Is $CLIQUE_{30}$ in **P**?

Ans. If the graph has n vertices then there are $\binom{n}{30} O(n^{30})$ subsets of vertices of size 30. Testing each of them for clique will take $\binom{30}{2}$ tests. So it can be performed in polynomial time.

Answer any three (3) of the following questions.

[3 × 5 = 15]

Note: *decidable* ≡ *recursive, semi-decidable* ≡ *recursively-enumerable.*

Q 2. **[2 + 3]**

(a) Prove that for any set A there is no *onto* map from A to its *power set* $\mathcal{P}A$.

(b) Let $L_H = \{< M, x >: \ M \text{ is a Turing machine that halts on its input } x\}$. Prove that L_H is undecidable.

Ans.

(a) Suppose $f : A \to \mathcal{P}A$ is an *onto* map. We define a subset B of A as follows:

$$B = \{a \in A : \ a \notin f(a)\}.$$

As f is *onto* and $B \in \mathcal{P}A$, there is a $b \in A$ such that $f(b) = B$. But the contradiction is

$$b \in f(b) = B \text{ if and only if } b \notin f(b) = B.$$

(b) Suppose the Turing machine M_h decides L_H i.e. M_h halts on every input. M_h *accepts* if M halts on x and it *rejects* if M does not halts on x.
We design the following Turing machine using M_h as an oracle.

D: input $< M >$

(i) Call M_h on input $(< M >, < M >)$.

(ii) If M_h accepts, then *reject*, else *accept*.

$$D(< M >) = \begin{cases} \textit{accept} & M \text{ does not halt on } < M > \\ \textit{reject} & M \text{ halts on } < M >. \end{cases}$$

But then there is a contradiction if we feed D with its own description $< D >$.

$$D(< D >) = \begin{cases} \textit{accept} & D \text{ does not halt on } < D > i.e. \\ \textit{reject} & M \text{ halts on } < M >. \end{cases}$$

D halts and accept $< D >$ if and only if D does not halt on $< D >$.

→ **See page 2**

Q 3. **[2 + 3]**

(a) Let $A, B \subseteq \{0,1\}^*$. Define *mapping reducibility* ($\leq_m$) from A to B. What conclusions can you draw about the relative computability (*decidability*, semi-decidability etc.) between A and B?

(b) Show that $\overline{L_H}\{< M, x >:$ the Turing machine M does not halt on $x\}$ is mapping reducible to $\overline{L_{reg}} = \{< M >: L(M)$ is not a regular language$\}$. Is $\overline{L_{reg}}$ *recursive* or *recursively-enumerable* or *not recursively-enumerable*?

Ans.

(a) The set A is mapping reducible to the set B if there is a computable function $f : \{0,1\}^* \to \{0,1\}^*$ so that for all $x \in \{0,1\}^*$, $x \in A$ if and only if $f(x) \in B$.

A cannot be computationally *more difficult* than B.
If B is *decidable* (resp. semi-decidable), then so is A.
If A is *undecidable* (resp. not recursively enumerable) then so B.

(b) Consider the following Turing machine constructed by the computable function $f : \{0,1\}^* \to \{0,1\}^*$ i.e. $< M, x > \mapsto < \overline{M_{reg}} >$.

$\overline{M_{reg}}$: input y

(a) If $y = y^R$ then *accept*.

(b) Simulate M on x.

(c) If the simulation halts then *accept*.

$$L(\overline{M_{reg}}) = \begin{cases} \{0,1\}^* & \text{if } M \text{ halts on } x \\ \{y \in \{0,1\}^* : \ y = y^R\}\} & \text{if } M \text{ does not halt on } x. \end{cases}$$

As $\overline{L_H}$ is not recursively-enumerable, the language $\overline{L_{reg}}$ is also not recursively enumerable.

Q 4. **[4 + 1]**

(a) Give two definitions of the class **NP** and show that they are equivalent.

