HINDUSTHAN EDUCATIONAL AND CHARITABLE TRUST
HITECH

HINDUSTHAN INSTITUTE OFTECHNOLOGY
(An Autonomous Institution)

(Approved by AICTE, New Delhi, Affiliated to Anna University, Chennai,
Accredited with "A" Grade by NAAC and Accredited by NBA (Aero, CSE, ECE & Mech.)
Valley Campus, Pollachi Main Road, Coimbatore 641 032.

Department of Computer Science and Engineering

22CS406-AUTOMATA AND COMPILER DESIGN

PEDAGOGICAL ACTIVITY REPORT-TUTORIAL

Collaborative Learning:

AICTE PARAKH and NPTEL in conjunction to create a robust collaborative learning experience.

1. **NPTEL as the Primary Learning Resource:** Students can enroll in relevant NPTEL courses for Theory of Computation and Compiler Design. They should diligently follow lectures, complete assignments, and participate in discussion forums.
2. **PARAKH for Targeted Assessment and Feedback:** After (or even during) completing NPTEL modules, students can utilize AICTE PARAKH's self-assessment tools. If PARAKH provides specific subject-wise or topic-wise assessments for TOC and Compiler Design, students can take these tests to gauge their understanding.
    - **Example:** A student completes the NPTEL module on "Finite Automata and Regular Expressions." They then take a PARAKH assessment on "Lexical Analysis," which heavily relies on these TOC concepts. If they perform poorly, the PARAKH report can direct them back to specific NPTEL lectures or textbook sections for review.
3. **Collaborative Study Groups :** Students can form collaborative groups.
    - They can jointly watch NPTEL lectures, discuss difficult proofs in Automata Theory, work through compiler construction problems (e.g., designing a parser for a small grammar), and clarify each other's doubts.
    - They can use PARAKH's assessment insights to collectively identify common areas of confusion and focus their collaborative efforts on those topics.
4. **Preparation for GATE/Competitive Exams:** Both TOC and Compiler Design are core subjects for competitive exams like GATE. NPTEL courses are often designed to align with GATE syllabi. PARAKH, by testing fundamental and higher-order skills, can also indirectly aid in preparation by providing a structured assessment environment.
5. **Real-world Projects:** As students progress through Compiler Design, they can collaborate on mini-compiler projects, applying the theoretical knowledge from NPTEL to practical implementation. This hands-on experience, perhaps influenced by the skill expectations highlighted by PARAKH, can significantly deepen understanding.

By actively engaging with NPTEL courses for in-depth learning and utilizing AICTE PARAKH for structured assessment and feedback, students can achieve a comprehensive and practical understanding of Automata Theory and Compiler Design, bridging the gap between theoretical knowledge and industry-relevant skills.

CO-PO-PSO Mapping:

Course outcome –Program outcome - Program specific outcomes -Mapping Table

22CS406 – AUTOMATA AND COMPILER DESIGN		Program outcomes 1-Slight, 2-Moderate , 3-High												Program specific outcomes	
		PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2
CO1	Understand the basic properties of formal languages and regular sets	3	-	-	-	-	-	-	-	-	-	-	-	2	-
CO2	Design the Context free grammar and Push down automata for a context free Language	-	-	3	-	-	-	-	-	-	-	-	-	2	3
CO3	Design a lexical analyzer to identify the tokens in a program	-	2	2	-	-	-	-	-	-	-	-	-	2	3
CO4	Construct the different Translation Schemes.	2	2	-	-	-	-	-	-	-	-	-	-	2	3
CO5	Analyze various code optimizatio n techniques	-	-	3	-	-	-	-	-	-	-	-	-	2	-

JUSTIFICATION

CO1: Understand the basic properties of formal languages and regular sets.

- **PO2 (Problem Analysis): 2 (Moderate)**
  - **Justification:** This CO involves analyzing the grammar rules and determining how to associate semantic actions or code generation rules to produce a meaningful translation, which is a form of problem analysis.
- **PSO1 (Apply programming skills & implement algorithms): 2 (Moderate)**
  - **Justification:** Translation schemes are essentially algorithms that define how source code is transformed into another form. Understanding and constructing them directly enhances the ability to implement complex algorithms for language processing.
- **PSO2 (Design, develop, test, and integrate software/hardware): 3 (High)**
  - **Justification:** Translation schemes are integral parts of the semantic analysis and intermediate code generation phases of a compiler. Constructing these schemes is crucial for designing and developing the core functionality of a compiler as an effective software system.

