Department of Computer Science and Engineering

Details of Lesson Plan

S.No.	Particulars	Details
1.	Course Name	Theory of Computation
2.	Course Code	COM-604
3.	Academic Year	2024-25
4.	Semester	6th
5.	Number of Lesson plans	47
6.	Faculty Assigned	Dr. Rajneet Kaur

Faculty Signature

Lesson Plan No. 1.1	Course Name: Theory of Computation Topic Name: Introduction to Theory of Computation	Course No.: COM-604

Objectives	At the end of the lesson, the student shall be able to: a. Understand the mathematical preliminaries behind formal languages. b. Articulate the concept of fsm c. Understand the basic concepts of Strings, Alphabets, and Languages. d. Understand the properties of union, intersection, complement and difference. e. Illustrate different computation devices and models.
Teaching Aids (if any)	a. Chalk and talk b. Video of NPTEL
Teaching Development	1. **Introduction** (5 minutes) - Ask questions Which example can you think of automata in daily life? Can we say pen and pencil are the simplest examples of a computational device? Introducing the concept of the turing machine. Show the definition on the slide. - Discuss the concept of computational models with turing machines - Introduce the formal definition of formal language by https://www.iitg.ac.in/dgoswami/Flat-Notes.pdf Highlight the important applications of automata. 2. **Development** (30 minutes) a. Theory of Computation preliminaries - Introduce the concept of computational devices. - Show video of introduction to TOC https://www.youtube.com/watch?v=SV57Yv8BXBc&t=52s - Introduce concept of abstract machine and computational models - Introduce the concepts of alphabet, string and language. - Illustrate concepts of the alphabet, string and language with the help of examples. - Explain the concept of Kleene Star with example. b. Finite tape - Explain the components of finite tape - Illustrate the input string computation - Give examples to illustrate the concept of Finite Tape.

	3. **Exercise** (5 minutes) – Give different example to solve for properties : - If $L_1 = \{0,1,01\}$ and $L_2 = \{1,00\}$, then $L_1L_2 = \{01,11,011,000,100,0100\}$. - Kleene star of the language {01} is $\{\varepsilon,01,0101,010101,...\} = \{(01)_n \mid n \geq 0\}$. - The set of all strings over some alphabet Σ with even number of a's is $\{x \in \Sigma* \mid\mid x\mid_a = 2n, \text{ for some } n \in N\}$.
Closure	1. Summarize the Lesson Learning Outcomes and get affirmation from students on these. 2. Suggested Reading - https://www.iitg.ac.in/dgoswami/Flat-Notes.pdf 3. Homework - What are the application of automata? Spend 5 minutes to wrap up and consolidate the learnings
Evaluation	1. Reflective Questions Which example can you think of automata in daily life? Spend 5 minutes evaluating student assimilation of the lesson contents

Lesson Plan No. 1.2	Course Name: Theory of Computation Topic Name: Generalized Transition Graphs	Course No.: COM-604

Objectives	At the end of the lesson, the student shall be able to: a. Understand the concept of Generalized Transition Graphs (GTGs) as an extension of Finite Automata (FA) to model languages with multiple accept states. b. Learn how GTGs can represent languages that cannot be represented by either Deterministic Finite Automata (DFA) or Nondeterministic Finite Automata (NFA). c. Gain proficiency in constructing GTGs from regular expressions and vice versa, to aid in language recognition and conversion. d. Explore the relationship between GTGs and regular languages, understanding the conditions under which a language can be represented by a GTG.
Teaching Aids (if any)	a. Chalk and talk b. Video of NPTEL
Teaching Development	1. **Introduction** (5 minutes) - Ask questions What is a Generalized Transition Graph (GTG), and how does it differ from a regular Transition Graph? How can GTGs be used to represent the behavior of a system or process? What are the advantages of using GTGs over other modeling techniques? Can you give an example of a real-world application where GTGs would be useful? How do you define the alphabet and states in a GTG? What is the significance of the initial state and accepting states in a GTG? How are transitions represented in a GTG, and what do they signify? 2. **Development** (30 minutes) a. Introduction to Generalized Transition Graphs (GTG) Define GTGs as a generalization of Finite State Machines (FSMs) to represent languages that are not necessarily regular. Explain the need for GTGs in representing languages beyond the regular languages. b. Components of GTGs Nodes: Represent states in the machine. Edges: Represent transitions between states labeled with input symbols. Accepting states: States that indicate the machine has accepted the input. c. Formal Definition:

	Define a GTG as a 5-tuple (Q, Σ, Δ, q0, F), where: Q is a finite set of states. Σ is the input alphabet. Δ is the transition relation. q0 is the initial state. F is a set of final states. d. Transition Function Δ: Explain that Δ is a partial function that maps a state and an input symbol to a set of states. Discuss how Δ can represent non-determinism, where multiple states are possible for a given input and state.
Closure	1. Summarize the Lesson Learning Outcomes and get affirmation from students on these. 2. Suggested Reading - https://www.iitg.ac.in/dgoswami/Flat-Notes.pdf 3. Homework Explain the need for GTGs in representing languages beyond the regular languages. Spend 5 minutes to wrap up and consolidate the learnings
Evaluation	1. Reflective Questions: Define GTGs as a generalization of Finite State Machines Spend 5 minutes evaluating student assimilation of the lesson contents

Lesson Plan No. 1.3	Course Name: Theory of Computation Topic Name: Concept of Automata, strings, Language Deterministic Finite Automata (DFA)	Course No.: COM-604

Objectives	At the end of the lesson, the student shall be able to: a. Understand the concept of a Deterministic Finite Automaton (DFA) and its components, including states, transitions, alphabet, and accepting states. b. Use a DFA to recognize whether a given input string belongs to a specified language. c. To represent a DFA using a transition diagram and a transition table. d. Explore real-world applications of DFAs, such as lexical analysis in compilers and pattern matching in text processing.
Teaching Aids (if any)	a. Chalk and talk b. Video of NPTEL
Teaching Development	1. **Introduction** (5 minutes) - Ask questions What are the components of a DFA and how are they represented? How does a DFA differ from a Non-Deterministic Finite Automaton (NFA)? How do you define the language accepted by a DFA? How do you represent the transitions of a DFA? What is the significance of the accepting states in a DFA? How do you use a DFA to recognize whether a given string belongs to a specified language? 2. **Development** (30 minutes) a. Introduction to DFAs Provide a brief overview of DFAs, their components, and their role in formal language theory. b. DFA Representation Explain how to represent a DFA using a transition diagram and a transition table. c. Formal Definition Present the formal definition of a DFA, including the formal notation used to describe its components. d. Language Recognition Demonstrate how a DFA can be used to recognize whether a given input string belongs to a specified language. e. Designing DFAs Provide examples and practice problems for designing DFAs for simple language recognition tasks.

	f. DFA vs. NFA Discuss the differences between DFAs and NFAs, highlighting the concept of determinism. g. Applications Explore real-world applications of DFAs, such as lexical analysis in compilers and pattern matching in text processing. h. Complexity Discuss the computational complexity of DFAs and their limitations in recognizing certain types of languages.
Closure	1. Summarize the Lesson Learning Outcomes and get affirmation from students on these. 2. Suggested Reading - https://www.iitg.ac.in/dgoswami/Flat-Notes.pdf 3. Homework - Design the finite automata for the language for input symbol {0,1} that starts with 01. Spend 5 minutes to wrap up and consolidate the learnings
Evaluation	1. Reflective Questions Define NFA with example Spend 5 minutes evaluating student assimilation of the lesson contents

Lesson Plan No. 1.4	Course Name: Theory of Computation Topic Name: Simpler Notations for DFA	Course No.: COM-604

Objectives	At the end of the lesson, the student shall be able to: a. Understand the concept of deterministic finite automata (DFA) and their use in recognizing regular languages. b. Learn about the transition function, state set, start state, and accept states in a DFA. c. Introduce simpler notations for representing DFAs, such as state transition diagrams and transition tables. d. Practice converting between different notations for DFAs to enhance understanding. e. Explore the concept of DFA minimization and its importance in reducing the number of states in a DFA.
Teaching Aids (if any)	a. Chalk and talk b. Video of NPTEL
Teaching Development	1. **Introduction** (5 minutes) - Ask questions What is a DFA (Deterministic Finite Automaton) and what purpose does it serve in computer science? How are states, transitions, and the alphabet represented in a DFA? What is the significance of the initial state and the accepting states in a DFA? How can we simplify the representation of a DFA using state diagrams? Discuss the concept of equivalent DFAs. When are two DFAs considered equivalent? Can you explain the concept of a transition function in DFAs? How does it determine the next state of the automaton? 2. **Development** (30 minutes) a. Introduction to Simplified Notations Start by introducing the concept of simplifying the representation of DFAs to make them easier to understand and work with. b. State Naming Conventions Discuss the use of simpler state names (e.g., A, B, C) instead of more complex names to reduce cognitive load. c. Transition Table Abbreviations Explain how to use abbreviations in the transition table for easier readability and comprehension (e.g., using 'A' for state A instead of writing the full state name). d. State Diagrams with Simple Labels Show examples of state diagrams with simple labels on transitions (e.g., 0, 1) instead of complex symbols or expressions.

	e. Reducing State Diagram Complexity Discuss techniques for simplifying state diagrams, such as grouping states with similar transitions or eliminating redundant states. f. Minimization of DFA Introduce the concept of DFA minimization to reduce the number of states and transitions while preserving the language accepted by the DFA.
Closure	1. Summarize the Lesson Learning Outcomes and get affirmation from students on these. 2. Suggested Reading - https://www.iitg.ac.in/dgoswami/Flat-Notes.pdf 3. Homework How are states, transitions, and the alphabet represented in a DFA? What is the significance of the initial state and the accepting states in a DFA? Spend 5 minutes to wrap up and consolidate the learnings
Evaluation	1. Reflective Questions What is purpose of DFA? Spend 5 minutes evaluating student assimilation of the lesson contents

Lesson Plan No. 1.5	Course Name: Theory of Computation Topic Name: The language of DFA and Facts in designing procedure of FA	Course No.: COM-604

Objectives	At the end of the lesson, the student shall be able to: a. Understand the role of DFA in recognizing regular languages. b. Identify key components of a DFA, including states, transitions, and the alphabet. c. Explore the concept of a DFA's language as the set of all strings accepted by the DFA. d. Discuss the significance of the initial state and accepting states in defining the language of a DFA. e. Analyze examples of DFAs and their corresponding languages to deepen understanding.
Teaching Aids (if any)	a. Chalk and talk b. Video of NPTEL
Teaching Development	1. **Introduction** (5 minutes) - Ask questions How is the language of a DFA defined, and what role does it play in automata theory? Can you describe the components of a DFA that are essential for defining its language? How can you determine whether a string is accepted or rejected by a DFA? What is the significance of the transition function in a DFA, and how does it contribute to the DFA's language recognition capability? What are some key facts or principles to consider when designing a DFA for a specific language? How do you design a DFA for a given regular language? Can you outline the steps involved in this process? 2. **Development** (30 minutes) a. Languages Recognized by DFA Discuss how DFAs recognize languages, i.e., how a DFA processes input and determines whether it is accepted or rejected. Illustrate with examples of DFAs recognizing specific languages (e.g., binary strings with an even number of 0s). b. Facts in Designing DFA Cover key facts and principles in designing DFAs, such as the Myhill-Nerode theorem. Explain how to construct a DFA to recognize a given language, emphasizing the importance of state minimization for efficiency. c. Procedure for Designing DFA Outline a step-by-step procedure for designing a DFA for a given language:

	Define the states and alphabet. Determine the transition function. Identify the start state(s). Specify the accepting states. Provide examples and exercises to practice designing DFAs.
Closure	1. Summarize the Lesson Learning Outcomes and get affirmation from students on these. 2. Suggested Reading - https://www.iitg.ac.in/dgoswami/Flat-Notes.pdf 3. Homework - Procedure for Designing DFA Spend 5 minutes to wrap up and consolidate the learnings
Evaluation	1. Reflective Questions How can you determine whether a string is accepted or rejected by a DFA? Spend 5 minutes evaluating student assimilation of the lesson contents

