CLUSTER UNIVERSITY SRINAGAR

SYLLABUS (FYUP UNDER NEP - 2020)

Offered by Department of Information Technology

Semester 5th (Major Course – CT3)

Title: Theory of Computation

Course Code: UGICT22J503
Credits: 6 (Theory: 4, Practical: 2)
Contact Hrs: 120 (Theory: 60, Practical: 60)

Max. Marks: 150
Theory External: 80; Min Marks: 32
Theory Internal (Continuous Assessment): 20 Marks, Min Marks: 08
Practical Experimental Basis= 30, Min. Marks: 12
Practical Experimental (Continuous assessment) = 20, Min. Marks: 08

Course Objectives:
This Automata Theory and Computability course equips 5th semester Computer Applications majors with the fundamentals of theoretical computer science. Students will learn to design finite automata and context-free grammars, analyze computability with Turing machines, and explore complexity theory concepts like P vs. NP, all building a strong foundation for approaching computational problems with a theoretical and analytical lens.

Course Outcome

1. Students will be able to define automata and languages, explaining their role in the theory of computation.
2. Students will demonstrate an understanding of finite automata, including both deterministic and non-deterministic versions, and their equivalence.
3. Students will understand the closure properties of regular languages under various operations and be able to prove the equivalence of NFAs and DFAs.
4. Students will be able to describe and differentiate between Mealy and Moore machines, understanding their applications and significance.
5. Students will understand the formal definition of pushdown automata (PDAs) and their application in recognizing context-free languages.
6. Students will gain a foundational understanding of computability theory, including the Church-Turing Thesis and Turing Machines.
7. Students will be introduced to the P vs NP problem and will be able to identify NP-complete problems, understanding their importance in the theory of computation.

UNIT 1: (15 Hrs)
Introduction to Automata Theory: Automata and Languages. Regular Languages: Finite Automata, regular operations. Deterministic and non-deterministic automata, formal definitions, equivalence of NFAs and DFAs. Closure under the regular operations. Mealy and Moore Machines.
Regular Expressions: Formal definition, equivalence with finite automata.

UNIT 2: (15 Hrs)
Pushdown Automata: Formal definition. Introduction to Grammar and its Hierarchy Context Free Languages: Formal Definition, Examples, Design of CFGs, Ambiguity. Equivalence of PDAs with CFGs.

UNIT 3: (15 Hrs)
Computability Theory. The Church-Turing Thesis: Turing Machines (formal definition, examples).
Recursive and Recursively enumerable languages.

UNIT 4: (15 Hrs)
Complexity Theory. Time Complexity: Asymptotic notations, Class P (polynomial time, examples), The class NP, P vs NP, NP-completeness. Introduction to Space complexity.

Practical Course (2 Credit- 30 Hrs)

1. **Design finite automata:**
    - To accept strings ending with 'a'.
    - To accept strings with exactly two consecutive 'b's.
    - To accept strings with an even number of 'a's.

2. **Convert between DFAs and NFAs:**
    - Convert a given DFA to an equivalent NFA & Vice versa
3. **Regular expressions:**
    - Write regular expressions for simple languages (e.g., strings starting with 'a', ending with 'b').
    - Convert regular expressions to NFAs and vice versa.
4. **Minimize a given DFA.**
5. Design simple Mealy and Moore machines for specific input-output relations.
6. **Design pushdown automata:**
    - To accept the language of palindromes.
    - To accept the language of balanced parentheses.
7. **Context-free grammars:**
    - Write CFGs for simple languages (e.g., arithmetic expressions).
    - Check if a given string is generated by a CFG.
8. **Identify ambiguous grammars and remove ambiguity if possibl**e.
9. **Turing machines:**
    - Design simple Turing machines for basic computations (e.g., incrementing a binary number).
10. **Understand the difference between these language types through examples.:**

Time complexity analysis:
- Analyze the time complexity of simple algorithms (e.g., linear search, binary search).
- Identify problems that can be solved in polynomial time.

11. **NP-complete problems:**
    - Understand the concept of NP-completeness.
    - Recognize examples of NP-complete problems (e.g., satisfiability, traveling salesman).
12. **Space complexity:**
    - Analyze the space complexity of simple algorithms.
    - Understand the concept of space-bounded computations.

SUGGESTED READING:

1. Introduction to the Theory of Computation, Third Edition Michael Sipser, Cengage Leaming, ISBN-13: 978-1-133-18779-0
2. Cohen, D. I. A. (2003). *Introduction to computer theory* (2nd ed., ISBN: 8126513349). New Delhi, India: McGraw-Hill Education.

Formal Languages and Automata Theory by Peter Linz (3rd edition, 2011)