AUTOMATA THEORY AND COMPILER DESIGN

Course Code	**21CS51**	**CIE Marks**	**50**
Teaching Hours/Week (L:T:P: S)	**3:0:0:0**	**SEE Marks**	**50**
Total Hours of Pedagogy	**40**	**Total Marks**	**100**
Credits	**03**	**Exam Hours**	**03**

Course Learning Objectives
CLO 1. Introduce the fundamental concepts of Automata Theory, Formal Languages and compiler design
CLO 2. Principles Demonstrate Application of Automata Theory and Formal Languages in the field of compiler design
CLO 3. Develop understanding of computation through Push Down Automata and Turing Machines CLO 4. Introduce activities carried out in different phases of Phases compiler
CLO 5. Identify the undecidability problems.

Teaching-Learning Process (General Instructions) These are sample Strategies, which teachers can use to accelerate the attainment of the various course outcomes.

1. Lecturer method (L) needs not to be only a traditional lecture method, but alternative effective teaching methods could be adopted to attain the outcomes.

2. Use of Video/Animation to explain functioning of various concepts.

3. Encourage collaborative (Group Learning) Learning in the class.

4. Ask at least three HOT (Higher order Thinking) questions in the class, which promotes critical thinking.

5. Adopt Problem Based Learning (PBL), which fosters students' Analytical skills, develop design thinking skills such as the ability to design, evaluate, generalize, and analyze information rather than simply recall it.

6. Introduce Topics in manifold representations.

7. Show the different ways to solve the same problem with different approaches and encourage the students to come up with their own creative ways to solve them.

8. Discuss how every concept can be applied to the real world - and when that's possible, it helps improve the students' understanding.

Module-1

Introduction to Automata Theory: Central Concepts of Automata theory, Deterministic Finite Automata(DFA), Non- Deterministic Finite Automata(NFA) ,Epsilon- NFA, NFA to DFA Conversion, Minimization of DFA I

Introduction to Compiler Design: Language Processors, Phases of Compilers

Textbook 1: Chapter1 – 1.5, Chapter2 – 2.2,2.3,2.5 Chapter4 –4.4 Textbook 2: Chapter1 – 1.1 and 1.2 Teaching-Learning Process Chalk and board, Active Learning, Problem based learning

Module-2

Regular Expressions and Languages: Regular Expressions, Finite Automata and Regular Expressions, Proving Languages Not to Be Regular

Lexical Analysis Phase of compiler Design: Role of Lexical Analyzer, Input Buffering , Specification of Token, Recognition of Token.

Textbook 1: Chapter3 – 3.1, 3.2, Chapter4- 4.1 Textbook 2: Chapter3- 3.1 to 3.4 Teaching-Learning Process Chalk and board, Active Learning, Demonstration

Module-3

Context Free Grammars: Definition and designing CFGs, Derivations Using a Grammar, Parse Trees, Ambiguity and Elimination of Ambiguity, Elimination of Left Recursion, Left Factoring.

Syntax Analysis Phase of Compilers: part-1: Role of Parser , Top-Down Parsing

Textbook 1: Chapter 5 – 5.1.1 to 5.1.6, 5.2 (5.2.1, 5.2.2), 5.4 Textbook 2: Chapter 4 – 4.1, 4.2, 4.3 (4.3.2 to 4.3.4) ,4.4 Teaching-Learning Process Chalk and board, Problem based learning, Demonstration

Module-4

Push Down Automata: Definition of the Pushdown Automata, The Languages of a PDA.

Syntax Analysis Phase of Compilers: Part-2: Bottom-up Parsing, Introduction to LR Parsing: SLR, More Powerful LR parsers

Textbook1: Chapter 6 – 6.1, 6.2 Textbook2: Chapter 4 – 4.5, 4.6, 4.7 (Up to 4.7.4) Teaching-Learning Process Chalk & board, Problem based learning

Module-5

Introduction to Turing Machine: Problems that Computers Cannot Solve, The Turing machine, problems, Programming Techniques for Turing Machine, Extensions to the Basic Turing Machine Undecidability : A language That Is Not Recursively Enumerable, An Undecidable Problem That Is RE.

Other Phases of Compilers: Syntax Directed Translation- Syntax-Directed Definitions, Evaluation Orders for SDD's. Intermediate-Code Generation- Variants of Syntax Trees, Three-Address Code. Code Generation- Issues in the Design of a Code Generator

Textbook1: Chapter 8 – 8.1, 8.2,8.3,8.4 Chapter 9 – 9.1,9.2 Textbook2: Chapter 5 – 5.1, 5.2, Chapter 6- 6.1,6.2 Chapter 8- 8.1

Course Outcomes

At the end of the course the student will be able to:

CO 1. Acquire fundamental understanding of the core concepts in automata theory and

Theory of Computation

CO 2. Design and develop lexical analyzers, parsers and code generators

CO 3. Design Grammars and Automata (recognizers) for different language classes and become knowledgeable about restricted models of Computation (Regular, Context Free) and their relative powers.

CO 4. Acquire fundamental understanding of the structure of a Compiler and Apply concepts automata theory and Theory of Computation to design Compilers

CO 5. Design computations models for problems in Automata theory and adaptation of such model in the field of compiler

Suggested Learning Resources:

Textbooks

1. John E Hopcroft, Rajeev Motwani, Jeffrey D. Ullman,“ Introduction to Automata Theory, Languages and Computation”, Third Edition, Pearson.
2. Alfred V.Aho, Monica S.Lam,Ravi Sethi, Jeffrey D. Ullman, “ Compilers Principles, Techniques and Tools”, Second Edition,Perason.

Reference:

1. Elain Rich, “Automata,Computability and complexity”, 1st Edition, Pearson Education,2018.
2. K.L.P Mishra, N Chandrashekaran , 3rd Edition , ‘Theory of Computer Science”,PHI,2012.
3. Peter Linz, “An introduction to Formal Languages and Automata “, 3rd Edition, Narosa Publishers,1998.
4. K Muneeswaran, ”Compiler Design”, Oxford University Press 2013.

Weblinks and Video Lectures (e-Resources):

1. https://nptel.ac.in/courses/106/106/106106049/#
2. https://nptel.ac.in/courses/106/104/106104123/
3. https://www.jflap.org/

Activity Based Learning (Suggested Activities in Class)/ Practical Based learning