Lesson Plan No. 1.6	Course Name: Theory of Computation Topic Name: Extending Transition Function to Strings in DFA	Course No.: COM-604

Objectives	At the end of the lesson, the student shall be able to: a. Understand the concept of a Deterministic Finite Automaton (DFA) and its transition function. b. Explore the limitations of the DFA transition function with single inputs. c. Introduce the concept of extending the DFA transition function to handle strings as inputs. d. Learn how to define the transition function for strings in a DFA. e. Practice constructing DFAs with transition functions that accommodate strings. f. Analyze the relationship between the transition function for single inputs and strings in a DFA.
Teaching Aids (if any)	a. Chalk and talk b. Video of NPTEL
Teaching Development	1. **Introduction** (5 minutes) - Ask questions What is the purpose of extending the transition function to strings in a Deterministic Finite Automaton (DFA)? How does the transition function handle input strings in a DFA? Can you explain the process of extending the transition function to accept strings in a DFA? What role does the extended transition function play in determining the acceptance of a string by a DFA? How does the extended transition function help in simulating the computation of a DFA on a given input string? Can you provide an example of how the transition function is extended to strings in a DFA? What are the advantages of extending the transition function to strings in a DFA? 2. **Development** (30 minutes) a. Transition Function in DFA Explain the transition function δ that maps each state and input symbol to the next state. b. Transition Function for Strings Introduce the concept of extending the transition function to handle strings, which involves iteratively applying the transition function for each symbol in the input string. c. Formal Definition Provide a formal definition of the extended transition function

	δ* that takes a state and a string as input and returns the resulting state after processing the entire string.
Closure	1. Summarize the Lesson Learning Outcomes and get affirmation from students on these. 2. Suggested Reading - https://www.iitg.ac.in/dgoswami/Flat-Notes.pdf 3. Homework - Provide a formal definition of the extended transition function Spend 5 minutes to wrap up and consolidate the learnings
Evaluation	1. Reflective Questions How does the transition function handle input strings in a DFA? Spend 5 minutes evaluating student assimilation of the lesson contents

Lesson Plan No. 1.7	Course Name: Theory of Computation Topic Name: Designing the DFA, NFA, States, Transition Tables	Course No.: COM-604

Objectives	At the end of the lesson, the student shall be able to: a. Understand the basic concepts of finite automata, including states, transitions, and input symbols. b. Design a deterministic finite automaton (DFA) from a given set of requirements or rules. c. Design a nondeterministic finite automaton (NFA) from a given set of requirements or rules. d. Convert a nondeterministic finite automaton (NFA) to an equivalent deterministic finite automaton (DFA).
Teaching Aids (if any)	a. Chalk and talk b. Video of NPTEL
Teaching Development	1. **Introduction** (5 minutes) - Ask questions What is the purpose of designing a Deterministic Finite Automaton (DFA)? How does a DFA differ from an NFA in terms of design and functionality? What are the key components of a DFA and an NFA, and how do they contribute to the design? Can you describe the process of designing a DFA for a given language? What are the advantages and disadvantages of using an NFA over a DFA in certain language recognition tasks? How does the choice between a DFA and an NFA affect the complexity of a language recognition problem? 2. **Development** (30 minutes) a. Introduce the concept of designing DFAs and NFAs as a way to recognize languages. b. Discuss the process of designing a DFA, starting from defining the states, alphabet, transitions, and accepting states. c. Explain the differences between designing a DFA and an NFA, focusing on the non-deterministic choices in NFAs. d. Provide examples of simple languages and guide students through the process of designing DFAs and NFAs to recognize these languages. e. Discuss the limitations of DFAs in recognizing certain languages and how NFAs can overcome some of these limitations. f. Illustrate the concept of epsilon transitions in NFAs and how they can simplify the design of certain language recognizers.

Closure	1. Summarize the Lesson Learning Outcomes and get affirmation from students on these. 2. Suggested Reading - https://www.iitg.ac.in/dgoswami/Flat-Notes.pdf 3. Homework - Explain the differences between designing a DFA and an NFA, focusing on the non-deterministic choices in NFAs. Spend 5 minutes to wrap up and consolidate the learnings
Evaluation	1. Reflective Questions Why do we design NDFA Spend 5 minutes evaluating student assimilation of the lesson contents

Lesson Plan No. 1.8	Course Name: Theory of Computation Topic Name: NFA without E moves	Course No.: COM-604

Objectives	At the end of the lesson, the student shall be able to: a. Understand the concept of Non deterministic finite automata b. Define the formal definition of NFA c. Implement the graphical representation of NFA d. Illustrate the differences between NFA and DFA
Teaching Aids (if any)	a. Chalk and talk b. Video of NPTEL c. PPT
Teaching Development	1. **Introduction** (5 minutes) - Explain the homework questions on DFA. - Introducing the concept of transition diagram and transition states for NFA. - Articulate the computation of NFA. - Implement the transition diagrams for a few examples. - Introduce the concept of NFA construction with https://www.youtube.com/watch?v=wgC3BCyjzbM&list=PL_Si_t2-nG7cMHfpLOZ88TgEPbXp489HZ&index=5 2. **Development** (30 minutes) a. Practise Problems - NFA with ∑ = {0, 1} accepts all strings with 01. - Design an NFA with ∑ = {0, 1} in which double '1' is followed by double '0'. - Design an NFA in which all the string contains a substring 1110 - Introduce concept of Transition state, transition functions, initial state, final state, set of states for NFA. b. Transition Table - Illustrate the representation of transition table - Illustrate construction of transition tables for the given examples. 3. **Exercise** (5 minutes) – Solve the following for both transition table and diagram:

	- Design an NFA with ∑ = {0, 1} accepts all string in which the third symbol from the right end is always 0.
Closure	1. Summarize the Lesson Learning Outcomes and get affirmation from students on these. 2. Suggested Reading - https://www.iitg.ac.in/dgoswami/Flat-Notes.pdf 3. Homework Design a NFA with ∑ = {0, 1} accepts the strings with an even number of 0's followed by single 1. Spend 5 minutes to wrap up and consolidate the learnings
Evaluation	1. Reflective Questions Explain the **NFA without E moves with example** Spend 5 minutes to evaluate student assimilation of the lesson contents

Lesson Plan No. 1.9	Course Name: Theory of Computation Topic Name: Conversions NFA to DFA	Course No.: COM-604

Objectives	At the end of the lesson, the student shall be able to: a. Learn the process of converting a DFA to an equivalent NFA. b. Explore the need for converting between DFA and NFA. c. Convert between DFA and NFA through examples and exercises. d. Apply the conversion techniques to solve problems related to regular languages and automata theory. e. Analyze and compare the complexity of operations on DFA and NFA.
Teaching Aids (if any)	a. Chalk and talk b. Video of NPTEL
Teaching Development	1. **Introduction** (5 minutes) - Ask questions What is the purpose of converting a Non-deterministic Finite Automaton (NFA) to a Deterministic Finite Automaton (DFA)? How does the conversion process from NFA to DFA help in simplifying the automaton? What are the key differences between NFA and DFA that make the conversion necessary? Can every NFA be converted into an equivalent DFA? Why or why not? What is the significance of the subset construction method in converting an NFA to a DFA? 2. **Development** (30 minutes) a. Converting NFA to DFA Present the step-by-step process of converting an NFA to a DFA. Explain the subset construction method or any other relevant conversion technique. b. Converting DFA to NFA Explain the concept of converting a DFA to an NFA. Discuss how the determinism of the DFA is relaxed to create a nondeterministic structure. c. Example Problems Provide examples of converting NFA to DFA and DFA to NFA. Include both simple and complex examples to illustrate the conversion process. d. Application in Theory and Practice Discuss the importance of understanding these conversions in automata theory. c. Highlight practical applications such as pattern matching, lexical analysis in compilers, etc.

Closure	1. Summarize the Lesson Learning Outcomes and get affirmation from students on these. 2. Suggested Reading - https://www.iitg.ac.in/dgoswami/Flat-Notes.pdf 3. Homework How does the epsilon-closure concept play a role in the conversion of NFA to DFA? 3. Spend 5 minutes to wrap up and consolidate the learnings
Evaluation	1. Reflective Questions Briefly explain the steps to convert the NFA to DFA Spend 5 minutes evaluating student assimilation of the lesson contents

Lesson Plan No. 1.10	Course Name: Theory of Computation Topic Name: Equivalence of NFA and DFA	Course No.: COM-604

Objectives	At the end of the lesson, the student shall be able to: a. Explore the concept of equivalence between DFA and NFA. b. Understand the proof of equivalence between DFA and NFA. c. Learn the implications of the equivalence between DFA and NFA in terms of language recognition. d. Practice converting an NFA to an equivalent DFA. e. Apply the concept of equivalence between DFA and NFA to solve problems related to automata theory.
Teaching Aids (if any)	a. Chalk and talk b. Video of NPTEL c. PPT
Teaching Development	1. **Introduction** (5 minutes) - Ask questions What is a Deterministic Finite Automaton (DFA), and what are its key components? What is a Nondeterministic Finite Automaton (NFA), and how does it differ from a DFA? How can we prove that for every NFA, there exists an equivalent DFA that recognizes the same language? What are the main steps involved in converting an NFA to an equivalent DFA? How do we show that a language is regular by using the equivalence between NFAs and DFAs? 2. **Development** (30 minutes) a. Equivalence of NFA and DFA: State the theorem: "For any NFA, there exists an equivalent DFA that recognizes the same language." Discuss the significance of this theorem in automata theory. b. Converting NFA to DFA Explain the steps involved in converting an NFA to an equivalent DFA. Provide examples to illustrate the conversion process. c. Converting DFA to NFA Explain the steps involved in converting a DFA to an equivalent NFA. Provide examples to illustrate the conversion process. d. Proof of Equivalence Outline the proof that shows the equivalence between NFA and DFA. Discuss the implications of the proof on the computational power of

	NFA and DFA.
Closure	1. Summarize the Lesson Learning Outcomes and get affirmation from students on these. 2. Suggested Reading - https://www.iitg.ac.in/dgoswami/Flat-Notes.pdf 3. Homework Outline the proof that shows the equivalence between NFA and DFA. Spend 5 minutes to wrap up and consolidate the learnings
Evaluation	Reflective Questions: Can you provide an example of converting an NFA to a DFA, showing the transition function and accepting states? Spend 5 minutes evaluating student assimilation of the lesson contents

Lesson Plan No. 1.11	Course Name: Theory of Computation Topic Name: NFA with E-moves	Course No.: COM-604

Objectives	At the end of the lesson the student shall be able to: a. Understand the concept of Non deterministic finite automata b. Define the E transition on NFA c. Illustrate the elimination of E transition from NFA d. Articulate the E closure property e. Conversion from NFA to DFA
Teaching Aids (if any)	a. Chalk and talk b. Video of NPTEL c. PPT
Teaching Development	1. **Introduction** (5 minutes) - Explain the homework questions on NFA. - Introducing the concept of E transition for NFA. - Articulate the computation of NFA to DFA conversion. - Introduce the concept of NFA to DFA construction with https://www.youtube.com/watch?v=nyH7zyx7J5I&list=PL_Si_t2-nG7cMHfpLOZ88TgEPbXp489HZ&index=7 2. **Development** (30 minutes) a. Practise Problems - NFA with ∑ = {0, 1} accepts all strings with 01. Convert it into its equivalent DFA. - Design an NFA with ∑ = {0, 1} in which double '1' is followed by double '0'. Convert it into its equivalent DFA. - Design an NFA in which all the string contain a substring 1110. Convert it into its equivalent DFA. b. Equivalence Theorem - Illustrate the theorem for proving the equivalence of NFA to DFA. 3. **Exercise** (5 minutes) – Solve the following for both transition table and diagram : - Design an NFA with ∑ = {0, 1} accepts all string in which the third symbol from the right end is always 0.Convert it into its equivalent DFA

	- Design a NFA with ∑ = {0, 1} accepts the strings with an even number of 0's followed by single Convert it into its equivalent DFA.
Closure	1. Summarize the Lesson Learning Outcomes and get affirmation from students on these. 2. Suggested Reading - https://www.iitg.ac.in/dgoswami/Flat-Notes.pdf 3. Homework - Illustrate the theorem for proving the equivalence of NFA to DFA. Spend 5 minutes to wrap up and consolidate the learnings
Evaluation	1. Reflective Questions What is the importance E move NFA? Spend 5 minutes to evaluate student assimilation of the lesson contents

Lesson Plan No. 1.12	Course Name: Theory of Computation Topic Name: Regular expression designing	Course No.: COM-604

Objectives	At the end of the lesson, the student shall be able to: a. Understand the language states and transitions. b. Understand the basic concept of Finite Automata and regular expressions. c. Understand the different definitions of regular expressions. d. Illustrate the properties of regular expressions.
Teaching Aids (if any)	a. Chalk and talk b. Video of NPTEL c. PPT
Teaching Development	1. **Introduction** (5 minutes) - Ask questions on previous topic Which example can you think of automata in daily life? Give a formal definition of Finite Automata. Explain the concept of words, length of string. - Introducing the concept of transition diagram and transition states. - Articulate the computation of DFA. - Implement the transition diagrams for a few languages. - Introduce the concept of dfa construction with https://www.youtube.com/watch?v=NNSK4570Kho&list=PL_Si_t2-nG7cMHfpLOZ88TgEPbXp489HZ&index=3 2. **Development** (30 minutes) a. Practise Problems - DFA with ∑ = {0, 1} accepts all starting with 0. - DFA with ∑ = {0, 1} accepts all ending with 0. - Design a FA with ∑ = {0, 1} accepts those string which starts with 1 and ends with 0. - Introduce concept of Transition state, transition functions, initial state, final state, set of states. b. Transition Table - Illustrate the representation of transition table - Illustrate construction of transition tables for the given examples.

