RAMAIAH
Institute of Technology

An Assignment on

DFA and Turing Machine Assignment

Theory of Computation IS54

Submitted to

Dr. S R Mani Sekhar

Associate Professor

Submitted by:

K Sree Vaishnavi

(1MS22IS057)

Nikhilesh N Marali

(1MS22IS090)

Department of Information Science and Engineering

RAMAIAH INSTITUTE OF TECHNOLOGY

Index Sheet

References

- https://www.jflap.org
- Introduction to Automata Theory, Languages and Computation, 3rd Edition, Pearson education, 2014
- https://youtu.be/lRKE2RznkUU?feature=shared

Q7) Design DFA that accept binary string containing both '101' and '110' as substring.

1. Introduction

In Formal Language Theory, a Deterministic Finite Automaton (DFA) is a mathematical model used to recognize patterns in strings of symbols. This report discusses the construction of a DFA in JFLAP.

A DFA is a simple machine that reads input symbols one at a time and transitions between states based on the current state and the input symbol. It accepts or rejects the input string based on whether it ends in an "accept" state.

Key Uses:

- **Pattern recognition:** Finding specific patterns in text.
- **Lexical analysis:** Identifying tokens (keywords, identifiers) in code.
- **Protocol validation:** Checking if data conforms to a specific format.

2. Problem Statement

Design a single-tape deterministic Turing Machine to accept precisely those strings of the form:

L = {w | w contains both '101' and '110' as substrings}

Any string in LLL consists of:

1. **At least one occurrence of '101'** as a substring.
2. **At least one occurrence of '110'** as a substring.
3. These substrings can occur in **any order**, and they may **overlap**.

For example:

- Strings in LLL: "101110", "110101", "10100110", "1110110","1101".
- Strings not in LLL: "1010", "100110", "111".

This report provides a detailed explanation of the design, implementation, and validation of this DFA using the JFLAP tool. The machine should ensure that the input string contains both '101' and '110' as substrings, in any order or with possible overlap.

Any deviation—whether an incorrect count of characters, an incorrect order, or an absence of required blocks—should lead the machine to reject.

3. High-Level Design Approach

Understand the Requirements:

- The DFA must:
  - Detect the substring '101'.
  - Detect the substring '110'.
  - Accept only if **both substrings** are present in the input string.

State Design:

- **Initial State (q0)**:
  - Start of the DFA, no progress toward detecting '101' or '110'.
- **Tracking States (q1, q2, q3)**:
  - Track progress toward '101' and '110' simultaneously.
  - q1: '1' seen, starting detection of both substrings.
  - q2: '10' seen, progressing toward '101'.
  - q3: '11' seen, progressing toward '110'.
- **Detection States (q4, q5)**:
  - q4: '101' detected, now look for '110'.
  - q5: '110' detected, now look for '101'.
- **Intermediate State (q6)**:
  - Handle overlaps between '101' and '110' (e.g., '1101').
- **Final State (q9)**:
  - Accepting state, both substrings detected.

Transitions:

- **Simultaneous Detection**: Use transitions to track prefixes of '101' and '110' concurrently.
- **Overlap Handling**: For cases like '1101', ensure the DFA continues detecting the second substring without restarting.
- **Acceptance**: Transition to q9 only when both substrings are detected.

General Workflow:

