SAHYADRI
COLLEGE OF ENGINEERING & MANAGEMENT
An Autonomous Institution
MANGALURU

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

ATC Assignment-Academic Year 2024-25

COURSE NAME: AUTOMATA THEORY AND COMPUTABILITY

COURSE CODE:CS522T3C

SECTION: 5A

COURSE INSTRUCTOR: Dr. Shivanna K

NAME OF THE STUDENTS WITH USN:

1. BRINDA U – 4SF22CS045
2. PRAJNA S KOTIAN – 4SF22CS136
3. PRIYA K R – 4SF22CS153
4. SHREYA K – 4SF22CS204

6/02/2025

Evaluation Metrics:

1	Problem Understanding and Modeling	Clear definition of states, transitions, alphabets and symbols.		
			3M	
2	Automaton implementation	Correct Implementation of DFA / NFA /PDA /TM using JFLAP.		
			6M	
3	Example Analysis	Accurate interpretation of input sequence and resulting output.		
			7M	
4	Analysis and Interpretation	Accurate interpretation of results and implications.		
			4M	
		Total		20
		Signature		

INTRODUCTION

Automata theory, a pivotal field in computer science, explores abstract machines and their computational capabilities, laying the theoretical groundwork for various computing models and formal languages. This assignment utilizes JFLAP software to tackle three key problems in deterministic finite automata (DFA), non-deterministic pushdown automata (NPDA), and finite for regular expressions.

t systematically presents methodologies for solving each problem, showcases results AP software simulations, engages in discussions highlighting the findings' significance, ludes with insights into automata theory's broader implications in computer science. these exercises, our aim is to deepen our understanding of formal languages, automata, r practical applications in solving real-world computational challenges. Subsequent will delve into problem statements, methodologies, results, discussions, and conclusions, ng a concise exploration of applied concepts within the realm of automata theory.

PROBLEM STATEMENT:

- Given the Alphabet {a,b,c} construct a DFA which accepts (a|b)*c
- Design a non deterministic PDA for accepting the language L = {a^i b^j c^k d^l : i==k or j==l, i>=1, j>=1}
- Construct finite automata which accepts the languages defined by the following regular expressions {w | w ∈ {0, 1}* ∧ w doesn't end in 01}

METHODOLOGY:

1. Given the Alphabet {a,b,c} construct a DFA which accepts (a|b)*c

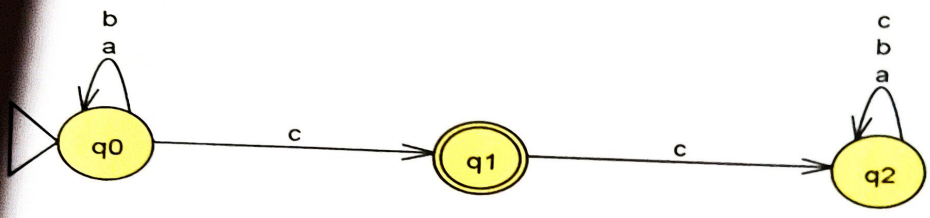

Table Text Size

Input	Result
aabac	Accept
bbaac	Accept
abcaa	Reject
abacba	Reject

1. Open JFLAP:
   - Launch JFLAP and select "Finite Automaton" to create a new DFA.

2. Add States:
   - Add three states: (q0)-> (start state), (q1)-> (accept state), and (q2) ->(dead state).

3. Set Start and Accept States:
   - Mark (q0) as the start state with an arrow pointing to it.
   - Mark (q1) as the accept state (double circle).

4. Define Transition:
   - (q0 to q0): Add self-loop for inputs 'a' and 'b'.
   - (q0 to q1): Add transition for input 'c'.
   - (q1 to q2): Add transition for input 'c'.
   - (q2 to q2): Add self-loop for inputs 'a', 'b', and 'c'.

5. Verify the DFA:
   - Test the DFA using JFLAP's input testing feature to ensure correct acceptance and rejection of strings.

6. Save the DFA:
   - Save the DFA in JFLAP format or export the diagram for documentation.

2. **Design a non-deterministic PDA for accepting the language L = {a^i b^j c^k d^l : i==k or j==l, i>=1, j>=1}**