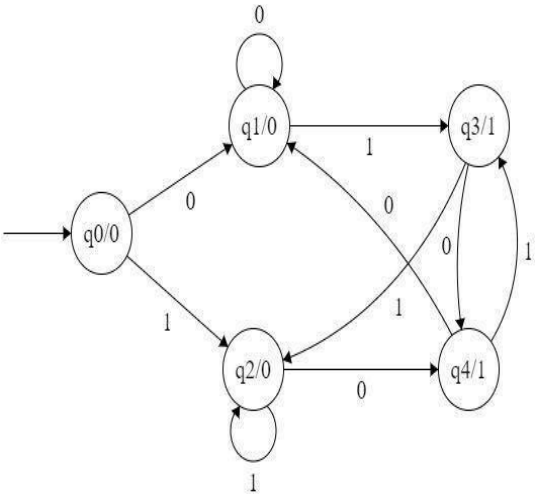
	3. **Exercise** (5 minutes) – Solve the following for both transition table and diagram : - IDesign a DFA L(M) = {w | w ε {0, 1}*} and W is a string that does not contain consecutive 1's.
Closure	1. Summarize the Lesson Learning Outcomes and get affirmation from students on these. 2. Suggested Reading - https://www.iitg.ac.in/dgoswami/Flat-Notes.pdf 3. Homework - Design a FA with ∑ = {0, 1} accepts the strings with an even number of 0's followed by single 1. Spend 5 minutes to wrap up and consolidate the learnings
Evaluation	1. Reflective Questions Why do we need regular expression? Spend 5 minutes to evaluate student assimilation of the lesson contents

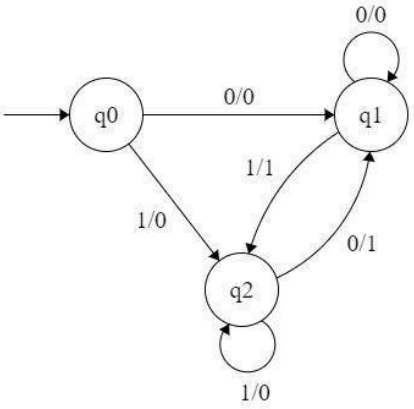
Lesson Plan No. 1.13	Course Name: Theory of Computation Topic Name: Finite machine with output assigned	Course No.: COM-604

Objectives	At the end of the lesson, the student shall be able to: a. Understand the components of an FMO, including states, transitions, inputs, outputs, and the initial state. b. Describe the behaviour of an FMO using transition diagrams and state transition tables. c. Design an FMO for a given problem, including identifying states, inputs, outputs, and transitions. d. Implement an FMO using appropriate programming constructs, such as switch-case statements or state transition functions. e. Analyze the functionality of an FMO by tracing its behaviour for a sequence of inputs.
Teaching Aids (if any)	a. Chalk and talk b. Video of NPTEL
Teaching Development	1. **Introduction** (5 minutes) - Ask questions What is a finite machine with output assigned? How does a finite machine with output differ from a regular finite machine? What are the components of a finite machine with output? How is the output of a finite machine determined? Can a finite machine with output assigned have multiple outputs for the same input? How are finite machines with output used in real-world applications? 2. **Development** (30 minutes) a. Finite Machines with Output Explain that in addition to states, finite machines can also have outputs associated with transitions between states. These outputs can represent actions taken by the machine or information produced. b. Components of Finite Machines with Output Describe the components of a finite machine with output, including states, inputs, transitions, outputs, and the initial state. c. Example Provide a simple example of a finite machine with output, such as a vending machine. Show how the machine transitions between states based on inputs (coins inserted) and produces outputs (dispensing products). d. Formal Definition Present the formal definition of a finite machine with output, including the tuple (Q, Σ, δ, q0, F, O), where Q is the set of states, Σ is the input alphabet, δ is the transition function, q0 is the initial state, F is the set of accepting states, and O is the

	output function. e. Transition Diagrams Introduce transition diagrams as a graphical representation of finite machines with output. Show how transitions are represented with arrows labelled with inputs and outputs. f. State Tables Discuss state tables as an alternative representation of finite machines with output, showing how states, inputs, outputs, and next states are organized in a tabular format. g. Applications: Highlight some applications of finite machines with output in real-world scenarios, such as control systems, communication protocols, and digital circuits.
Closure	1. Summarize the Lesson Learning Outcomes and get affirmation from students on these. 2. Suggested Reading - https://www.iitg.ac.in/dgoswami/Flat-Notes.pdf 3. Homework - Difference between finite automata and FAO Spend 5 minutes to wrap up and consolidate the learnings
Evaluation	1. Reflective Questions What is FAO? Spend 5 minutes evaluating student assimilation of the lesson contents

Lesson Plan No. 1.14	Course Name: Theory of Computation Topic Name: Moore and mealy machines	Course No.: COM-604

Objectives	At the end of the lesson, the student shall be able to: a. Understand the concept of converting Mealy to Moore machine b. Define the conversion process of Mealy to Moore Machine c. Illustrate the Mealy to Moore Machine conversion with the help of examples
Teaching Aids (if any)	a. Chalk and talk b. Video of NPTEL c. PPT
Teaching Development	1. **Introduction** (15 minutes) - Explain the homework questions on Moore Machine. - Introducing the steps of converting the Mealy to Moore Machine. - Articulate the Mealy to Moore Machine conversion with the help of Transition Diagram and Transition table. - Introduce the concept of Mealy to Moore Machine conversion with https://www.youtube.com/watch?v=O3If0Nr9to0 2. **Development** (30 minutes) a. Elaborating the steps. - Design the Mealy to Moore Machine conversion through the transition diagram and table.. b. Understanding with Example - Design a Mealy to Moore machine for the following:- 

	3. **Exercise** (5 minutes) – Solve the following Mealy to Moorey Machine for transition diagram and table :  - Use Google Classroom to collect responses and discuss the answers.
Closure	1. Summarize the Lesson Learning Outcomes and get affirmation from students on these. 2. Suggested solving more problems on Mealy Machine. - https://www.iitg.ac.in/dgoswami/Flat-Notes.pdf 3. Homework - Difference between mealy and moore machine Spend 5 minutes to wrap up and consolidate the learnings
Evaluation	1. Reflective Questions What are the various tuple that define moore machine? Spend 5 minutes to evaluate student assimilation of the lesson contents

Lesson Plan No.1.15	Course Name: Theory of Computation Topic Name: Conversion and Equivalence	Course No.: COM-604

Objectives	At the end of the lesson, the student shall be able to: a. Understand the basic concepts of Moore and Mealy machines, including their structures and functionalities. b. Convert a Moore machine to an equivalent Mealy machine and vice versa. c. Explore the differences between Moore and Mealy machines in terms of output behavior and state transitions. d. Understand the importance of equivalence in automata theory and its implications for solving computational problems. e. Apply the concepts learned to solve problems related to the conversion and equivalence of Moore and Mealy machines.
Teaching Aids (if any)	a. Chalk and talk b. Video of NPTEL c. Use of Nearpod tool for online quiz
Teaching Development	1. **Introduction** (5 minutes) - Ask questions What is the key difference between Moore and Mealy machines? How do you convert a Moore machine to an equivalent Mealy machine? How do you convert a Mealy machine to an equivalent Moore machine? Explain the concept of equivalence in the context of finite state machines. Can you describe a scenario where a Moore machine would be more suitable than a Mealy machine, and vice versa? How does the number of states in a Moore or Mealy machine affect its conversion and equivalence? 2. **Development** (30 minutes) a. Moore Machines Define Moore machines and their characteristics (states, outputs, transition function). Explain how Moore machines differ from other types of FSMs. b. Mealy Machines Define Mealy machines and their characteristics (states, outputs, transition function). Discuss the differences between Moore and Mealy machines, focusing on output behavior. c. Conversion from Moore to Mealy Explain the process of converting a Moore machine to a Mealy machine and Provide examples to illustrate the conversion steps. d. Conversion from Mealy to Moore Explain the process of converting a Mealy machine to a Moore machine and Provide examples to illustrate the conversion steps.

	e. Equivalence of Moore and Mealy Machines Define equivalence in the context of FSMs and explain how to determine if two machines are equivalent.
Closure	1. Summarize the Lesson Learning Outcomes and get affirmation from students on these. 2. Suggested Reading - https://www.iitg.ac.in/dgoswami/Flat-Notes.pdf 3. Homework - Write the steps to convert the mealy to moore machine? Spend 5 minutes to wrap up and consolidate the learnings
Evaluation	1. Reflective Questions Discuss the equivalence and conversion of moore to mealy and vice versa. Spend 5 minutes evaluating student assimilation of the lesson contents

Lesson Plan No. 1.16	Course Name: Theory of Computation Topic Name: Myhill-Nerode Theorem	Course No.: COM-604

Objectives	At the end of the lesson, the student shall be able to: a. Understand the concept of equivalence classes in the context of formal languages and automata theory. b. Learn the definition and significance of the Myhill-Nerode relation. c. Explore how the Myhill-Nerode theorem characterizes regular languages in terms of their equivalence classes. d. Apply the Myhill-Nerode theorem to determine whether a language is regular or not. e. Gain an appreciation for the theoretical foundations of regular languages and their equivalence to finite automata.
Teaching Aids (if any)	a. Chalk and talk b. Video of NPTEL
Teaching Development	1. **Introduction** (5 minutes) - Ask questions What is the Myhill-Nerode Theorem, and what does it state about the relationship between languages and automata? How is the Myhill-Nerode Theorem used to prove that a language is non-regular? Can you explain the concept of an equivalence relation in the context of the Myhill-Nerode Theorem? How do you determine the number of equivalence classes of strings for a given language using the Myhill-Nerode Theorem? Can you provide an example of applying the Myhill-Nerode Theorem to prove that a language is non-regular? 2. **Development** (30 minutes) a. Introduction to Equivalence Relations Explain equivalence relations and their relevance to string languages. b. Definition of DFA Review the components of a Deterministic Finite Automaton (DFA). c. Equivalence of DFAs Introduce the concept of DFA equivalence and how it relates to language equivalence. d. Indistinguishability of States Define indistinguishability in a DFA and its role in the Myhill-Nerode Theorem. e. Myhill-Nerode Theorem Present the theorem, stating that two states in a DFA are distinguishable if there exists a string that distinguishes them.

