Project Report
Binary Pattern Detector using DFA

Zaid Ahmed (66037)
Muhammad Ahmed (66073)

Theory of Automata (118692)
Faculty: Miss Misbah Anwer

Contents

1 Introduction

This project presents an interactive Python-based tool for detecting fifteen different binary string patterns over the alphabet $\{0, 1\}$. Deterministic Finite Automata (DFA) are employed to recognize these regular patterns and to provide a formal verification mechanism for user-supplied binary inputs. Users select a rule from a menu and enter a binary string; the program then generates a step-by-step state transition trace and determines whether the string is accepted or rejected. The `automata-lib` library is used to construct automata from regular expressions, making advanced automata theory accessible through a simple console-based interface.

1.1 Background

Finite automata form the foundation of formal language theory in computer science. Deterministic Finite Automata recognize regular languages and are widely used in compilers, network protocols, data validation, and pattern matching. Binary pattern detection is particularly useful in enforcing structural constraints in data streams and communication systems.

1.2 Motivation

Manual DFA construction and testing can be time-consuming and error-prone for students. This project automates common DFA problems such as detecting even length strings, enforcing symbol order constraints, and limiting occurrences of symbols. It provides an educational yet practical environment for understanding DFA behavior.

1.3 Significance

The project bridges theoretical automata concepts with real-world validation systems. The interactive transition traces enhance learning and demonstrate how DFAs operate internally, making the system valuable for both academic and applied contexts.

2 Related Work

Deterministic Finite Automata are extensively used in lexical analysis, intrusion detection systems, protocol verification, and pattern matching engines. Research focuses on optimizing DFA memory usage and improving transition efficiency for large-scale pattern detection systems. Other works include:

- Neural networks simulating DFAs for parity and sequence checking.
- Delayed-input DFA (D2FA) to optimize memory usage in hardware.
- Symbolic string analysis using automata abstractions for program verification.
- Memory-efficient bit-split DFAs for high-speed network packet inspection.

3 Methodology

The system implements a console-based simulator that constructs DFAs from regular expressions and evaluates user-input binary strings. The automata are determinized, totalized using a dead state, and relabeled for readability.

3.1 System Design

- Menu-driven rule selection
- Regular expression to DFA conversion
- Dead state handling for invalid transitions
- Step-by-step transition tracing

3.2 DFA States

States are relabeled as $q_0, q_1, \ldots$ with q_0 as the initial state and q_d as the dead state.

3.3 Input Symbols

The alphabet is $\Sigma = \{0, 1\}$. Input validation ensures correctness before execution.

3.4 Transition Function

Transitions are generated automatically using automata-lib and guarantee deterministic execution.

3.5 Validation and Output

The simulator prints the initial state, each transition, and the final state, followed by acceptance or rejection.