CO5: Analyze various code optimization techniques.

- **PO3 (Design/Development of Solutions): 3 (High)**
  - **Justification:** Analyzing optimization techniques is critical for *improving* the efficiency, performance, and effectiveness of software solutions. This directly relates to enhancing the design and development of robust computer-based systems.
- **PSO1 (Apply programming skills & implement algorithms): 2 (Moderate)**
  - **Justification:** Analyzing optimization techniques requires a deep understanding of the algorithms involved in code transformation and their impact on performance. This knowledge directly informs the implementation of more efficient algorithms and the writing of optimized code.

EVIDENCE :

Student Learning Assessment

ALL INDIA COUNCIL FOR TECHNICAL EDUCATION, NEW DELHI

Student Learning Assessment

SANJAIKUMAR R S (SLAS1469453)

3rd Year

COMPUTER SCIENCE AND ENGINEERING

HINDUSTHAN INSTITUTE OF TECHNOLOGY (1-4294141)

Date : 09-05-2025

Type : Self-Assessment

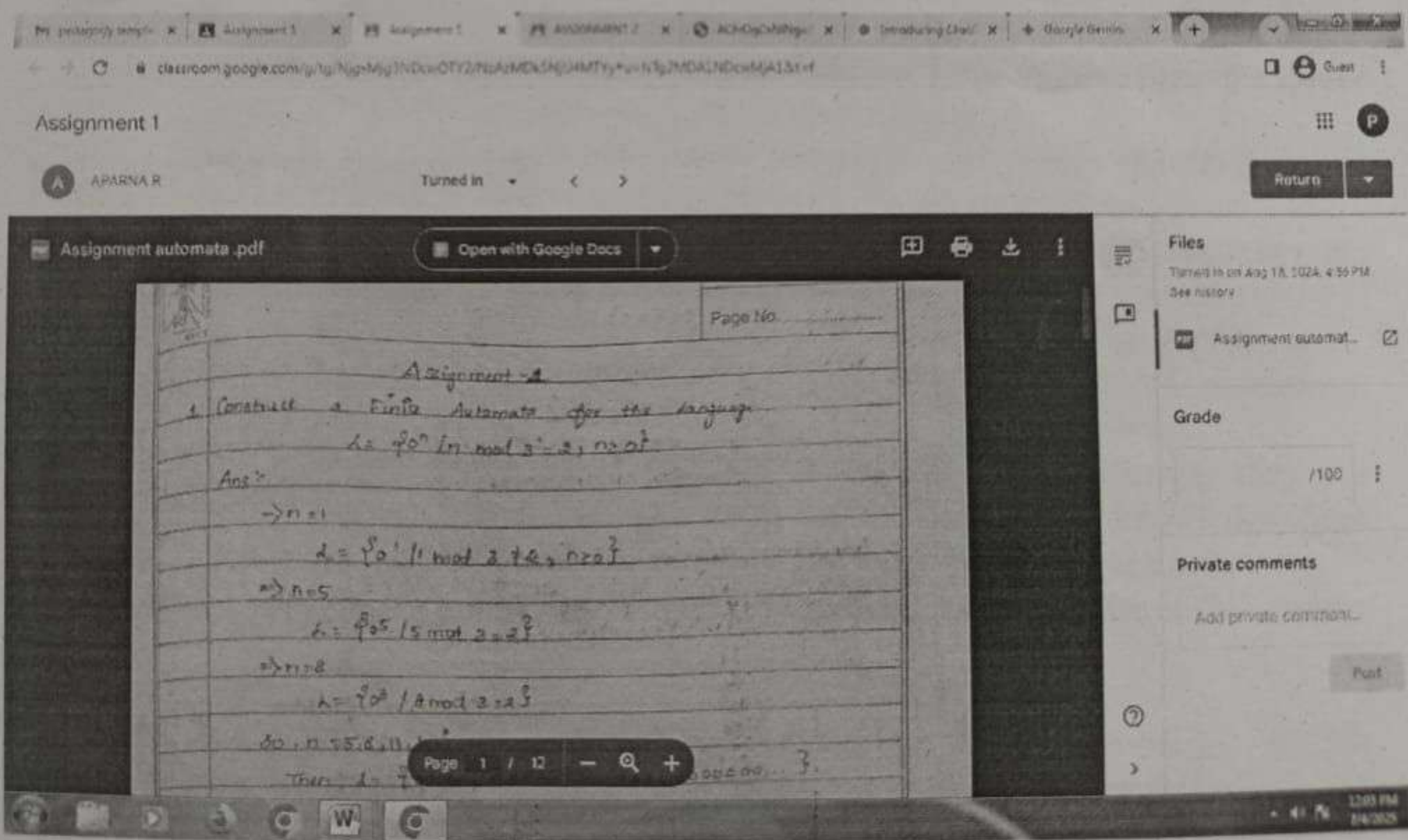
Overall Rating

★★★☆☆

★★★★★ Excellent | ★★★★☆ Very Good | ★★★☆☆ Good

Artificial Intelligence ★★★☆☆

- You be valuable as an engineer.
- Innovative and good technical skills.



- **Justification:** This CO represents fundamental theoretical knowledge in computer science. Understanding formal languages, alphabets, strings, and regular sets is a core pillar of computing theory, directly aligning with applying basic engineering knowledge.

- **PSO1 (Apply programming skills & implement algorithms): 2 (Moderate)**
  - **Justification:** Understanding the properties of formal languages is foundational for comprehending how programming languages are structured and for designing efficient algorithms for string processing, pattern matching (e.g., using regular expressions), and language recognition, which are integral to programming skills and algorithm implementation.

CO2: Design the Context-Free Grammar and Push down automata for a context free Language.

- **PO3 (Design/Development of Solutions): 3 (High)**
  - **Justification:** The verb "Design" in this CO directly aligns with PO3. Students are creating abstract computational models (CFGs and PDAs) to define and recognize languages, which are critical components in the design phase of language processing software.