	f. Proof Overview Outline the proof, showing that the number of equivalence classes of strings is finite if and only if the language is regular. g. Application and Examples Provide examples of applying the theorem to determine language regularity.
Closure	1. Summarize the Lesson Learning Outcomes and get affirmation from students on these. 2. Suggested Reading - https://www.iitg.ac.in/dgoswami/Flat-Notes.pdf 3. Homework - Discuss the steps in Myhill- nerode theorem. Spend 5 minutes to wrap up and consolidate the learnings
Evaluation	1. Reflective Questions What is the significance of the Myhill-Nerode Theorem in the study of formal languages and automata theory? Spend 5 minutes evaluating student assimilation of the lesson contents

Lesson Plan No. 2.1	Course Name: Theory of Computation Topic Name: Context-free Grammar, Context-Free Languages	Course No.: COM-604

Objectives	At the end of the lesson, the student shall be able to: a. Define context-free grammar (CFG), context-free language (CFL), terminals, non-terminals, productions, and start symbols. b. Explain the concept of "context-free", how CFGs define language structure, and the relationship between CFGs and CFLs. c. Identify CFG components in examples, strings belonging to a CFG's language. d. Apply CFG rules to rewrite symbols and analyze code structure
Teaching Aids (if any)	a. Chalk and talk b. Video of NPTEL
Teaching Development	**Introduction** (5 minutes) - Ask questions What is context-free grammar (CFG), and how does it differ from a regular language? What are the key components of a CFG (terminals, non-terminals, productions, start symbol)? How do CFGs define the structure of a language? Can you explain the relationship between a CFG and the set of strings (language) it generates? How can we use CFG rules to rewrite non-terminal symbols and generate strings? Can you analyze a short code snippet or expression using a simple CFG to determine if it's valid syntax? 1. **Development** (30 minutes) a. Can every CFG be converted to an equivalent Chomsky Normal Form (CNF)? If so, how? b. What is the importance of CNF in parsing algorithms? c. How can a CFG be simplified by removing useless symbols, epsilon (ε) productions, and unit productions? d. How are production rules structured in a CFG? e. What are derivation trees and how are they used to represent CFG derivations? f. What is the difference between leftmost and rightmost derivations in CFGs?

Closure	1. Summarize the Lesson Learning Outcomes and get affirmation from students on these. 2. Suggested Reading - https://www.iitg.ac.in/dgoswami/Flat-Notes.pdf 3. Homework - What properties distinguish context-free languages from other types of languages? Spend 5 minutes to wrap up and consolidate the learnings
Evaluation	1. Reflective Questions: Define context free language. Spend 5 minutes evaluating student assimilation of the lesson contents

Lesson Plan No. 2.2	Course Name: Theory of Computation Topic Name: Reduced form of Grammar	Course No.: COM-604

Objectives	At the end of the lesson, the student shall be able to: a. Define what is meant by the "reduced form of grammar." b. Understand the importance of simplifying grammar in computational linguistics and automata theory. c. Identify and remove useless symbols (non-terminals that do not lead to terminal strings) from a given grammar. d. Detect and eliminate unnecessary productions that do not contribute to the derivation of terminal strings.
Teaching Aids (if any)	a. Chalk and talk b. Video of NPTEL
Teaching Development	1. **Introduction** (5 minutes) - Ask questions What is the reduced form of a grammar in the context of the theory of automata? How can we determine which symbols in grammar are useless? What are the steps to remove useless symbols from grammar in the context of automata theory? What is an epsilon (ε) production in a context-free grammar? Why might we want to remove epsilon productions in the theory of automata? How can we eliminate epsilon productions while ensuring the language generated by the grammar remains the same? 2. **Development** (30 minutes) a. Why is it important to have a reduced form of context-free grammar (CFG)? b. How can students identify useless symbols (non-terminals that never contribute to generating strings) within a CFG? c. How do Chomsky Normal Forms (CNF and GNF) further restrict CFGs and what are their benefits? d. How can students leverage reduced forms to analyze the complexity of a CFG and its generated language? e. Can students explore the relationship between reduced forms and the parsing process for CFGs?

Closure	1. Summarize the Lesson Learning Outcomes and get affirmation from students on these. 2. Suggested Reading - https://www.iitg.ac.in/dgoswami/Flat-Notes.pdf 3. Homework - How context free language differ from regular language? Spend 5 minutes to wrap up and consolidate the learnings
Evaluation	1. Reflective Questions Why is grammar reduction important in the theory of automata? Spend 5 minutes evaluating student assimilation of the lesson contents

Lesson Plan No. 2.3	Course Name: Theory of Computation Topic Name: Ambiguous and non-ambiguous grammar	Course No.: COM-604

Objectives	At the end of the lesson, the student shall be able to: a. Define grammar and its components: terminals, non-terminals, production rules, and start symbols. b. Explain the significance of grammar in defining the syntax of a language. c. Illustrate the differences between ambiguous and non-ambiguous grammar through practical examples. d. Teach strategies to convert ambiguous grammar to non-ambiguous grammar. e. Provide step-by-step examples of transforming ambiguous grammar.
Teaching Aids (if any)	a. Chalk and talk b. Video of NPTEL c. Use of Nearpod tool for online quiz
Teaching Development	1. **Introduction** (5 minutes) - Ask questions How can we determine if a given grammar is ambiguous? Explain with an example how the same string can have more than one parse tree in ambiguous grammar. What are the potential consequences of using ambiguous grammar in language processing? Provide an example of an ambiguous grammar and show how a string derived from it can have multiple parse trees. Provide an example of a non-ambiguous grammar and show how any string derived from it has a unique parse tree. 2. **Development** (30 minutes) a. How can you determine if a given grammar is ambiguous? b. What are some common methods or tools used to detect ambiguity in grammars? c. How do ambiguous and non-ambiguous grammars impact programming language design? d. Can you think of real-world scenarios where ambiguous grammars might cause issues? e. What are some advanced techniques for resolving ambiguity in complex grammars? f. How does ambiguity affect the efficiency and performance of parsers? g. Can you explain the concept of inherent ambiguity and provide examples of inherently ambiguous languages?

Closure	1. Summarize the Lesson Learning Outcomes and get affirmation from students on these. 2. Suggested Reading - https://www.iitg.ac.in/dgoswami/Flat-Notes.pdf 3. Homework - Can you describe a step-by-step process to identify ambiguity in a sample grammar? Spend 5 minutes to wrap up and consolidate the learnings
Evaluation	1. Reflective Questions Why the conversion of ambiguous grammar to un ambiguous grammar is required? Spend 5 minutes evaluating student assiilation of the lesson contents

Lesson Plan No. 2.4	Course Name: Theory of Computation Topic Name: Acceptors and generators	Course No.: COM-604

Objectives	At the end of the lesson, the student shall be able to: a. Define and differentiate between acceptors and generators in the context of automata theory. b. Identify the key components of acceptors and generators, such as states, transitions, input alphabet, acceptance criteria, and output generation rules. c. Explain how generators produce valid strings or sequences that belong to a language defined by a formal automaton. d. Apply their understanding by designing acceptors and generators for given formal languages, using both deterministic and non-deterministic automata where applicable.
Teaching Aids (if any)	a. Chalk and talk b. Video of NPTEL c. Use of Nearpod tool for online quiz
Teaching Development	1. **Introduction** (5 minutes) - Ask questions How are acceptors and generators related in terms of formal languages? Can an acceptor and a generator be based on the same automaton? Explain. What are some scenarios where understanding both acceptors and generators is crucial in automata theory? What are the key components of an acceptor (e.g., states, transitions, accepting states)? Can you provide examples of different types of automata that can function as acceptors? 2. **Development** (30 minutes) a. Is there a theoretical relationship between acceptors and generators for a particular language? b. How do the inputs and outputs differ between acceptors and generators? c. How do Mealy and Moore machines differ from traditional acceptors in terms of output generation? d. How do performance metrics such as time complexity and space complexity impact the design of acceptors and generators? e. Discuss optimization techniques for improving the efficiency of automata-based solutions.

Closure	1. Summarize the Lesson Learning Outcomes and get affirmation from students on these. 2. Suggested Reading - https://www.iitg.ac.in/dgoswami/Flat-Notes.pdf 3. Homework - Describe acceptor and generator in detail. Spend 5 minutes to wrap up and consolidate the learnings
Evaluation	1. Reflective Questions How does an acceptor determine whether a string belongs to a formal language? Spend 5 minutes evaluating student assimilation of the lesson contents

Lesson Plan No. 2.5	Course Name: Theory of Computation Topic Name: Relations between Classes of Languages	Course No.: COM-604

Objectives	At the end of the lesson, the student shall be able to: a. Define and differentiate between various classes of languages recognized by automata theory. b. Understand the closure properties of each language class under various operations (union, concatenation, Kleene star), and how these properties relate to the hierarchy of language classes. c. Identify the defining properties and characteristics of each language class, including their respective generating mechanisms (automata types) and accepted by (automata types). d. Explore the inclusion relationships among different classes of languages.
Teaching Aids (if any)	a. Chalk and talk b. Video of NPTEL c. PPT
Teaching Development	1. **Introduction** (5 minutes) - Ask questions What are the fundamental classes of languages in automata theory? How do these classes (regular, context-free, context-sensitive, recursively enumerable) relate to each other in terms of their expressive power? Why is understanding the hierarchy of language classes important in computer science and linguistics? What are closure properties of language classes? Provide examples. Which classes of languages are closed under union, intersection, concatenation, and Kleene star operations? Discuss the closure properties of regular, context-free, and context-sensitive languages. 2. **Development** (30 minutes) a. How do closure properties help in understanding the relationships between classes of languages? b. Provide examples of closure properties for regular languages, context-free languages, and context-sensitive languages. c. Provide examples of automata that recognize languages from each class (regular, context-free, context-sensitive, recursively enumerable). d. Discuss the complexity of automata required to recognize languages from each class and how it relates to the hierarchy. e. Can you think of real-world examples where understanding these relationships would be crucial for developing efficient algorithms or systems?

Closure	1. Summarize the Lesson Learning Outcomes and get affirmation from students on these. 2. Suggested Reading - https://www.iitg.ac.in/dgoswami/Flat-Notes.pdf 3. Homework - Define different Language? Spend 5 minutes to wrap up and consolidate the learnings
Evaluation	1. Reflective Questions What is relationship between different types of language? Spend 5 minutes evaluating student assimilation of the lesson contents

Lesson Plan No. 2.6	Course Name: Theory of Computation Topic Name: Pumping lemma of regular sets	Course No.: COM-604

Objectives	At the end of the lesson, the student shall be able to: a. Explain what the Pumping Lemma is and its significance in the theory of regular languages. b. Understand the properties of regular languages and how the Pumping Lemma is used to demonstrate these properties. c. Use the Pumping Lemma to prove that certain languages are not regular. d. Create their own counterexamples to demonstrate the non-regularity of specific languages using the Pumping Lemma.
Teaching Aids (if any)	a. Chalk and talk b. Video of NPTEL
Teaching Development	1. **Introduction** (5 minutes) - Ask questions What are regular sets, and why are they important in the study of formal languages and automata? What is the Pumping Lemma for regular sets? Why is the Pumping Lemma an important tool in the theory of computation? Can you state the Pumping Lemma for regular languages formally? What does it mean for a string to be "pumped" in the context of the Pumping Lemma? How does the Pumping Lemma help in proving that a language is not regular? 2. **Development** (30 minutes) a. Provide a step-by-step example of applying the Pumping Lemma to demonstrate that a specific language is not regular. b. What are some common mistakes or misconceptions we might encounter when applying the Pumping Lemma? c. How does the Pumping Lemma relate to closure properties of regular languages? d. Discuss the role of the Pumping Lemma in the design and analysis of regular expressions and finite automata.