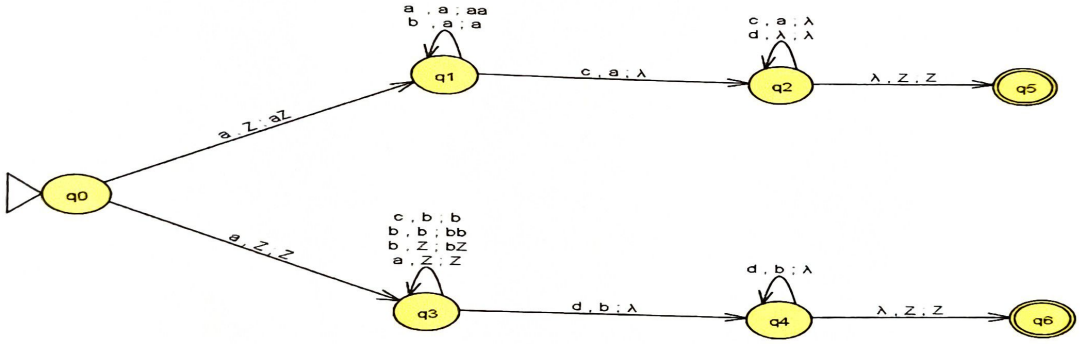

Table Text Size

Input	Result
aaabcccd	Accept
aabbbccddd	Accept
aabbccdd	Accept
abcd	Accept
abc	Reject
a	Reject

1. Open JFLAP:
   - Launch the JFLAP software and select "Pushdown Automaton" from the main menu.

2. Set Up States:

- Add states (q0, q1, q2, q3, q4, q5, q6).
- Set (q0) as the initial state and (q5, q6) as accepting states.

3. Define Transitions:

(q0):
- Transition ((a, Z; aZ)) to (q1).
- Transition ((a, Z; Z)) to (q3).

(q1):
- Transition (a, a; aa)), ((b, a; a)) self-loop.
- Transition ((c, a; λ)) to (q_2).

(q2):
- Self-loop ((c, a; λ)), ((d, λ; λ))
- Transition ((λ, Z; Z)) to (q5).

(q3):
- Transition ((c, b; b)), ((b, b; bb)), ((b, Z; bZ)), ((a, Z; Z)) self-loop.
- Transition ((d, b; λ)) to (q4).

(q4):
- Self-loop ((d, b; λ)).
- Transition ((λ, Z; Z)) to (q6).

4. Test and Save:

- Simulate different inputs to ensure the PDA behaves as required.
- Save the PDA and export the image for future use.

3. Construct finite automata which accepts the languages defined by the following regular expressions {w | w ∈ {0, 1}* ∧ w doesn't end in 01}

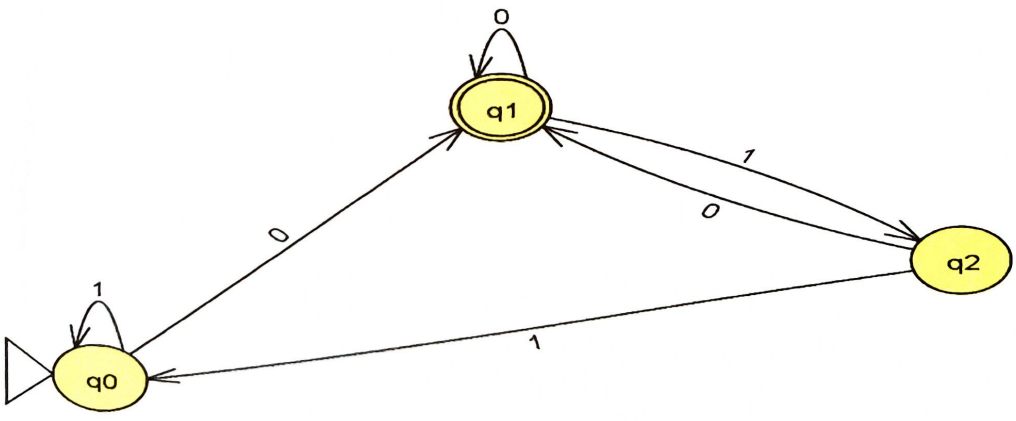

Table Text Size

Input	Result
0	Accept
01	Reject
101	Reject
000	Accept
1000101001	Reject
1010	Accept

1. Open JFLAP:
   - Launch the JFLAP software and select "Finite Automaton".

2. Set Up States:
   - Add three states (q0, q1,q2).
   - Mark (q0) as the initial state and (q0, q1) as accepting states.

3. Define Transitions:
   - (q0):
     - Self-loop (1).
     - Transition (0) to (q1).
   - (q1):
     - Self-loop (0).
     - Transition (1) to (q2).
   - (q2):
     - Transition (0) to (q1).
     - Transition (1) to (q0).

4. Test and Save:
   - Simulate binary strings (e.g., "00", "10", "101") to ensure strings ending in "01" are rejected.
   - Save the file and export the diagram as needed.

RESULTS:

1. Given the Alphabet {a,b,c} construct a DFA which accepts (a|b)*c

The DFA has three states:

- A start state to process any number of aaas and bbbs before ccc.
- A final state for strings ending with ccc.
- A dead state for strings with additional characters after ccc.

Transitions:

- Stay in the start state for any aaa or bbb.
- Transition to the final state when ccc is encountered.
- Transition to the dead state if any character follows ccc.

