Course Title : Theory of Computation

Course Code: CSE1008

Class number: AP2025264000421 | AP2025264000424

LTP: 4-0-0

Slot: A1+TA1+TAA1 | A2+TA2+TAA2

Venue: G13 - CB | **G15** - CB

Name of the Instructor: Dr. D. VENKATA LAKSHMI, PhD(Maths), M Tech(CSE), PhD(CSE)
Professor, School of Computer Science & Engineering (SCOPE)
VIT-AP University, Amaravati, Andhra Pradesh, India

Room no: G07- G Central Block.

Email id: venkatalakshmi.d@vitap.ac.in

Mobile no: 8790657430

Objective and Outcome(CO&PO)

VIT-AP UNIVERSITY

Objectives:

1. To understand the essential mathematical foundations of automata theory.

Objective and Outcome(CO&PO)

VIT-AP UNIVERSITY

Objectives:

2. To design, finite state automata and the equivalent regular expression for any given pattern.

Objective and Outcome(CO&PO)

VIT-AP UNIVERSITY

Objectives:

3.To analyze and design Context Free Grammar, Pushdown Automata and Turing Machine.

Objective and Outcome(CO&PO)

VIT-AP UNIVERSITY

Objectives:

4. To understand the difference between decidability and undecidability

NBA PO'S

PO1- Engineering Knowledge

PO2- Problem Analysis

PO3 – Design and Development of Solution

PO4 - Conduct investigation of complex problem

PO5 - Modern Tool usage

PO6 - The engineer and Society

PO7 - Environment, and Sustainability.

PO8 - Ethics

PO9 – Individual and Teamwork

PO10 - Communication

PO11 – Project management and Finance

PO12 - Life-Long Learning

Program Outcomes (POs) for B.Tech Students:

1.**Engineering Knowledge:** Apply mathematics, science, and engineering fundamentals to solve complex problems.
2.**Problem Analysis:** Identify, formulate, and analyse engineering problems using a structured approach.
3.**Design/Development of Solutions:** Design innovative and sustainable solutions for complex engineering challenges.

Program Outcomes (POs) for B.Tech Students:

4.**Conduct Investigations of Complex Problems:** Use research methods, including experiment design and data analysis, to draw conclusions.

5.**Modern Tool Usage:** Apply modern engineering tools and techniques to model and solve engineering tasks, recognizing their limitations.

Program Outcomes (POs) for B.Tech Students:

6.**The Engineer and Society:** Assess societal, health, safety, and legal issues and their implications in engineering practice.
7.**Environment and Sustainability:** Understand the impact of engineering solutions on the environment and promote sustainable practices.

Program Outcomes (POs) for B.Tech Students:

8.**Ethics:** Uphold ethical principles and professional responsibilities in engineering. Individual and Team
9.**Work:** Function effectively as an individual and as part of diverse and multidisciplinary teams.
10.**Communication:** Communicate complex engineering concepts effectively through reports, presentations, and documentation.

Program Outcomes (POs) for B.Tech Students:

11.**Project Management and Finance:** Demonstrate management skills and apply engineering principles to lead projects efficiently.

12. **Life-Long Learning:** Recognize the need for lifelong learning to adapt to technological changes and innovations.

Programme Outcomes (PO): (Common for all the programmes)

After successful completion of the program a student is expected to have abilities to:

PO1. Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

PO2. Problem analysis: Identify, formulate, research literature, and analyse complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

PO3. Design/development of solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

PO4. Conduct investigations of complex problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

PO5. Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modelling to complex engineering activities with an understanding of the limitations.

PO6. The engineer and society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues, and the consequent responsibilities relevant to the professional engineering practice.

PO7. Environment and sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

PO8.Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

PO9. Individual and teamwork: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

PO10. Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

PO11. Project management and finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

PO12. Life-long learning: Recognize the need for and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

CO's Mapping with PO's and PEO's

VIT-AP UNIVERSITY

Course Outcomes	Course Outcome Statement	PO's
CO1	Understand the abstract machines, computation and basic properties of formal languages, finite automata	**PO1, PO3**
CO2	Familiarity with regular language, regular expression, context-free grammars and can make grammars to produce strings from a specific language	**PO1, PO2, PO3**
CO3	Can differentiate regular, context-free and recursively enumerable languages and can compare and analyse to different computational models.	**PO3, PO4, PO5**
CO4	Acquire concepts relating to computational models including decidability and intractability and can identify limitations of some computational models and possible methods of proving them	**PO3, PO4, PO5**
		Total Hours of Instructions: 60

Course Content (60 hrs)