Closure	1. Summarize the Lesson Learning Outcomes and get affirmation from students on these. 2. Suggested Reading - https://www.iitg.ac.in/dgoswami/Flat-Notes.pdf 3. Homework - How can you use the Pumping Lemma to show that a language is not regular? Spend 5 minutes to wrap up and consolidate the learnings
Evaluation	1. Reflective Questions Why do use pumping lemma theorem? Spend 5 minutes evaluating student assimilation of the lesson contents

Lesson Plan No. 2.7	Course Name: Theory of Computation Topic Name: Chomsky's hierarchy of languages	Course No.: COM-604

Objectives	At the end of the lesson, the student shall be able to: a. Define and describe the four levels of Chomsky's hierarchy: regular languages, context-free languages, context-sensitive languages, and recursively enumerable languages. b. Categorize different formal languages according to Chomsky's hierarchy. c. Identify the type of automaton associated with each language class in Chomsky's hierarchy. d. Explain the characteristics and limitations of each language class in Chomsky's hierarchy.
Teaching Aids (if any)	a. Chalk and talk b. Video of NPTEL
Teaching Development	1. **Introduction** (5 minutes) - Ask questions Who is Noam Chomsky, and what is Chomsky's Hierarchy of Languages? Why is Chomsky's Hierarchy important in the study of formal languages and automata theory? Which type of automaton recognizes context-free languages? What are the defining features of pushdown automata (PDA)? Provide an example of a context-free grammar and explain how it generates a context-free language. 2. **Development** (30 minutes) a. What distinguishes recursively enumerable languages from context-sensitive languages? b. Can you provide examples of problems that can be solved using recursively enumerable languages but not context-sensitive languages? c. Can you provide examples of real-world applications for each type of language in Chomsky's hierarchy? d. How do the different types of languages in Chomsky's hierarchy relate to each other? e. Can you describe the inclusiveness of Chomsky's hierarchy (i.e., how one type of language is included in another)? f. Why are all regular languages also context-free, but not all context-free languages are regular?

Closure	1. Summarize the Lesson Learning Outcomes and get affirmation from students on these. 2. Suggested Reading - https://www.iitg.ac.in/dgoswami/Flat-Notes.pdf 3. Homework - How are the different types of languages used in the design of programming languages and compilers? Spend 5 minutes to wrap up and consolidate the learnings
Evaluation	1. Reflective Questions What is relation between different types of language ? Spend 5 minutes evaluating student assimilation of the lesson contents

Lesson Plan No. 2.8	Course Name: Theory of Computation Topic Name: Derivation Trees	Course No.: COM-604

Objectives	At the end of the lesson, the student shall be able to: a. Define what a derivation tree (also known as a parse tree) is in the context of formal languages and automata. b. Identify the key components of a derivation tree, including nodes, and edges. c. Demonstrate how to construct a derivation tree for a given string using a specified context-free grammar (CFG). d. Identify and analyze valid derivations of strings from a given grammar using derivation trees.
Teaching Aids (if any)	a. Chalk and talk b. Video of NPTEL c. Use of Nearpod tool for online quiz
Teaching Development	1. **Introduction** (5 minutes) - Ask questions What is a derivation tree (also known as a parse tree) in the context of formal languages and automata? What are the components of a derivation tree (nodes, branches, root, leaves)? How do you construct a derivation tree for a given string using a context-free grammar? What is the significance of the root and leaf nodes in a derivation tree? How do the internal nodes of a derivation tree relate to the production rules of the CFG? 2. **Development** (30 minutes) a. How are derivation trees used in compiler design and construction? b. What is the relationship between derivation trees and syntax-directed translation? c. How do leftmost and rightmost derivations relate to derivation trees? d. Can you explain the concept of derivation sequences and their connection to derivation trees? e. What is the role of derivation trees in the context of deterministic and non-deterministic parsing techniques?

Closure	1. Summarize the Lesson Learning Outcomes and get affirmation from students on these. 2. Suggested Reading - https://www.iitg.ac.in/dgoswami/Flat-Notes.pdf 3. Homework How does a derivation tree represent the structure of a string derived from a context-free grammar (CFG)? Spend 5 minutes to wrap up and consolidate the learnings
Evaluation	1. Reflective Questions What are the properties of derivation trees that make them useful in parsing and syntax analysis? Spend 5 minutes evaluating student assimilation of the lesson contents

Lesson Plan No. 2.9	Course Name: Theory of Computation Topic Name: CYK Algorithm for CFL Membership	Course No.: COM-604

Objectives	At the end of the lesson, the student shall be able to: a. Explain the significance of the CYK algorithm in computational theory and automata. b. Identify the key components of the CYK algorithm, including the parse table, production rules, and the input string. c. Describe how the CYK algorithm uses a dynamic programming approach to determine membership in a context-free language. d. Explain the importance of CNF and how it simplifies the application of the CYK algorithm.
Teaching Aids (if any)	a. Chalk and talk b. Video of NPTEL c. Use of Nearpod tool for online quiz
Teaching Development	1. **Introduction** (5 minutes) - Ask questions Can you describe the basic steps involved in the CYK algorithm? How is the CYK algorithm used to determine whether a string belongs to a given context-free language? What is the role of the parse table in the CYK algorithm? Can you explain the process of filling in the parse table with production rules from the context-free grammar? How does the CYK algorithm ensure that all possible substrings of the input string are considered? 2. **Development** (30 minutes) a. How do you construct a CYK table for a given context-free grammar and input string? b. Can you walk through an example of applying the CYK algorithm to a simple context-free grammar and string? c. What are the challenges or limitations associated with using the CYK algorithm? d. How can you implement the CYK algorithm in a programming language of your choice? e. Can you provide a sample implementation of the CYK algorithm and explain how it works?

Closure	1. Summarize the Lesson Learning Outcomes and get affirmation from students on these. 2. Suggested Reading - https://www.iitg.ac.in/dgoswami/Flat-Notes.pdf 3. Homework How does the CYK algorithm handle the parsing of strings based on context-free grammar? Spend 5 minutes to wrap up and consolidate the learnings
Evaluation	1. Reflective Questions What are some common pitfalls or errors to watch out for when coding the CYK algorithm? Spend 5 minutes evaluating student assimilation of the lesson contents

Lesson Plan No. 2.10	Course Name: Theory of Computation Topic Name: Testing emptiness of CFLs	Course No.: COM-604

Objectives	At the end of the lesson, the student shall be able to: a. Define what a context-free language is and provide examples. b. Describe the characteristics and properties of context-free grammars and context-free languages. c. Explain how to use context-free grammars to test the emptiness of CFLs. d. Discuss the implications of a language being empty in the context of formal languages and automata.
Teaching Aids (if any)	a. Chalk and talk b. Video of NPTEL c. Use of Nearpod tool for online quiz
Teaching Development	1. **Introduction** (5 minutes) - Ask questions What does it mean for a CFL to be "empty"? How can we formally define the concept of emptiness in the context of CFLs? Why is testing for the emptiness of a CFL important in formal language theory? What are the common methods for testing the emptiness of a CFL? How can you use context-free grammar to determine if the corresponding language is empty? What role do parse trees play in testing the emptiness of a CFL? 2. **Development** (30 minutes) a. Given a CFG, how would you test whether the language it generates is empty? Provide an example. b. How can you apply the emptiness test algorithm to different types of grammars and CFLs? c. How does the process differ for testing the emptiness of CFLs compared to regular languages? d. Can you discuss how the concept of emptiness in CFLs is used in more advanced topics like parsing and compiler design?

Closure	1. Summarize the Lesson Learning Outcomes and get affirmation from students on these. 2. Suggested Reading - https://www.iitg.ac.in/dgoswami/Flat-Notes.pdf 3. Homework - How does the concept of language inclusion relate to testing emptiness in CFLs? Spend 5 minutes to wrap up and consolidate the learnings
Evaluation	1. Reflective Questions • What challenges might arise when testing for the emptiness of CFLs with complex grammar? Spend 5 minutes evaluating student assimilation of the lesson contents

Lesson Plan No. 3.1	Course Name: Theory of Computation Topic Name: Church Testing Hypothesis, Turing Computability	Course No.: COM-604

Objectives	At the end of the lesson, the student shall be able to: a. Explain the Church-Turing Hypothesis and its significance in the theory of computation. b. Illustrate how Turing machines serve as a model for computation and their role in defining computable functions. c. Define Turing computability and explain the concept of a Turing-computable function. d. Analyze the differences between Turing-computable functions and those that can be computed by other models of computation.
Teaching Aids (if any)	a. Chalk and talk b. Video of NPTEL
Teaching Development	1. **Introduction** (5 minutes) - Ask questions What is the Church-Turing Hypothesis, and who proposed it? How does the Church-Turing Hypothesis relate to the concept of computability? What is the significance of the Church-Turing Hypothesis in the theory of computation? How does the Church-Turing Hypothesis influence our understanding of what can be computed? What are Turing machines, and how do they formalize the concept of computability? Can you describe the key components and functioning of a Turing machine? 2. **Development** (30 minutes) a. How does Turing computability impact real-world computing and programming languages? b. What are some practical examples of problems that are known to be computable by Turing machines? c. How do modern computational models (e.g., quantum computers) relate to the Church-Turing Hypothesis? d. What are the different types of Turing machines (e.g., deterministic, nondeterministic, multi-tape), and how do they compare in terms of computational power? e. How can you simulate a Turing machine using a different computational model?

Closure	1. Summarize the Lesson Learning Outcomes and get affirmation from students on these. 2. Suggested Reading - https://www.iitg.ac.in/dgoswami/Flat-Notes.pdf 3. Homework What is the significance of the Church-Turing Hypothesis in the theory of computation? Spend 5 minutes to wrap up and consolidate the learnings
Evaluation	1. Reflective Questions How does the Church-Turing Hypothesis relate to the Halting Problem and its implications for computability? Spend 5 minutes evaluating student assimilation of the lesson contents

Lesson Plan No. 3.2	Course Name: Theory of Computation Topic Name: Non- deterministic, Multitape and other versions of Turing machines	Course No.: COM-604

Objectives	At the end of the lesson, the student shall be able to: a. Define what a non-deterministic Turing machine is and explain how it differs from a deterministic Turing machine. b. Illustrate how non-determinism in Turing machines allows multiple computational paths and the concept of accepting states in such machines. c. Explain the operation of multitape Turing machines, including how they use multiple tapes and tape heads to perform computations. d. Analyze the computational power of multitape Turing machines and show how they can simulate single-tape Turing machines efficiently.
Teaching Aids (if any)	a. Chalk and talk b. Video of NPTEL c. PPT
Teaching Development	1. **Introduction** (5 minutes) - Ask questions What does it mean for a Turing machine to be non-deterministic? How does a non-deterministic Turing machine differ from a deterministic Turing machine? Can you provide an example of a problem that is more easily solved by a non-deterministic Turing machine than by a deterministic one? How does the concept of non-determinism affect the complexity class of problems solvable by a Turing machine? What is a multi-tape Turing machine, and how does it differ from a single-tape Turing machine? What advantages do multi-tape Turing machines offer over single-tape Turing machines? 2. **Development** (30 minutes) a. How can non-deterministic and multitape Turing machines be used to prove theoretical results in computer science? b. What are some advanced techniques for analyzing the performance and capabilities of different types of Turing machines? c. How can you simulate a multi-tape Turing machine with a single-tape Turing machine?

Closure	1. Summarize the Lesson Learning Outcomes and get affirmation from students on these. 2. Suggested Reading - https://www.iitg.ac.in/dgoswami/Flat-Notes.pdf 3. Homework Can you explain how Turing machines with additional computational resources (e.g., infinite tapes) affect their computational power? Spend 5 minutes to wrap up and consolidate the learnings
Evaluation	1. Reflective Questions What is the computational capability of Turing machine? Spend 5 minutes evaluating student assimilation of the lesson contents

Lesson Plan No. 3	Course Name: Theory of Computation Topic Name: Churches Hypothesis	Course No.: COM-604

Objectives	At the end of the lesson, the student shall be able to: a. Explain Church's Hypothesis and its significance in the context of computability and formal languages. b. Identify the different computational models (e.g., lambda calculus, Turing machines) that are used to express Church's Hypothesis. c. Discuss how Church's Hypothesis relates to the concept of decidability and computability. d. Solve problems that involve demonstrating the equivalence of computational models based on Church's Hypothesis.
Teaching Aids (if any)	a. Chalk and talk b. Video of NPTEL c. Use of Nearpod tool for online quiz
Teaching Development	1. **Introduction** (5 minutes) - Ask questions What is Church's Hypothesis (Church-Turing Thesis) in the context of computability theory? Who are the key figures associated with the Church's Hypothesis and what were their contributions? How does Church's Hypothesis relate to the concept of algorithmic computability? What were the main computational models proposed around the time Church's Hypothesis was formulated? How did Church's Hypothesis influence the development of modern computer science and automata theory? What were the key motivations behind proposing the Church's Hypothesis? 2. **Development** (30 minutes) a. How has Church's Hypothesis influenced the development of modern computing and programming languages? b. What role does Church's Hypothesis play in the study of complexity theory and algorithm design? c. How does Church's Hypothesis impact the classification of problems as decidable or undecidable? d. How does Church's Hypothesis relate to finite automata, pushdown automata, and Turing machines? e. How can Church's Hypothesis be used to argue about the equivalence of different computational models?