Acceptance:

- A string is accepted if it ends with exactly one ccc after any sequence of aaas and bbbs.

Rejection:

- A string is rejected if it contains no ccc, if ccc is not the last character, or if any character follows ccc.
- Behavior verified through simulations to ensure correctness.

2. Design a non deterministic PDA for accepting the language L = {a^i b^j c^k d^l : i==k or j==l, i>=1, j>=1}

Stack Mechanism:

- The stack is used to compare aaas with cccs and bbbs with ddds by pushing and popping symbols to track counts.
- Non-determinism allows the PDA to guess whether to verify i=ki = ki=k or j=lj = lj=l.

Transitions:

- For i=ki = ki=k: Push an aaa onto the stack for every aaa read, and pop for every ccc.
  Accept if the stack is empty after processing all cccs.
- For j=lj = lj=l: Push a placeholder onto the stack for every bbb read, and pop for every ddd.
  Accept if the stack is empty after processing all ddds.

Non-determinism:

- At the start, the PDA guesses whether to track aaa-ccc parity (i=ki = ki=k) or bbb-ddd parity (j=lj = lj=l).

Acceptance Conditions:

- The PDA accepts if the stack is empty and all input is consumed, ensuring either i=ki = ki=k or j=lj = lj=l.

Rejection Conditions:

- The PDA rejects if neither condition (i=ki = ki=k or j=lj = lj=l) is satisfied, such as unmatched
- numbers of aaas and cccs or bbbs and ddds.

Behavior Verified:

- Through simulations, the PDA demonstrated the ability to process strings like a3b2c3d1a^3 b^2 c^3 d^1a3b2c3d1 (where i=ki = ki=k) and a2b3c1d3a^2 b^3 c^1 d^3a2b3c1d3 (where j=lj = lj=l), confirming correctness.
- Rejected strings such as a2b3c1d2a^2 b^3 c^1 d^2a2b3c1d2 (neither i=ki = ki=k nor j=lj = lj=l).
- This PDA efficiently handles the given language by leveraging non-determinism for choosing the correct parity to check.

Construct finite automata which accepts the languages defined by the following regular expressions {w | w ∈ {0, 1}* ∧ w doesn't end in 01} Pattern Description:

- The language consists of all strings over {0,1}\{0, 1\}{0,1} that do not end in "01".
- Strings ending with "0" or "1", or those containing "01" but not as the last two characters, are accepted.

Construction Approach:

- The automaton tracks the last one or two characters of the string to detect if the string ends in "01".
- States correspond to whether the last input leads to the "01" pattern.

NFA Design:

- The NFA processes any sequence of 000s and 111s while ensuring that transitions prevent strings ending in "01" from being accepted.
- If "01" is detected at the end, the automaton transitions to a rejecting state.

DFA Conversion:

- The DFA eliminates nondeterminism by explicitly tracking all possible states and ensuring the machine keeps track of whether it is currently safe to accept the string.
- States include combinations such as "safe to accept" (no trailing "01") and "rejecting" (trailing "01")

Acceptance Conditions:

- A string is accepted if it does not end with "01".
- For example, strings like "10", "111", or "100" are accepted.

Rejection Conditions:

- A string is rejected if it ends with "01".
- For example, "101", "001", or "01" are rejected.

Simulations and Verification:

- Strings like "1010" and "1111" were correctly accepted, as they do not end in "01".
- Strings like "1001" and "01" were rejected, verifying the automaton's correctness.

Outcome:

- The DFA accurately recognizes strings that do not end in "01", confirming its functionality.

DISCUSSION:

1. Discussion Points for DFA Constant

- The DFA construction for the language (a|b)*c requires careful state management to ensure correct acceptance of strings.
- Ensuring that 'c' is always the last symbol in the accepted strings is crucial for the DFA's correctness.
- Minimizing states in the DFA is essential to reduce complexity while maintaining correctness.
- The DFA should be designed to handle empty strings, and its behavior in such cases should be clearly defined.
- Constructing a DFA for this language involves a systematic approach to ensure coverage of all possible input strings.

2. Discussion Points for Non-Deterministic PDA Construction

- The non-deterministic PDA for the language L = {a^i b^j c^k d^l : i==k or j==l, i>=1, j>=1} relies heavily on stack management to track symbol counts.
- Verifying the equality of symbol counts is a critical aspect of the PDA's design, requiring careful consideration of stack operations.
- Non-determinism in the PDA allows for flexibility in handling different input strings, but also increases the complexity of the design.
- Ensuring the correctness of the PDA involves rigorous testing and validation to guarantee that it accepts the intended language.
- The non-deterministic PDA's design is influenced by the need to balance correctness with efficiency in terms of stack usage and computation.