VIT-AP UNIVERSITY

Module No. 1	Finite Automata (FA)	11 Hours
Mathematical preliminaries and notations-Finite Automata-Deterministic Finite Automata – Non-Deterministic Finite Automata and equivalence with DFA - Epsilon transitions – Minimization of Finite Automata and its applications.		
Module No. 2	Regular Expressions (RE)	9 Hours
Definition, Operators of regular expression and their precedence- Algebraic laws for Regular expressions and Kleene's Theorem - Regular expression to FA- DFA to Regular expression- Pumping Lemma for regular Languages		
Module No. 3	RE & Context Free Grammar (CFG)	9 Hours
Closure properties of Regular Languages- Decision properties of Regular Languages -Context-Free Grammar (CFG) – Derivation Trees – Ambiguity in Grammars and Languages		
Module No. 4	Pushdown Automata (PDA)	11 Hours
Definition, Graphical Notation, Instantaneous Descriptions of PDA- Acceptance by Final state, Acceptance by empty stack, Deterministic PDA- CFG to PDA - PDA to CFG		
Module No. 5	Normal forms of CFG	9 Hours
Normal forms for CFGs: CNF and GNF, Closure properties of CFLs, Decision Properties of CFLs: Emptiness, Finiteness and Membership, Pumping lemma for CFLs		
Module No. 6	Turing Machine	11 Hours
Turing machines (TM): Basic model, definition and representation, Instantaneous Description, Language acceptance by TM- Undecidable problems about TMs-Post correspondence problem (PCP)-Modified PCP- Introduction to recursive function theory - Introductory ideas on Time complexity of deterministic and nondeterministic Turing machines		

Books and study Materials

VIT-AP UNIVERSITY

Text Books

1. J.E.Hopcroft, R.Motwani and J.D Ullman, "Introduction to Automata Theory, Languages and Computations", Pearson Education, 3rd Edition, 2013.

References

1. Micheal Sipser, "Introduction of the Theory and Computation", Cengage Learning, 3rd edition, 2014.
2. Martin J. C., "Introduction to Languages and Theory of Computations", McGraw Higher Ed, 3rd edition, 2009.
3. K.L.P. Mishra and N.Chandrasekaran, "Theory of Computer Science: Automata, Languages and Computation", PHI, Third edition, 2009.
4. Papadimitriou, C. and Lewis, C. L., "Elements of the Theory of Computation", Pearson, 2nd edition, 2015.

Mode of Evaluation

VIT-AP UNIVERSITY

Mode of Evaluation	**Continuous Assessment Tests-60%, Practical Assesment-40%**
	Continuous Assessment Test-1 (CAT -1) 15%
	Continuous Assessment Test-2 (CAT -2) 15%
	Continuous Assessment Test-3 (FAT) 40%
	Cumulative class Assessments 5%
	(Home works)
	Assignments – 2 and 1 Quiz 25%
	(After completion of 2 modules)

VIT-AP UNIVERSITY

Expected Outcome:

On completion of the course, students will have the ability to

1. Understand the abstract machines and computation, formal languages, finite automata, grammars, pushdown automata and Turing machines.
2. To explore the theoretical foundations of computer science from the perspective of formal languages and classify machines by their power to recognize languages.
3. Demonstrate the knowledge of mathematical models of computation and describe how they relate to formal languages.
4. Derive an appropriate model of computation for a given language and vice versa.
5. Infer the equivalence of languages described using different automata or grammars.
6. Distinguish the computability power of automata and their limitations.
7. Understand the of key notions, such as computability, decidability and complexity

Prerequisites:

Set theory

Sets and operations on sets

Relations and classification of relations

Equivalence relations and partitions

Functions operations of functions

Fundamentals of logic

Graph theory

Algorithms and data structures at the level of an introductory programming sequence.

Mathematical induction and its applications

Home work Rubrics:

One homework every week: Assigned on Tuesday and due the following Monday (at 11 PM)
No late home works.

Home works (Weightage 5 %)

Quiz and Assignments

Will be in class (and maybe online on Test portal or Google form/Classroom)
Goal: To help you ascertain your understanding of the material.

Quiz : (25 Marks)(Weightage 10 %)
Assignment 1 for 30 Marks. (Weightage 10 %)
Assignment 2 for 30 Marks. (Weightage 5 %)
Home works (Weightage 5 %)

VIT-AP UNIVERSITY

Examinations: Tentative Dates

CAT-1: 2nd to 10th February 2026 (Monday to Tuesday)
CAT-2: 23rd - 30th March 2026 (Monday to Monday)

Commencement of FAT for Theory component / Courses: 4th May 2026 (Monday)

Last Instructional day: 2nd May 2026 (Saturday)

CAT will only test material since the previous exam.

Final Exam will test all the course material.

What, Why, and How of Studying Theory of Computation (ToC)
(B.Tech Program Level)

1. WHAT is Theory of Computation?

Theory of Computation is a foundational Computer Science course that studies **what problems can be solved by computers, how efficiently they can be solved, and what limits all computational models have**.
It focuses on:
•**Formal models of computation:** Finite Automata, Pushdown Automata, Turing Machines
•**Languages and grammars:** Regular, Context-Free, and Recursively Enumerable languages
•**Computability:** What is solvable and what is impossible to compute
•**Complexity:** How much time and space problems require
ToC formalizes the **mathematical principles behind computation**.