- **PSO1 (Apply programming skills & implement algorithms): 2 (Moderate)**
  - **Justification:** Designing CFGs and PDAs provides the blueprint for parsing algorithms and language recognition algorithms. This conceptual design work is essential for anyone who will later implement or apply these algorithms in code.
- **PSO2 (Design, develop, test, and integrate software/hardware): 3 (High)**
  - **Justification:** Context-Free Grammars and Pushdown Automata are the theoretical underpinnings for the parser component of a compiler. The ability to design these abstract models directly translates into the ability to design and develop a key software component of a computer-based system.

CO3: Design a lexical analyzer to identify the tokens in a program.

- **PO2 (Problem Analysis): 2 (Moderate)**
  - **Justification:** Designing a lexical analyzer requires students to analyze the input source code, understand its structure, and define rules for breaking it down into meaningful tokens, which is a problem analysis task.
- **PO3 (Design/Development of Solutions): 2 (Moderate)**
  - **Justification:** This CO involves the practical "Design" of a specific and vital component (lexical analyzer) within the larger system of a compiler, directly contributing to solution development.
- **PSO1 (Apply programming skills & implement algorithms): 2 (Moderate)**
  - **Justification:** Lexical analysis heavily relies on applying pattern matching algorithms (often based on finite automata and regular expressions) which are then implemented using programming skills.
- **PSO2 (Design, develop, test, and integrate software/hardware): 3 (High)**
  - **Justification:** A lexical analyzer is a tangible software component. This CO directly involves designing such a component that will be integrated into a larger compiler system, demonstrating a high ability in constructing computer-based systems.

CO4: Construct the different Translation Schemes.

- **PO1 (Engineering Knowledge): 2 (Moderate)**
  - **Justification:** Constructing translation schemes requires specific engineering knowledge about syntax-directed translation, semantic rules, and the generation of intermediate representations within the compiler's pipeline.