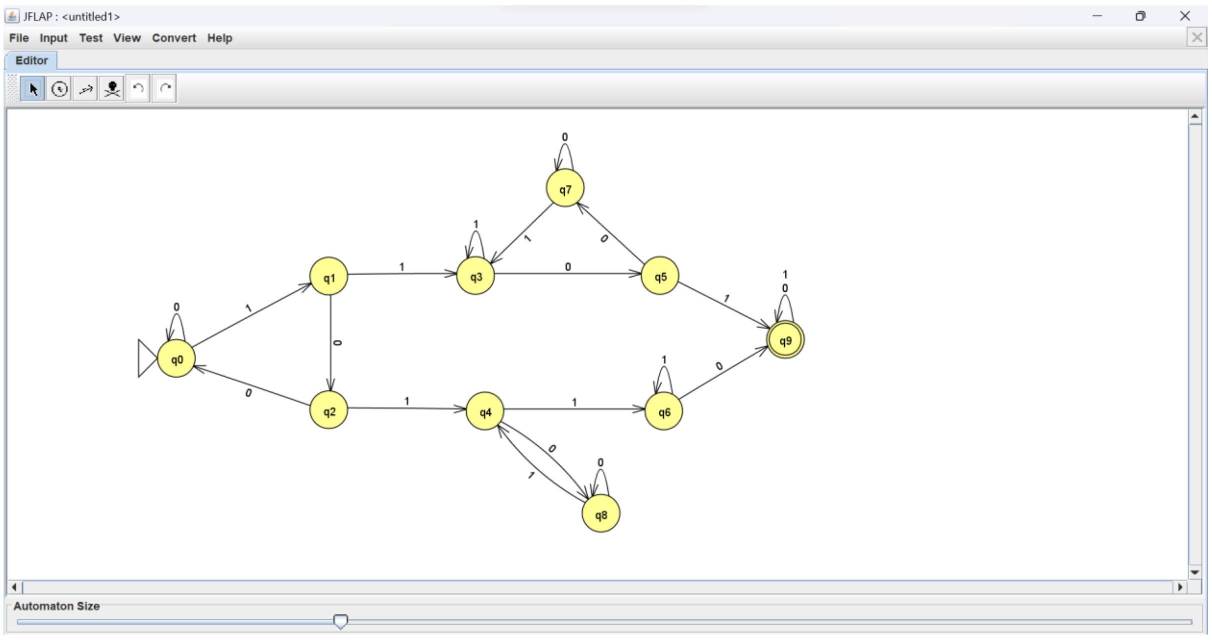
- Start at q0.
- Read the input string symbol by symbol.
- Transition between states based on whether input matches prefixes of '101' or '110'.
- Accept if both substrings are found (reach q9)
- Reject if the input ends without reaching q9.

Efficiency

- The DFA processes the input in **O (n)** time, where n is the length of the string.
- Tracks both substrings simultaneously, minimizing redundant computations.

4. Explanation of States and Transitions

Below is a summary of each state and its purpose. The figure (screenshot in JFLAP) illustrates how these states connect.



States:

- q0: Start state, no progress towards '101' or '110'.
- q1: '1' seen, possible start of '101' or '110'.
- q2: '10' seen, progressing towards '101'.
- q3: '11' seen, progressing towards '110'.
- q4: '101' detected, now searching for '110'.
- q5: '110' detected, now searching for '101'.
- q6: Intermediate state, handling overlaps.
- q9: Final state, both '101' and '110' detected.

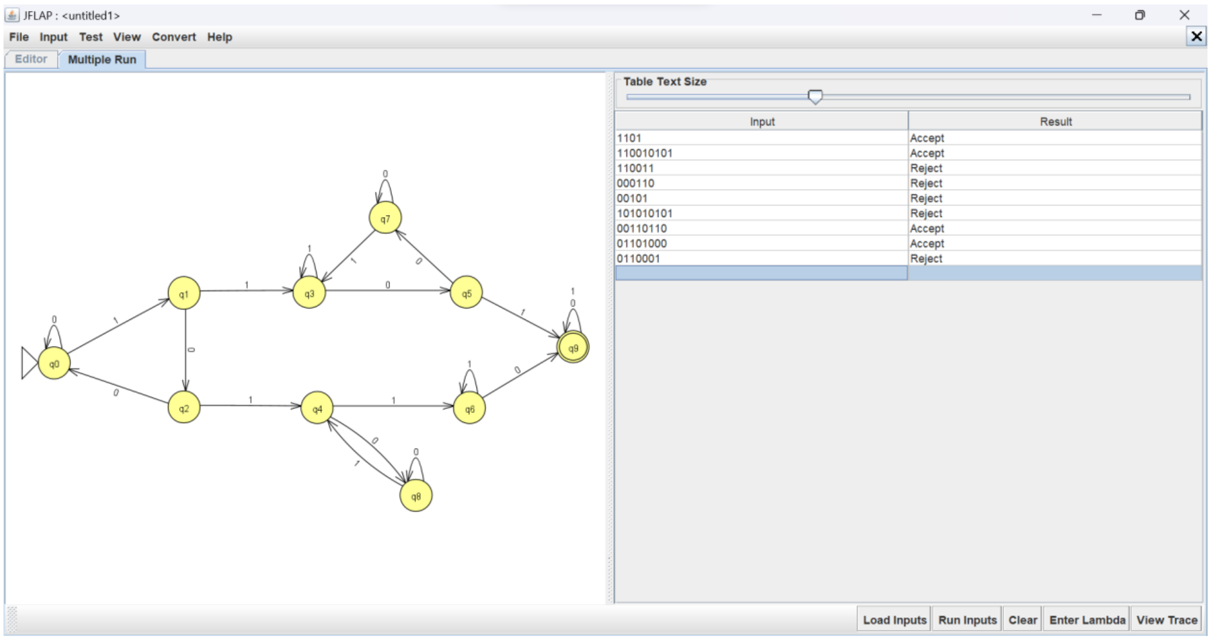
Transitions:

- Track prefixes of '101' and '110' simultaneously.
- Handle overlaps (e.g., '1101' contains both substrings).
- Accept only when both substrings are found (transition to q9).