3. Discussion Points for Finite Automaton Construction

- The finite automaton for the language $\{w \mid w \in \{0, 1\}^* \wedge w \text{ doesn't end in } 01\}$ relies on recognizing patterns in binary strings.
- Handling leading zeros in binary strings is an important consideration in the design of the finite automaton.
- Minimizing states in the finite automaton is crucial to reduce complexity while maintaining correctness.
- Ensuring the correctness of the finite automaton involves rigorous testing and validation to guarantee that it accepts the intended language.
- The finite automaton's design is influenced by the need to balance correctness with efficiency in terms of state usage and computation.

COMPARATIVE ANALYSIS:

Complexity:

1. **DFA Construction for (a|b)_c_:** Moderate complexity. Designing a DFA for this language requires careful state management and ensuring correct state transitions.

2. **Non-Deterministic PDA for L = {a^i b^j c^k d^l : i==k or j==l, i>=1, j>=1}:** High complexity. This problem requires a deep understanding of non-deterministic PDAs and stack manipulation to verify the intricate relationships between the counts of 'a's, 'b's, 'c's, and 'd's.

3. **Finite Automaton for {w | w∈ {0, 1}_ ∧ w doesn't end in 01}_:** Moderate complexity. Constructing a finite automaton for this language requires understanding how to recognize patterns and validate inputs according to the given regular expression.

Conceptual Understanding:

1. **DFA Construction for (a|b)_c__:** Requires understanding of deterministic finite automata, state transitions, and how to design a DFA for a specific language.

2. **Non-Deterministic PDA for L = {a^i b^j c^k d^l : i==k or j==l, i>=1, j>=1}:** Demands a strong understanding of non-deterministic PDAs, stack manipulation, and how to verify complex relationships between symbol counts.

3. **Finite Automaton for {w | w∈ {0, 1}_ ∧ w doesn't end in 01}_:** Involves understanding how to convert regular expressions to finite automata and recognize patterns in binary strings.

Application:

1. **DFA Construction for (a|b)_c__:** Applicable in pattern matching, lexical analysis, and text processing tasks, particularly when the system needs to match specific sequences of symbols.

2. **Non-Deterministic PDA for L = {a^i b^j c^k d^l : i==k or j==l, i>=1, j>=1}:** Has applications in compiler design, parsing, and formal language theory, particularly when dealing with complex relationships between symbol counts.

3. **Finite Automaton for {w | w∈ {0, 1}_ ∧ w doesn't end in 01}_:** Used in lexical analyzers, text processing, and data validation tasks, particularly when dealing with binary strings and pattern recognition.

Challenge Level:

1. **DFA Construction for (a|b)_c__:** Medium challenge level. Requires careful state management and DFA design, but is relatively straightforward compared to the other problems.
2. **Non-Deterministic PDA for L = {a^i b^j c^k d^l : i==k or j==l, i>=1, j>=1}:** High challenge level. Demands a deep understanding of non-deterministic PDAs and complex relationships between symbol counts.
3. **Finite Automaton for {w | w∈ {0, 1}_ ∧ w doesn't end in 01}_:** Medium challenge level. Requires understanding how to convert regular expressions to finite automata and recognize patterns in binary strings.

CONCLUSION:

The construction of automata for various languages presents a fascinating challenge in the realm of theoretical computer science. The DFA for (a|b)*c requires meticulous state management to ensure that the machine accepts strings ending in 'c' after any sequence of 'a's and 'b's. In contrast, the non-deterministic PDA for L = {a^i b^j c^k d^l : i==k or j==l, i>=1, j>=1} demands a deep understanding of stack manipulation and non-deterministic computation to verify the intricate relationships between the counts of 'a's, 'b's, 'c's, and 'd's.

Similarly, the finite automaton for {w | w ∈ {0, 1}* ∧ w doesn't end in 01} necessitates careful consideration of the language's constraints, ensuring that the machine rejects strings ending in '01' while accepting all other binary strings. These problems not only test one's grasp of automata theory but also hone skills in analytical thinking, problem-solving, and theoretical computer science.

In conclusion, the journey through these problems illustrates the escalating complexity and sophistication of computational models, from the straightforward yet elegant DFA constructions to the more intricate and theoretically rich non-deterministic PDA designs. Each problem presents a unique opportunity to explore the frontiers of automata theory and computational language theory, ultimately deepening our understanding of the fundamental principles governing computation.

REFERENCES:

The completion of this assignment drew upon foundational concepts from the field of automata theory. The following resources served as references for theoretical principles, problem-solving strategies, and the use of JFLAP software:

.1 Hopcroft, .J E., Motwani, R., & Ullman, J. D. (2006). Introduction to Automata Theory, Languages, and Computation. Pearson.

2.Sipser, M. (2012). Introduction to the Theory of Computation. engage Learning.

3JFLAP Documentation and Tutorials. Retrieved from http://www.jflap.org/). JFLAP Official Website