Closure	1. Summarize the Lesson Learning Outcomes and get affirmation from students on these. 2. Suggested Reading - https://www.iitg.ac.in/dgoswami/Flat-Notes.pdf 3. Homework Define **Churches Hypothesis.**
Evaluation	1. Reflective Questions What are the implications of Church's Hypothesis for the limits of computation in different types of automata? Spend 5 minutes evaluating student assimilation of the lesson contents

Lesson Plan No. 3.4	Course Name: Theory of Computation Topic Name: Primitive Recursive functions	Course No.: COM-604

Objectives	At the end of the lesson, the student shall be able to: a. Define what primitive recursive functions are and how they differ from general recursive functions. b. Explain the importance of primitive recursive functions in the context of automata theory and computability. c. Understand how these basic functions serve as building blocks for more complex primitive recursive functions. d. Analyze the properties and limitations of primitive recursive functions in terms of their computational power.
Teaching Aids (if any)	a. Chalk and talk b. Video of NPTEL
Teaching Development	1. **Introduction** (5 minutes) - Ask questions What are the basic building blocks of Primitive Recursive Functions (e.g., zero function, successor function, projection function)? How are Primitive Recursive Functions constructed using composition and primitive recursion? What is the role of the base functions in defining Primitive Recursive Functions? Can you provide examples of simple Primitive Recursive Functions and explain their construction? How can you determine if a given function is Primitive Recursive? What are the common operations and transformations used in Primitive Recursive Functions? 2. **Development** (30 minutes) a. How do Primitive Recursive Functions relate to the theory of automata? b. Can you give examples of how Primitive Recursive Functions can be used to describe certain types of automata? c. In what ways do Primitive Recursive Functions help in understanding the computational limits of automata? d. What are the computational limitations of Primitive Recursive Functions compared to general recursive functions? e. How does the complexity of Primitive Recursive Functions compare to other classes of functions in formal language theory?

Closure	1. Summarize the Lesson Learning Outcomes and get affirmation from students on these. 2. Suggested Reading - https://www.iitg.ac.in/dgoswami/Flat-Notes.pdf 3. Homework - Solve the quiz questions on computational devices and model concepts and submit them on Google Classroom Spend 5 minutes to wrap up and consolidate the learnings
Evaluation	1. Reflective Questions (What, why, Who?). Allow students to answer and discuss. 2. Nearpod/ Google form Quiz Spend 5 minutes evaluating student assimilation of the lesson contents

Lesson Plan No. 3.6	Course Name: Theory of Computation Topic Name: Universal Turing machines	Course No.: COM-604

Objectives	At the end of the lesson, the student shall be able to: a. Understand what constitutes a Primitive Universal Turing Machine. b. Identify the key characteristics and components of a Primitive Universal Turing Machine. c. Describe the historical development and significance of Primitive Universal Turing Machines. d. Explain the concept of universality in the context of Turing machines.
Teaching Aids (if any)	a. Chalk and talk b. Video of NPTEL c. Use of Nearpod tool for online quiz
Teaching Development	1. **Introduction** (5 minutes) - Ask questions What is a Turing machine, and why is it important in the study of computation and automata theory? What are the basic components of a Turing machine (e.g., tape, head, states, transition function)? How does a Turing machine simulate a computation process? What is a Primitive Universal Turing Machine (PUTM)? How does a Primitive Universal Turing Machine differ from a standard Turing machine? Why are Primitive Universal Turing Machines considered "primitive" compared to more modern models of universal Turing machines? 2. **Development** (30 minutes) a. Provide an example of a simple computation or algorithm that can be executed by a Primitive Universal Turing Machine. b. What are the theoretical implications of having a Primitive Universal Turing Machine in terms of computational power? c. Explain why a Primitive Universal Turing Machine is still a relevant concept in the study of theoretical computer science. d. Discuss how to encode a given problem's input and output using a PUTM. e. What are the strengths and limitations of a PUTM compared to more advanced models of universal Turing machines?

Closure	1. Summarize the Lesson Learning Outcomes and get affirmation from students on these. 2. Suggested Reading - https://www.iitg.ac.in/dgoswami/Flat-Notes.pdf 3. Homework - Solve the quiz questions on computational devices and model concepts and submit them on Google Classroom Spend 5 minutes to wrap up and consolidate the learnings
Evaluation	1. Reflective Questions (What, why, Who?). Allow students to answer and discuss. 2. Nearpod/ Google form Quiz Spend 5 minutes evaluating student assimilation of the lesson contents

Lesson Plan No. 3.8	Course Name: Theory of Computation Topic Name: Decidability	Course No.: COM-604

Objectives	At the end of the lesson, the student shall be able to: a. Understand and articulate the concept of decidability in the context of formal languages and automata theory. b. Explain the criteria that make a language decidable. c. Analyze various problems and determine whether they are decidable or undecidable. d. Apply concepts from automata theory to assess the decidability of problems related to languages and grammars.
Teaching Aids (if any)	a. Chalk and talk b. Video of NPTEL c. Use of Nearpod tool for online quiz
Teaching Development	1. **Introduction** (5 minutes) - Ask questions What does "decidability" mean in the context of automata and formal languages? What is a decidable language? Can you provide examples of decidable languages? How do we determine if a problem is decidable or not? What role do algorithms play in establishing the decidability of a problem? What are some common techniques used to prove that a problem is decidable? How can reduction techniques be used to demonstrate decidability? Can you explain the concept of a decision procedure and how it relates to decidability? 2. **Development** (30 minutes) a. How does the decidability of problems impact practical applications such as compiler design or software verification? b. What are the implications of undecidability for the design of algorithms and software systems? c. How do decidability results influence the complexity of computing tasks? d. What is the difference between decidable and semi-decidable languages? e. How do recursive and recursively enumerable languages relate to decidability? f. What are some advanced results or theorems related to decidability in automata theory?

Closure	1. Summarize the Lesson Learning Outcomes and get affirmation from students on these. 2. Suggested Reading - https://www.iitg.ac.in/dgoswami/Flat-Notes.pdf 3. Homework - Solve the quiz questions on computational devices and model concepts and submit them on Google Classroom Spend 5 minutes to wrap up and consolidate the learnings
Evaluation	1. Reflective Questions (What, why, Who?). Allow students to answer and discuss. 2. Nearpod/ Google form Quiz Spend 5 minutes evaluating student assimilation of the lesson contents

Lesson Plan No. 3.8	Course Name: Theory of Computation Topic Name: Halting problem	Course No.: COM-604

Objectives	At the end of the lesson, the student shall be able to: a. Understand the concept of the Halting Problem as a decision problem in computability theory. b. Explain what it means for a program to "halt" and the significance of determining whether a program halts or runs indefinitely. c. Explain why the Halting Problem is undecidable using concepts from Turing machines. d. Understand how the undecidability of the Halting Problem affects the feasibility of creating general-purpose program analysis tools.
Teaching Aids (if any)	a. Chalk and talk b. Video of NPTEL c. Use of Nearpod tool for online quiz
Teaching Development	1. **Introduction** (5 minutes) - Ask questions Can you define what it means for a problem to be "decidable"? What does it mean for a problem to be "undecidable"? How can you formalize the Halting Problem in terms of a Turing machine and its input? Provide an example of a Turing machine and a corresponding input where the Halting Problem would be relevant. What is the concept of reduction in the context of proving undecidability? How does the proof of the Halting Problem's undecidability use a reduction from the Halting Problem itself? 2. **Development** (30 minutes) a. How does the concept of the Halting Problem fit into the broader theory of computability and complexity? b. What is the difference between the Halting Problem and the Halting Problem for specific restricted classes of Turing machines? c. How does the concept of the Halting Problem extend to other computational models, such as pushdown automata or finite automata? d. What are some variations or extensions of the Halting Problem, and how do they differ from the original problem? e. What are some practical implications of the Halting Problem in software development and debugging?

Closure	1. Summarize the Lesson Learning Outcomes and get affirmation from students on these. 2. Suggested Reading - https://www.iitg.ac.in/dgoswami/Flat-Notes.pdf 3. Homework - Solve the quiz questions on computational devices and model concepts and submit them on Google Classroom Spend 5 minutes to wrap up and consolidate the learnings
Evaluation	1. Reflective Questions (What, why, Who?). Allow students to answer and discuss. 2. Nearpod/ Google form Quiz Spend 5 minutes evaluating student assimilation of the lesson contents

Lesson Plan No. 3..9	Course Name: Theory of Computation Topic Name: Stack Automata	Course No.: COM-604

Objectives	At the end of the lesson, the student shall be able to: a. Define what a stack automaton is and explain its components. b. Identify the differences between a stack automaton and other types of automata (e.g., finite automata, pushdown automata). c. Construct a stack automaton for a given language or set of strings. d. Interpret the configuration of a stack automaton at various stages of computation. e. Apply stack automata to solve problems related to language recognition and parsing.
Teaching Aids (if any)	a. Chalk and talk b. Video of NPTEL c. Use of Nearpod tool for online quiz
Teaching Development	1. **Introduction** (5 minutes) - Ask questions What is a stack automaton, and how does it differ from a finite automaton? What role does the stack play in the operation of a stack automaton? What operations can be performed on the stack in a stack automaton (e.g., push, pop, top)? How do these stack operations affect the state transitions of the automaton? Can you provide an example of how stack operations are used to process a given input string? 2. **Development** (30 minutes) a. What are some practical applications of stack automata in computer science and formal language theory? b. How are stack automata used in the design and implementation of compilers or parsers? c. Can you discuss real-world scenarios where stack automata provide solutions to complex problems? d. How do stack automata compare to other types of automata, such as finite automata and Turing machines? e. What are the advantages and limitations of using stack automata for recognizing certain languages? f. How can stack automata be used to analyze and solve problems involving nested structures, such as balanced parentheses?

Closure	1. Summarize the Lesson Learning Outcomes and get affirmation from students on these. 2. Suggested Reading - https://www.iitg.ac.in/dgoswami/Flat-Notes.pdf 3. Homework - Solve the quiz questions on computational devices and model concepts and submit them on Google Classroom Spend 5 minutes to wrap up and consolidate the learnings
Evaluation	1. Reflective Questions (What, why, Who?). Allow students to answer and discuss. 2. Nearpod/ Google form Quiz Spend 5 minutes evaluating student assimilation of the lesson contents

Lesson Plan No. 4.1	Course Name: Theory of Computation Topic Name: Definition of Push Down Automata	Course No.: COM-604

Objectives	At the end of the lesson, the student shall be able to: a. Understand and articulate the definition of a Pushdown Automaton (PDA). b. Describe the key components of a PDA, including states, input tape, stack, and transition functions. c. Identify and explain the roles of the stack, input tape, and transition functions in a PDA. d. Understand the significance of the stack in providing additional memory beyond finite automata.
Teaching Aids (if any)	a. Chalk and talk b. Video of NPTEL c. Use of Nearpod tool for online quiz
Teaching Development	1. **Introduction** (5 minutes) - Ask questions What is a Pushdown Automaton (PDA) and how does it differ from a Finite Automaton? What are the components of a Pushdown Automaton? How does the addition of a stack enhance the computational power of a PDA compared to a finite automaton? What are the key components of a PDA, and what role does each component play? How does the stack in a PDA function, and what operations can be performed on it? What are the possible transitions of a PDA, and how are they determined? How does a PDA handle input symbols and stack operations during computation? 2. **Development** (30 minutes) a. How does a PDA process an input string? b. What are the different types of transitions in a PDA (e.g., transition functions with stack operations)? c. How does the PDA's stack influence the acceptance of an input string? d. What are the different criteria for accepting an input string in a PDA (e.g., acceptance by empty stack, acceptance by final state)? e. How does the choice of acceptance criteria affect the behavior of a PDA? f. Can you provide examples of languages accepted by a PDA using each type of acceptance criterion?