How the DFA Works:

- The DFA starts at q0 and reads the input string one symbol at a time.
- As it reads the input, it transitions between states based on the prefixes of '101' and '110' it encounters.
- If both '101' and '110' are found, the DFA transitions to q9, which is the accepting state.
- If the input ends without reaching q9, the string is rejected.

5. Sample Traces



Input	Result
1101	Accept
110010101	Accept
110011	Reject
000110	Reject
00101	Reject
101010101	Reject
00110110	Accept
01101000	Accept
0110001	Reject

5.1. Valid Input Example

Consider the string: 1101

Trace Table for w=1101:

Step	Current State	Input Symbol	Next State	Remark
1	q0	1	q1	Transition on 1
2	q1	1	q3	Transition on 1
3	q3	0	q5	Transition on 0
4	q5	1	q9	Transition on 1

Conclusion: Since the DFA ends in a final state (q9), the string w=1101 is accepted.

5.2. Invalid Input Example

Consider the string: 00101

Trace Table for w=00101:

Step	Current State	Input Symbol	Next State	Remark
1	q0	0	q0	Transition on 0
2	q0	0	q0	Transition on 0
3	q0	1	q1	Transition on 1
4	q1	0	q2	Transition on 0
5	q2	1	q4	Transition on 1

Conclusion: Since the DFA ends in a non-final state (q4), the string w=00101 is **rejected**.

6. Final Representation

The final DFA can be described as: **M = (Q, Σ, δ, q0, F)**

Q = {q0, q1, q2, q3, q4, q5, q6, q7, q8, q9}

Σ = {0, 1}

q0 is the start state q0

q9 is the final state F.

δ – Transition Functions:

From q0:	**From q1:**	**From q2:**	**From q3:**
δ(q0,0)=q0	δ(q1,0)=q2	δ(q2,0)=q0	δ(q3,0)=q5
δ(q0,1)=q1	δ(q1,1)=q3	δ(q2,1)=q4	δ(q3,1)=q3
From q4:	**From q5:**	**From q6:**	**From q7:**
δ(q4,0)=q8	δ(q5,0)=q7	δ(q6,0)=q9	δ(q7,0)=q7
δ(q4,1)=q6	δ(q5,1)=q9	δ(q6,1)=q6	δ(q7,1)=q3
From q8:	**From q9:**		
δ(q8,0)=q8	δ(q9,0)=q9		
δ(q8,1)=q4	δ(q9,1)=q9		

Transition Table:

State	Input 0	Input 1
q0	q0	q1
q1	q2	q3
q2	q0	q4
q3	q5	q3
q4	q8	q6
q5	q7	q9
q6	q9	q6
q7	q7	q3
q8	q8	q4
q9	q9	q9

7. Conclusion

The constructed DFA, as implemented in JFLAP, successfully recognizes strings in the language **L = {w | w contains both '101' and '110' as substrings}**

The machine ensures:

- **At least one occurrence of '101'** as a substring.
- **At least one occurrence of '110'** as a substring.
- These substrings can occur in **any order**, and they may **overlap**.
- Immediate rejection of invalid patterns, such as unmatched symbols or incorrect order.

This implementation demonstrates the systematic use of marking, scanning, and state transitions to validate the given language. The behaviour of the machine can be verified through step-by-step execution in JFLAP using both valid and invalid test cases.

Q1) Design a Turing Machine to recognize the language L= {a^n b^n | n ≥ 1}. The machine should check if the number of a's is equal to the number of b's and reject otherwise.

1. Introduction

In formal language theory, a Turing Machine (TM) is the most powerful computational model used to recognize recursively enumerable languages. This report discusses the construction of a Turing Machine in JFLAP that decides the language:

L = {a^n b^n | n ≥ 1}

Any string in L consists of:

- A non-empty block of a's,
- Followed by a non-empty block of b's,
- With the number of a's equal to the number of b's.

All other strings (those that have an unequal number of a's and b's, or the wrong order of letters, or are missing either a or b) must be rejected.