2. WHY study Theory of Computation ?

a) Builds strong problem-solving foundations

ToC strengthens analytical thinking, abstraction, and logical reasoning—skills crucial for algorithms, programming, and system design.

b) Helps understand limits of computers

•Some problems are *unsolvable* (e.g., Halting Problem)

•Some problems are *intractable* (e.g., NP-hard problems)

This prevents unrealistic expectations in software engineering.

c) Enables better algorithmic thinking

Understanding regular languages, grammars, and automata helps in:

•Compiler design

•Text processing

•Machine learning models for sequential data

•System design (protocols, pattern matching, formal verification)

d) Supports advanced courses and career paths

ToC is the base for:

- Compiler Construction
- Artificial Intelligence
- Formal Verification
- Cyber Security
- Data Science
- Operating Systems and Algorithms

e) Essential for competitive programming and interviews

Concepts like state machines, grammar parsing, NP-completeness are frequently tested in competitive programming, coding interviews, and tech hiring assessments.

Why to study/learn Theory of Computation:

Understanding Computational Limits: Determines what problems can be solved by computers (decidability) and what problems are unsolvable (undecidability).Helps in identifying the boundaries of computational feasibility.

Design of Programming Languages: Provides a formal foundation for defining syntax and semantics of programming languages using grammars and automata. Enables the development of compilers and interpreters.

Development of Algorithms: Offers insights into the complexity of algorithms and helps in designing efficient solutions for computational problems. Categorizes problems based on their computational complexity (P, NP, NP-complete, etc.).

Pattern Recognition and Text Processing: Automata and regular expressions are widely used in text editors, search engines, and data validation (e.g., syntax checking for email addresses).

Why to study/learn Theory of Computation:

Artificial Intelligence and Machine Learning: Supports the modeling of decision-making systems and parsing algorithms for natural language processing (NLP).

Formal Verification and Software Testing: Used in verifying the correctness of hardware and software systems. Provides tools like model checking to ensure systems meet desired specifications.

Cryptography and Security: Helps in understanding computational hardness, a basis for designing secure cryptographic protocols.

Network Protocols and Finite State Machines: Used in designing communication protocols and finite state systems in networking.

Robotics and Embedded Systems: Automata theory is employed to design control systems and task execution models in robots and embedded devices.

Bioinformatics and Computational Biology: Supports the study of biological sequences using algorithms derived from automata and formal languages.

3. HOW is Theory of Computation studied effectively?

a) Understand through Models, not Memorization

Students should focus on **concepts** like:

•How automata recognize languages

•How grammars generate strings

•How Turing machines simulate computation

Visual reasoning > rote learning.

b) Practice drawing and constructing automata

Hands-on practice includes:

•DFA/NFA construction

•Minimization

•Designing CFGs

•Converting between representations

These build conceptual clarity.

c) Learn by mapping real problems to formal models

Examples:

•Regex → Finite Automata

•Programming languages → CFGs

•Algorithmic problems → Turing machines or reductions

Connecting theory with real systems improves application skills.

d) Use tools and simulators

Automata/SIM, JFLAP, or online automata simulators help students visualize transitions and debug their models.

e) Work on incremental problems

Start with simple languages → build automata → add constraints → check closure properties → apply pumping lemmas.

f) Discuss proofs and reasoning

•Equivalent automata

•Non-regularity

•Reducibility

improve mathematical maturity.

Evolution of Theory of Computation — Timeline Overview

Theory of Computation evolved in 6 major stages:
Logic Foundations (1800–1930) – Boolean logic, formal reasoning.
Computability (1930s) – Turing, Church; limits of computation.
Formal Languages & Automata (1950s) – Chomsky hierarchy, automata.
Complexity Theory (1960s–70s) – P, NP, NP-complete problems.
Practical Adoption (1970s–2000s) – compilers, verification, linguistics.
Modern ToC (2000s–now) – Cryptography, ML theory, quantum computing.

Logic Foundations (1800s–1930s)

- Boolean algebra (Boole)
- Formal logic systems (Frege, Russell)
- Hilbert's Entscheidungsproblem – "Can every problem be solved mechanically?"

Computability Revolution (1930s)

- Church's λ-calculus
- Turing Machine model
- Church–Turing Thesis
- Discovery of unsolvable problems (Halting Problem)

Automata & Language Theory (1950s–60s)

- Chomsky Hierarchy (Regular → CFL → CSL → RE)
- Finite Automata, PDA, Linear-bounded Automata
- Foundations for compilers & language processing

Complexity Theory (1960s–70s)

- Time & space complexity classes
- NP-Completeness (Cook & Karp)
- The “P vs NP” challenge emerges

Practical Applications (1970s–2000s)

- Automata in compilers, regex, lexical analysis, text processing
- Model checking & verification
- State machines in hardware, networks, protocols

Modern ToC (2000s–Present)

- Cryptography based on computational hardness
- Quantum computation (BQP)
- Computational learning theory, ML computation models
- Bio-computing & distributed complexity

ToC evolved from pure mathematical logic to a central pillar of modern computer science. Understanding this evolution helps to appreciate why ToC concepts remain essential across programming, algorithms, AI, security, and system design.”