Closure	1. Summarize the Lesson Learning Outcomes and get affirmation from students on these. 2. Suggested Reading - https://www.iitg.ac.in/dgoswami/Flat-Notes.pdf 3. Homework - Solve the quiz questions on computational devices and model concepts and submit them on Google Classroom Spend 5 minutes to wrap up and consolidate the learnings
Evaluation	1. Reflective Questions (What, why, Who?). Allow students to answer and discuss. 2. Nearpod/ Google form Quiz Spend 5 minutes evaluating student assimilation of the lesson contents

Lesson Plan No. 4.2	Course Name: Theory of Computation Topic Name: Model of push down automata	Course No.: COM-604

Objectives	At the end of the lesson, the student shall be able to: a. Explain what a Pushdown Automaton is, including its components and how it extends the capabilities of a finite automaton with a stack. b. Construct a PDA that accepts simple context-free languages, such as those with balanced parentheses or palindromes. c. Design PDAs for specific context-free languages and describe the transition functions and stack operations required. d. Understand how to convert a given context-free grammar into an equivalent PDA.
Teaching Aids (if any)	a. Chalk and talk b. Video of NPTEL c. Use of Nearpod tool for online quiz
Teaching Development	1. **Introduction** (5 minutes) - Ask questions What is a Pushdown Automaton (PDA), and how does it differ from a finite automaton? What are the components of a PDA (e.g., states, input alphabet, stack alphabet, transition function, initial state, and accepting states)? How is the stack in a PDA different from the tape in a Turing machine? How does a PDA use its stack to process input strings? What are the roles of the push and pop operations in the stack of a PDA? How does a PDA transition between states based on the current input symbol and the top of the stack? 2. **Development** (30 minutes) a. What is the difference between deterministic and non-deterministic PDAs? b. How do deterministic PDAs differ in terms of their transition functions compared to non-deterministic PDAs? c. Can you provide an example of a language that can be recognized by a non-deterministic PDA but not by a deterministic PDA? d. How is a PDA related to context-free grammars? What is the connection between them? e. How can you use a PDA to parse strings generated by context-free grammar? f. What is the relationship between the parse tree of a context-free grammar and the computation of a PDA?

Closure	1. Summarize the Lesson Learning Outcomes and get affirmation from students on these. 2. Suggested Reading - https://www.iitg.ac.in/dgoswami/Flat-Notes.pdf 3. Homework - Solve the quiz questions on computational devices and model concepts and submit them on Google Classroom Spend 5 minutes to wrap up and consolidate the learnings
Evaluation	1. Reflective Questions (What, why, Who?). Allow students to answer and discuss. 2. Nearpod/ Google form Quiz Spend 5 minutes evaluating student assimilation of the lesson contents

Lesson Plan No. 4.3	Course Name: Theory of Computation Topic Name: Acceptance of CFL	Course No.: COM-604

Objectives	At the end of the lesson, the student shall be able to: a. Define and describe the characteristics of context-free languages. b. Identify the components of a PDA, including the stack, input tape, states, and transitions. c. Explain the relationship between context-free grammars (CFG) and pushdown automata. d. Construct a PDA for a given context-free language. e. Convert context-free grammars to equivalent pushdown automata.
Teaching Aids (if any)	a. Chalk and talk b. Video of NPTEL
Teaching Development	1. **Introduction** (5 minutes) - Ask questions What is a Context-Free Language (CFL)? How do Context-Free Grammars (CFGs) generate CFLs? Can you provide an example of a PDA that recognizes a simple CFL, such as $\{ a^n b^n \mid n \geq 0\}$? Why are non-deterministic PDAs more powerful than deterministic PDAs in recognizing CFLs? Can you give an example of a CFL that can be recognized by a non-deterministic PDA but not by a deterministic PDA? 2. **Development** (30 minutes) a. What is the role of the stack in the acceptance of CFLs by PDAs? b. What is the significance of the conversion from CFG to PDA in understanding CFL acceptance? c. Can you illustrate the conversion process with an example CFG and its corresponding PDA? d. What are the limitations of PDAs in recognizing languages beyond CFLs? e. What are the challenges in designing PDAs for more complex CFLs? f. How does the concept of ambiguity in CFGs affect the design of PDAs? g. Can you explain the significance of the pumping lemma for CFLs in the context of PDAs?

Closure	1. Summarize the Lesson Learning Outcomes and get affirmation from students on these. 2. Suggested Reading - https://www.iitg.ac.in/dgoswami/Flat-Notes.pdf 3. Homework - Solve the quiz questions on computational devices and model concepts and submit them on Google Classroom Spend 5 minutes to wrap up and consolidate the learnings
Evaluation	1. Reflective Questions (What, why, Who?). Allow students to answer and discuss. 2. Nearpod/ Google form Quiz Spend 5 minutes evaluating student assimilation of the lesson contents

Lesson Plan No. 4.4	Course Name: Theory of Computation Topic Name: Acceptance by Final State and Acceptance by Empty stack and its Equivalence	Course No.: COM-604

Objectives	At the end of the lesson, the student shall be able to: a. Explain the equivalence between acceptance by final state and acceptance by empty stack in the context of PDAs. b. Construct PDAs for given languages using both acceptance by final state and acceptance by empty stack. c. Analyze the behavior of PDAs to determine whether they accept input strings by final state or empty stack. d. Design equivalent PDAs for a given language using both acceptance methods.
Teaching Aids (if any)	a. Chalk and talk b. Video of NPTEL c. Use of Nearpod tool for online quiz
Teaching Development	1. **Introduction** (5 minutes) - Ask questions What does "acceptance by final state" mean in the context of finite automata? How do you determine if a string is accepted by an automaton based on its final state? How does "acceptance by empty stack" work in the context of pushdown automata? Can you explain the process of acceptance by empty stack with an example of a pushdown automaton? What does it mean for two acceptance criteria to be equivalent in the context of automata theory? How can acceptance by final state and acceptance by empty stack be shown to be equivalent? 2. **Development** (30 minutes) a. In what scenarios might you prefer to use acceptance by final state over acceptance by empty stack, or vice versa? b. How does the choice of acceptance criteria affect the design and implementation of an automaton? c. Can you discuss any real-world applications where acceptance by final state or acceptance by empty stack is particularly useful? d. What are some of the challenges associated with proving the equivalence of different acceptance criteria in more complex automata? e. How do different types of automata (e.g., deterministic vs. non-deterministic) handle acceptance by final state and empty stack?

Closure	1. Summarize the Lesson Learning Outcomes and get affirmation from students on these. 2. Suggested Reading - https://www.iitg.ac.in/dgoswami/Flat-Notes.pdf 3. Homework - Solve the quiz questions on computational devices and model concepts and submit them on Google Classroom Spend 5 minutes to wrap up and consolidate the learnings
Evaluation	1. Reflective Questions (What, why, Who?). Allow students to answer and discuss. 2. Nearpod/ Google form Quiz Spend 5 minutes evaluating student assimilation of the lesson contents

Lesson Plan No. 4.5	Course Name: Theory of Computation Topic Name: Equivalence of CFG	Course No.: COM-604

Objectives	At the end of the lesson, the student shall be able to: a. Explain the components of a CFG, including terminals, non-terminals, production rules, and the start symbol. b. Explain the concept of equivalence between two CFGs. c. Describe methods to determine if two CFGs are equivalent. d. Explain techniques for transforming CFGs to show their equivalence, such as converting to Chomsky Normal Form (CNF) or Greibach Normal Form (GNF).
Teaching Aids (if any)	a. Chalk and talk b. Video of NPTEL c. Use of Nearpod tool for online quiz
Teaching Development	1. **Introduction** (5 minutes) - Ask questions What are the steps to convert a CFG to an equivalent PDA? Can you provide an example of a CFG and show its equivalent PDA? What are the steps to convert a PDA to an equivalent CFG? Can you provide an example of a PDA and show its equivalent CFG? Why is it important to understand the equivalence between CFGs and PDAs? How does the equivalence between CFGs and PDAs help in parsing algorithms? What are the practical applications of converting CFGs to PDAs and vice versa? 2. **Development** (30 minutes) a. What are some challenges that arise when converting CFGs to PDAs? b. How do deterministic PDAs (DPDA) differ from non-deterministic PDAs (NPDA), and how does this affect their equivalence to CFGs? c. Can all CFGs be converted to deterministic PDAs? Why or why not? d. Provide a CFG and ask students to convert it to an equivalent PDA. e. Provide a PDA and ask students to convert it to an equivalent CFG. f. Discuss a real-world application where converting between CFGs and PDAs is useful. g. How can you prove the equivalence of a CFG and a PDA?

Closure	1. Summarize the Lesson Learning Outcomes and get affirmation from students on these. 2. Suggested Reading - https://www.iitg.ac.in/dgoswami/Flat-Notes.pdf 3. Homework - Solve the quiz questions on computational devices and model concepts and submit them on Google Classroom Spend 5 minutes to wrap up and consolidate the learnings
Evaluation	1. Reflective Questions (What, why, Who?). Allow students to answer and discuss. 2. Nearpod/ Google form Quiz Spend 5 minutes evaluating student assimilation of the lesson contents

Lesson Plan No. 4.6	Course Name: Theory of Computation Topic Name: Equivalence of PDA	Course No.: COM-604

Objectives	At the end of the lesson, the student shall be able to: a. Understand the basic structure and components of a Pushdown Automaton (PDA). b. Understand how PDAs can be equivalent to other computational models like context-free grammars (CFGs). c. Convert a given context-free grammar (CFG) into an equivalent PDA. d. Convert a given PDA into an equivalent CFG.
Teaching Aids (if any)	a. Chalk and talk b. Video of NPTEL c. Use of Nearpod tool for online quiz
Teaching Development	1. **Introduction** (5 minutes) - Ask questions What is a Pushdown Automaton (PDA) and how does it differ from a finite automaton? What are the components of a PDA? How does the stack in a PDA function, and what role does it play in recognizing language. What type of languages can PDAs recognize? How is a PDA used to recognize context-free languages? Can you provide an example of a language that can be recognized by a PDA? What does it mean for two PDAs to be equivalent? How can you prove that two PDAs are equivalent? 2. **Development** (30 minutes) a. What methods are commonly used to prove the equivalence of two PDAs? b. How can simulation techniques be used to demonstrate the equivalence of PDAs? c. Can you describe a step-by-step process to show that two given PDAs are equivalent? d. How does the equivalence of PDAs relate to compiler design and parsing? e. Can you discuss a real-world scenario where the equivalence of PDAs might be useful? f. How does the concept of PDA equivalence extend to other types of automata, such as Turing machines? g. What are the limitations of PDAs in recognizing languages, and how does this relate to the Chomsky hierarchy?