This report provides a detailed explanation of the design, implementation, and validation of this Turing Machine using the JFLAP tool.

2. Problem Statement

Design a single-tape deterministic Turing Machine to accept precisely those strings of the form:

L = {a^nb^n | n ≥ 1}

The machine should ensure that the number of as is equal to the number of bs, and that all as appear before any bs.

Any deviation—whether an incorrect count of characters, an incorrect order, or an absence of required blocks—should lead the machine to reject.

3. High-Level Design Approach

3.1. Marking Scheme

- a → X: When an **a** is matched (used up), it is replaced with **X**.
- b → Y: When a **b** is matched, it is replaced with **Y**.
- Blank is denoted by ☐ on the tape.

This marking ensures that each a is paired with a corresponding b, and no unmatched characters remain.

3.2. Matching Strategy

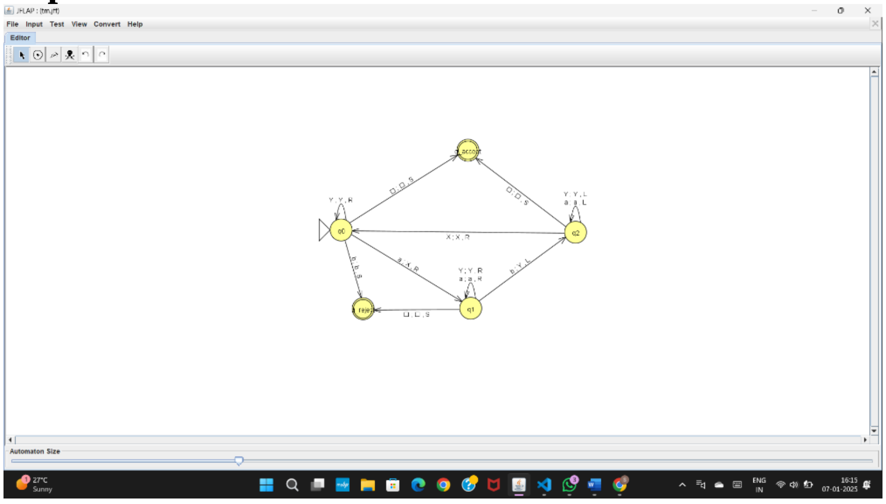
- Match each a with a b.
- From left to right, each a is replaced with **X**, and the machine looks for an unmarked b to replace with **Y**. This ensures a **one-to-one pairing** between as and bs.
- After marking a pair, the machine returns to the beginning to search for the next unmarked a.
- The process continues until all pairs are matched.

3.3. At Least One a and One b

- The machine is designed to **reject** if it encounters a b before seeing any a.
- It also **rejects** if there are more as than bs or vice versa.
- If the machine encounters a blank (☐) after successfully matching all as and bs, it transitions to the **accept state**.
- Any deviation from these rules causes the machine to transition to the **reject state**.

---

4. Explanation of States and Transitions

5. 

Below is a summary of each state (labeled q0, q1, q2, q_accept, q_reject) and its purpose. The figure (screenshot in JFLAP) illustrates how these states connect.

(q0) – Start State

- Expects an a. On reading an a, mark it as X and move right to q1.
- If it sees a b or a blank (☐) instead, it rejects the string by transitioning to q_reject.

(q1) – Match b for a just-marked a

- Scans right looking for an unmarked b.
- On finding a b, mark it as Y and move left to q2.
- If a blank (☐) is encountered before finding a b, the machine rejects the string by transitioning to q_reject.

(q2) – Return to Start

- Scans left to find the next unmarked a.
- Ignore already marked symbols (Y) and move left until encountering X.
- When X is encountered, move right to q0 to continue the matching process.

(q0) – Restart Matching Process

- The machine continues looking for the next unmarked a to repeat the matching cycle.
- If no more as are found and the head encounters a blank (☐), the machine transitions to q_accept.