4 Results and Experiments

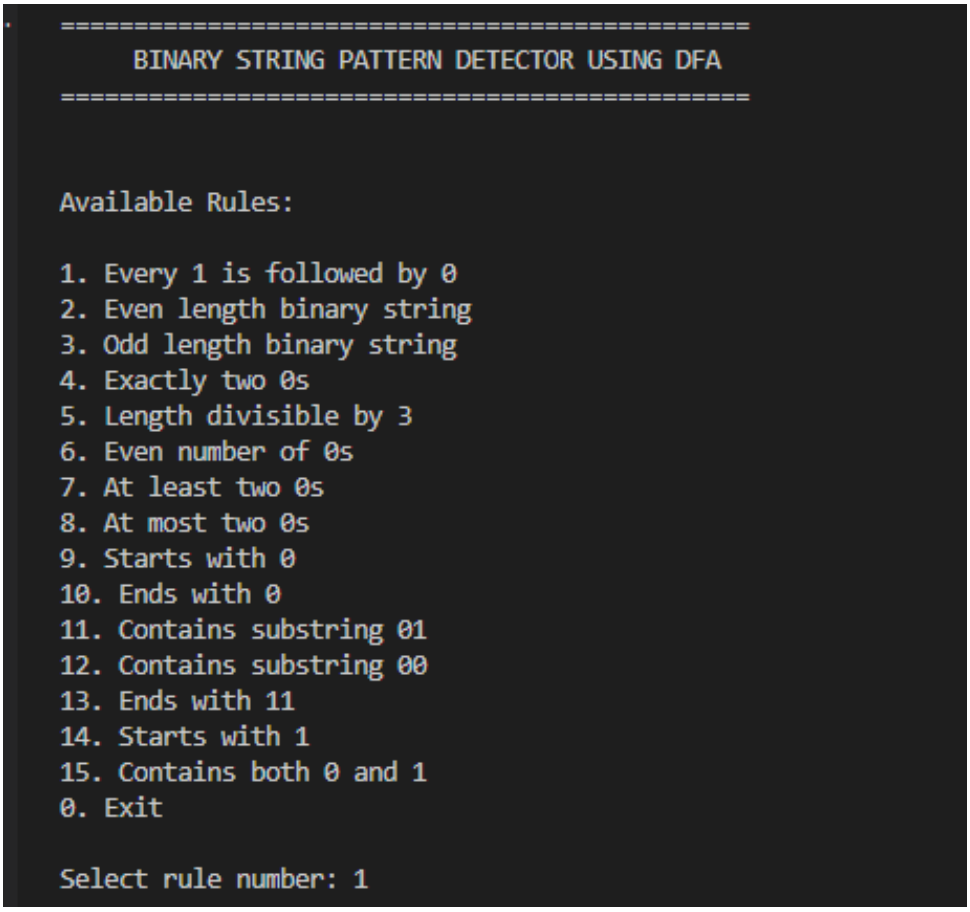

Figure 1: Output Display 1

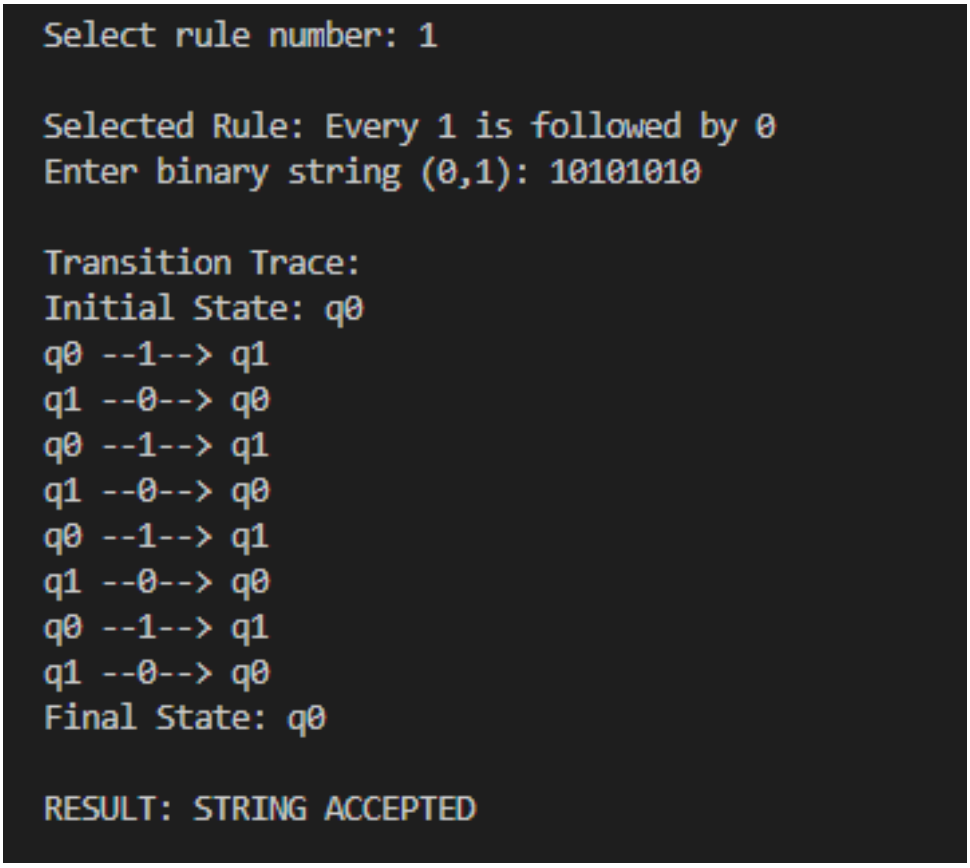