Closure	1. Summarize the Lesson Learning Outcomes and get affirmation from students on these. 2. Suggested Reading - https://www.iitg.ac.in/dgoswami/Flat-Notes.pdf 3. Homework - Solve the quiz questions on computational devices and model concepts and submit them on Google Classroom Spend 5 minutes to wrap up and consolidate the learnings
Evaluation	1. Reflective Questions (What, why, Who?). Allow students to answer and discuss. 2. Nearpod/ Google form Quiz Spend 5 minutes evaluating student assimilation of the lesson contents

Lesson Plan No. 4.7	Course Name: Theory of Computation Topic Name: Properties of recursive and recursively enumerable languages	Course No.: COM-604

Objectives	At the end of the lesson, the student shall be able to: a. Define what recursive languages and recursively enumerable languages are. b. Identify and describe the key properties of recursive languages. c. Understand the relationship between recursive languages and recursively enumerable languages. d. Illustrate the differences between these types of languages with concrete examples.
Teaching Aids (if any)	a. Chalk and talk b. Video of NPTEL c. Use of Nearpod tool for online quiz
Teaching Development	1. **Introduction** (5 minutes) - Ask questions What are the key properties of recursive languages? What are the key properties of recursively enumerable languages? How can a language be classified as recursive or recursively enumerable? Can a recursive language also be recursively enumerable? Explain with an example. What is the relationship between recursive and recursively enumerable languages in the Chomsky hierarchy? 2. **Development** (30 minutes) a. What is a reduction in the context of recursive and recursively enumerable languages? b. How are reductions used to prove that certain problems are undecidable? c. What is the significance of the halting problem in the study of recursive and recursively enumerable languages? d. How does the concept of oracle machines extend the study of recursive and recursively enumerable languages? e. Can you explain the concept of a universal Turing machine and its relevance to recursively enumerable languages? f. How do recursive and recursively enumerable languages apply to real-world computing problems? g. How do recursive and recursively enumerable languages relate to other complexity classes in computational theory?

Closure	1. Summarize the Lesson Learning Outcomes and get affirmation from students on these. 2. Suggested Reading - https://www.iitg.ac.in/dgoswami/Flat-Notes.pdf 3. Homework - Solve the quiz questions on computational devices and model concepts and submit them on Google Classroom Spend 5 minutes to wrap up and consolidate the learnings
Evaluation	1. Reflective Questions (What, why, Who?). Allow students to answer and discuss. 2. Nearpod/ Google form Quiz Spend 5 minutes evaluating student assimilation of the lesson contents

Lesson Plan No. 5.2	Course Name: Theory of Computation Topic Name: Context-sensitive language and linear bounded automata (LBA)	Course No.: COM-604

Objectives	At the end of the lesson, the student shall be able to: a. Define context-sensitive grammars and outline their components, including production rules and constraints. b. Explain how LBAs are similar to Turing machines but with space constraints. c. Create context-sensitive grammars for given languages and explain the design process. d. Analyze the efficiency and limitations of LBAs in recognizing context-sensitive languages.
Teaching Aids (if any)	a. Chalk and talk b. Video of NPTEL c. Use of Nearpod tool for online quiz
Teaching Development	1. **Introduction** (5 minutes) - Ask questions How do production rules in context-sensitive grammar differ from those in context-free grammar? Can you describe a scenario or problem where context-sensitive grammar is required? What is a linear bounded automaton, and how does it relate to context-sensitive languages? How is the computational power of an LBA characterized? What are the key components of an LBA, and how does it operate? Why are context-sensitive languages significant in the study of automata and formal languages? How does an LBA accept context-sensitive languages? 2. **Development** (30 minutes) a. Why are context-sensitive languages significant in the study of automata and formal languages? b. How does an LBA accept context-sensitive languages? c. Can you provide an example of a language that is context-sensitive but not context-free? d. What are some advanced properties or extensions of context-sensitive grammars and LBAs? e. How do context-sensitive languages and LBAs relate to other classes of languages and automata, such as Turing machines or recursive languages? f. What are the implications of context-sensitive languages for language theory and computational complexity?

Closure	1. Summarize the Lesson Learning Outcomes and get affirmation from students on these. 2. Suggested Reading - https://www.iitg.ac.in/dgoswami/Flat-Notes.pdf 3. Homework - Solve the quiz questions on computational devices and model concepts and submit them on Google Classroom Spend 5 minutes to wrap up and consolidate the learnings
Evaluation	1. Reflective Questions (What, why, Who?). Allow students to answer and discuss. 2. Nearpod/ Google form Quiz Spend 5 minutes evaluating student assimilation of the lesson contents

Lesson Plan No. 5.3	Course Name: Theory of Computation Topic Name: Chomsky hierarchy	Course No.: COM-604

Objectives	At the end of the lesson, the student shall be able to: a. Define the Chomsky Hierarchy and explain its significance in formal language theory. b. Identify the four types of grammars in the Chomsky Hierarchy: Type 0 (Unrestricted), Type 1 (Context-Sensitive), Type 2 (Context-Free), and Type 3 (Regular). c. Describe the key characteristics and constraints of each type of grammar in the Chomsky Hierarchy. d. Convert between different types of grammars, for example, from context-free to regular grammars.
Teaching Aids (if any)	a. Chalk and talk b. Video of NPTEL c. Use of Nearpod tool for online quiz
Teaching Development	1. **Introduction** (5 minutes) - Ask questions What are the main restrictions and capabilities of context-sensitive grammars? How do context-sensitive grammars relate to linear-bounded automata? Can you provide an example of a language that can be generated by context-sensitive grammar? What defines a Type 3 grammar (regular grammar) in the Chomsky Hierarchy? What are the restrictions and capabilities of regular grammars? How do regular grammars relate to finite automata? 2. **Development** (30 minutes) a. How do the different types of grammars in the Chomsky Hierarchy relate to each other in terms of language inclusion? b. Why is it said that every regular grammar is context-free, every context-free grammar is context-sensitive, and every context-sensitive grammar is unrestricted, but not necessarily vice versa? c. How do the computational power and expressive capabilities of the grammars change as you move up the Chomsky Hierarchy? d. How is the Chomsky Hierarchy used in designing compilers and interpreters? e. Can you provide examples of real-world applications or programming languages that correspond to different levels of the Chomsky Hierarchy?

Closure	1. Summarize the Lesson Learning Outcomes and get affirmation from students on these. 2. Suggested Reading - https://www.iitg.ac.in/dgoswami/Flat-Notes.pdf 3. Homework - Solve the quiz questions on computational devices and model concepts and submit them on Google Classroom Spend 5 minutes to wrap up and consolidate the learnings
Evaluation	1. Reflective Questions (What, why, Who?). Allow students to answer and discuss. 2. Nearpod/ Google form Quiz Spend 5 minutes evaluating student assimilation of the lesson contents

Lesson Plan No. 5.4	Course Name: Theory of Computation Topic Name: Decidability	Course No.: COM-604

Objectives	At the end of the lesson, the student shall be able to: a. Define decidability in the context of formal languages and automata theory. b. Explain the concept of a decision problem and its relevance to automata. c. Explain how to apply these procedures to determine if a given problem is decidable. d. Analyze specific problems and determine their decidability using the appropriate techniques.
Teaching Aids (if any)	a. Chalk and talk b. Video of NPTEL c. Use of Nearpod tool for online quiz
Teaching Development	1. **Introduction** (5 minutes) - Ask questions What is meant by the term "decidability" in the context of formal languages and automata theory? What are the differences between decidable and undecidable problems? How can you determine if a language is decidable? What does it mean for a problem to be decidable? How can you use automata to decide if a given problem is decidable? 2. **Development** (30 minutes) a. What is reducibility, and how does it relate to undecidability? b. How can one use reductions to show that a problem is undecidable? c. Can you give an example of a reduction proof demonstrating the undecidability of a language or problem? d. How does the concept of decidability affect the design and implementation of algorithms and computational models? e. What are the implications of decidability in real-world applications, such as compiler design and automated reasoning? f. What is the role of decidability in the context of complexity classes and computational limits? g. How do decidability and complexity theory intersect in automata and formal language theory?

Closure	1. Summarize the Lesson Learning Outcomes and get affirmation from students on these. 2. Suggested Reading - https://www.iitg.ac.in/dgoswami/Flat-Notes.pdf 3. Homework - Solve the quiz questions on computational devices and model concepts and submit them on Google Classroom Spend 5 minutes to wrap up and consolidate the learnings
Evaluation	1. Reflective Questions (What, why, Who?). Allow students to answer and discuss. 2. Nearpod/ Google form Quiz Spend 5 minutes evaluating student assimilation of the lesson contents

Lesson Plan No. 5.5	Course Name: Theory of Computation Topic Name: Post's correspondence problem (PCKP)	Course No.: COM-604

Objectives	At the end of the lesson, the student shall be able to: a. Define Post's Correspondence Problem (PCKP) and its significance. b. Understand and explain the proof techniques used to show the undecidability of PCKP, including reductions from other known undecidable problems. c. Identify and describe different variants of PCKP, such as the restricted PCKP and its computational characteristics. d. Explore applications of PCKP in practical contexts, such as string matching, pattern recognition, and computational linguistics.
Teaching Aids (if any)	a. Chalk and talk b. Video of NPTEL c. Use of Nearpod tool for online quiz
Teaching Development	1. **Introduction** (5 minutes) - Ask questions How is the Post's Correspondence Problem formally defined? What are the components of a PCKP instance (e.g., set of tiles, sequences)? Can you provide a simple example of a PCKP instance and explain how it is represented? How do you interpret the goal of the PCKP in terms of matching sequences? What does it mean for a PCKP instance to have a solution? How is a solution represented? Can you describe the process of attempting to find a solution to a PCKP instance? 2. **Development** (30 minutes) a. What are some theoretical and practical applications of PCKP in computer science and mathematics? b. How does the PCKP relate to other problems in formal language theory and automata? c. Can you provide real-world scenarios or problems that are analogous to PCKP? d. How can PCKP be generalized or extended to more complex versions, such as those involving multiple sets of tiles or additional constraints? e. What are some variations of PCKP and how do they differ from the classic version? f. How does PCKP connect with other areas of research, such as logic, proof theory, or complexity theory?

Closure	1. Summarize the Lesson Learning Outcomes and get affirmation from students on these. 2. Suggested Reading - https://www.iitg.ac.in/dgoswami/Flat-Notes.pdf 3. Homework - Solve the quiz questions on computational devices and model concepts and submit them on Google Classroom Spend 5 minutes to wrap up and consolidate the learnings
Evaluation	1. Reflective Questions (What, why, Who?). Allow students to answer and discuss. 2. Nearpod/ Google form Quiz Spend 5 minutes evaluating student assimilation of the lesson contents

Lesson Plan No. 5.6	Course Name: Theory of Computation Topic Name: Undecidability of PCP	Course No.: COM-604

Objectives	At the end of the lesson, the student shall be able to: a. Define the Post Correspondence Problem (PCP) and explain its components: the sets of strings and the correspondence between them. b. Explain the significance of decidability in computational theory and its implications for algorithmic problem-solving. c. Understand the concept of reduction and its role in proving undecidability. d. Explore how undecidability of PCP contributes to our understanding of the limits of computation.
Teaching Aids (if any)	a. Chalk and talk b. Video of NPTEL c. Use of Nearpod tool for online quiz
Teaching Development	1. **Introduction** (5 minutes) - Ask questions What are the components of a PCP instance (e.g., lists of strings)? How do you formally state the PCP decision problem? Why is the PCP an important problem in the study of computational theory? What is the significance of proving the undecidability of PCP? How does the concept of undecidability relate to the broader field of computability and complexity theory? Can you explain the intuitive idea behind why PCP is undecidable? What are the key steps involved in proving that PCP is undecidable? How does the reduction technique help in demonstrating the undecidability of PCP? 2. **Development** (30 minutes) a. What are the implications of PCP's undecidability for practical computation and algorithms? b. How does undecidability affect the development of algorithms in automata and formal language theory? c. What are some common methods or techniques used to approach problems involving PCP? d. How does the PCP relate to other undecidable problems in automata theory and formal languages? e. What are some advanced topics related to PCP and undecidability, such as variations of PCP or related decision problems? f. How do different formal systems or models (e.g., Turing machines, lambda calculus) relate to the PCP problem?

Closure	1. Summarize the Lesson Learning Outcomes and get affirmation from students on these. 2. Suggested Reading - https://www.iitg.ac.in/dgoswami/Flat-Notes.pdf 3. Homework - Solve the quiz questions on computational devices and model concepts and submit them on Google Classroom Spend 5 minutes to wrap up and consolidate the learnings
Evaluation	1. Reflective Questions (What, why, Who?). Allow students to answer and discuss. 2. Nearpod/ Google form Quiz Spend 5 minutes evaluating student assimilation of the lesson contents