(q_accept) – Final State (Accept State)

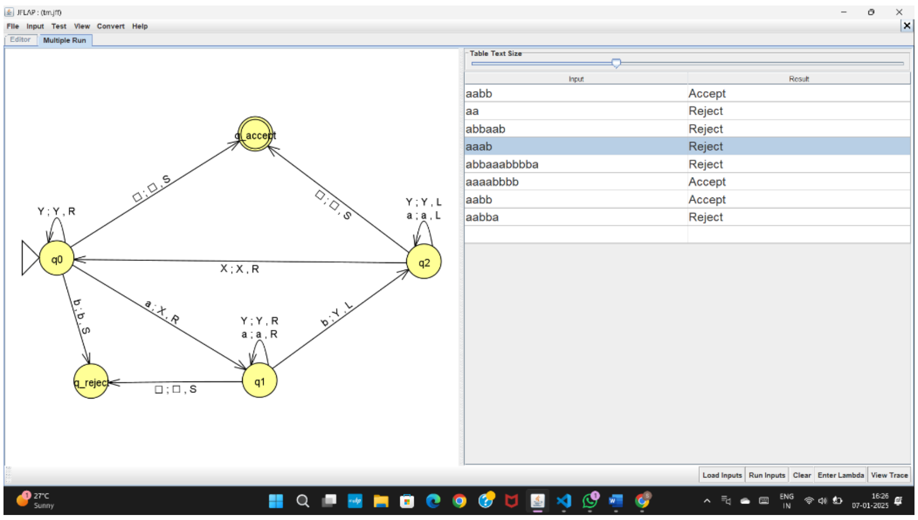
- If a blank (☐) is encountered after all as and bs are matched and marked, the string is accepted.

(q_reject) – Reject State

- Any unexpected situation, such as encountering a b before marking all as, encountering unmatched characters, or finding a mismatch in the count of as and bs, leads to this state.

---

5. Sample Traces



Input	Result
aabb	Accept
aa	Reject
abbaab	Reject
aaab	Reject
abbaaabbbba	Reject
aaaabbbb	Accept
aabb	Accept
aabba	Reject

5.1. Valid Input Example

Consider the string: aabb

- The machine starts at the first a, marks it as X, and moves right to find the corresponding b.
- Upon encountering the first b, it marks it as Y and moves left to return to the next unmarked a.

- The process is repeated for the second a and b.
- Once all as and bs are marked as X and Y, the machine scans the tape.
- If no unmarked symbols remain, and the head encounters a blank symbol (☐), the string is **accepted**.

Trace Table for Valid String aabb:

Step	Tape	Head Position	Current State
1	aabb☐	a	q0
2	Xabb☐	b	q1
3	XabY☐	a	q2
4	XXbY☐	b	q0
5	XXYY☐	☐	q_accept

The machine transitions to the **accept state**, confirming the string is valid.

5.2. Invalid Input Example

Consider the string: aab

- The machine marks the first a and attempts to find a corresponding b.
- Upon finding a mismatch or encountering an unmatched a, the machine transitions to the **reject state**.

Trace Table for Invalid String aab**:**

The machine transitions to the **reject state**, confirming the string is invalid.

Step	Tape	Head Position	Current State
1	aab☐	a	q0
2	Xab☐	b	q1
3	Xab☐	b	q_reject

The machine detects a b before matching all as and transitions immediately to the **reject state**.

6. Final Representation

The final Turing Machine can be described as:

M = (Q, Σ, Γ, δ, q0, ☐, q_accept, q_reject)

- Q = {q0, q1, q2, q_accept, q_reject }
- Σ = {a, b}
- Γ = {a, b, X, Y, ☐}
- q0 is the start state.

- ☐ is the blank symbol.
- q_accept is the accept state.
- q_reject is the reject state.
- δ is the transition function defined below:

From q0:

- (q0, a) → (q1, X, R)
- (q0, b) → (q_reject, b, S)
- (q0, ☐) → (q_accept, ☐, S)

From q1:

- (q1, a) → (q1, a, R)
- (q1, b) → (q2, Y, L)
- (q1, ☐) → (q_reject, ☐, S)

From q2:

- (q2, a) → (q2, a, L)
- (q2, X) → (q0, X, R)
- (q2, Y) → (q2, Y, L)

From q_accept:

- The machine halts and accepts the string when no mismatches are detected.

From q_reject:

- The machine halts and rejects the string when any mismatch, improper ordering, or unexpected symbol is encountered.

---

7. Conclusion

The constructed Turing Machine, as implemented in JFLAP, successfully recognizes strings in the language $L = \{a^n b^n \mid n \geq 1\}$.

The machine ensures:

- Proper ordering, where all as appear before bs.
- Equal counts of as and bs.
- Immediate rejection of invalid patterns, such as unmatched symbols or incorrect order.

This implementation demonstrates the systematic use of marking, scanning, and state transitions to validate the given language. The behaviour of the machine can be verified through step-by-step execution in JFLAP using both valid and invalid test cases.