(b) Is $COMPOSITE = \{n \in \mathbb{N} : \ n \text{ is a composite number}\}$ in **P**?

Ans.

(a) *Definition I:* A *decision problem* $S \subseteq \{0,1\}^*$ is in class **NP** if there is a polynomial time bounded nondeterministic Turing machine that accepts only the elements of S.
Definition II: A *decision problem* $S \subseteq \{0,1\}^*$ is in class **NP** if there is a polynomial $p()$ and a polynomial time bounded deterministic Turing machine V so that

(i) For every $x \in S$ there is an y ($|y| \leq p(|x|)$) such that V accepts (x, y). The string y is called a *witness/proof/certificate* of $x \in S$.

(ii) But for every $x \notin S$, there is no y such that V accepts (x, y).

Equivalence:
If there is a nondeterministic Turing machine N accepting $x \in S$, there is an accepting run (choice of transitions) of N. The description of N and the accepting run of N on x is a possible witness y for a verifier. The length of this witness is $O(poly(|x|))$.
A verifier can be designed that checks the correctness of the transition of N from its start state to the accepting state.

If there is a verifier V for S, then for every $x \in S$, there is a polynomial length certificate. A polynomial time bounded nondeterministic Turing machine N can be designed that generates strings y of required polynomial length and use the verifier to accept or reject $V(x, y)$. If $x \in S$, there is at lease one y generated by N, otherwise there is none.

(b) As it is known that $PRIME \in \mathbf{P}$, the set $COMPOSITE$ is also in **P**.

Q 5. **2 + 3**

(a) Define the class of search problems **PF** (solution can be found in polynomial time) and the class **PC** (solution can be verified in polynomial time).

(b) Prove that, if $\mathbf{PC} \subseteq \mathbf{PF}$ then $\mathbf{NP} = \mathbf{P}$.

Ans.

(a) A search problem corresponding to a polynomial bounded relation $S \subseteq \{0,1\}^* \times \{0,1\}^*$ solvable by a polynomial time algorithm M such that for each $x \in \{0,1\}^*$, $M(x) \in S(x)$ if and only if $S(x) \neq \emptyset$. Otherwise the algorithm halts indicating that there is no solution i.e. $M(x) = \perp$.
The collection of these problems is called **PF**.

A search problem S corresponding to a polynomial bounded relation $S \subseteq \{0,1\}^* \times \{0,1\}^*$ is in **PC** (efficiently checkable) if there is a polynomial time algorithm/TM M such that for every $x, y \in \{0,1\}^*$,

$$M(x,y) = \begin{cases} 1 & \text{if } (x,y) \in S, \\ 0 & \text{if } (x,y) \notin S. \end{cases}$$

(b) Suppose $\mathbf{PC} \subseteq \mathbf{PF}$:
Let $S \in \mathbf{NP}$ and V be the verifier for S.
Define the relation $R_S = \{(x,y) \in \{0,1\}^* \times \{0,1\}^*,\ V(x,y) = 1\}$ i.e. $(x,y) \in R_S$ if and only if $x \in S$. It is a polynomial bounded relation and can be checked in polynomial time by the verifier V, so $R_S \in \mathbf{PC} \Rightarrow R_S \in \mathbf{PF}$ (by the assumption $PC \subseteq PF$).
So the search problem of R_S is efficiently solvable and there is a polynomial time algorithm M_S to search for a solution of $x \in S$.
$M_S(x)$ returns a $y \in R_S(x)$ if $R_S(x) \neq \emptyset$; otherwise it returns $\perp$. The polynomial time decider for S using M_S as an oracle is as follows:
D_x: input x

(a) Call M_S with x as the parameter.

(b) If $M_S(x) \neq \perp$, *accept*.

(c) Else *reject*.

This makes $\mathbf{NP} \subseteq \mathbf{P}$. It is equivalent to say $\mathbf{P} = \mathbf{NP}$ (as $\mathbf{P} \subseteq \mathbf{NP}$).

***** End *****