Figure 2: Output Display 2

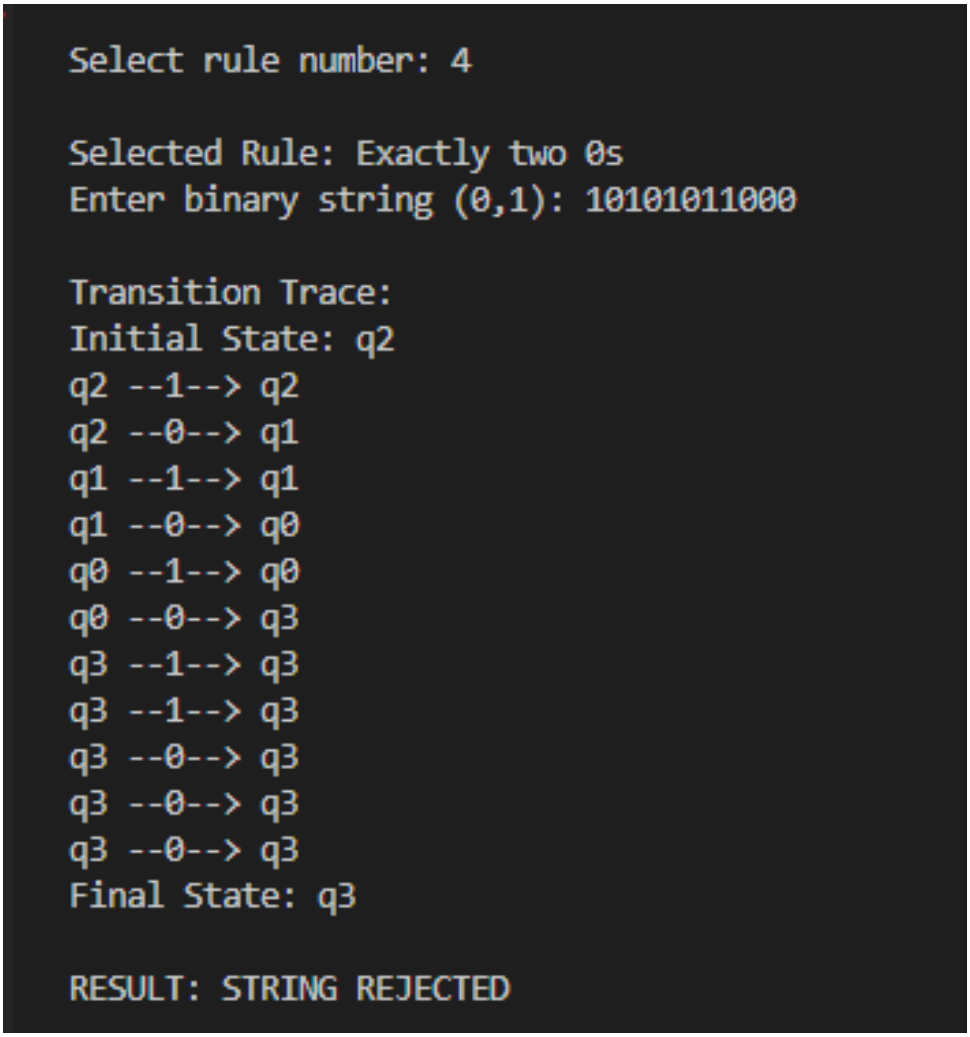

Figure 3: Output Display 3

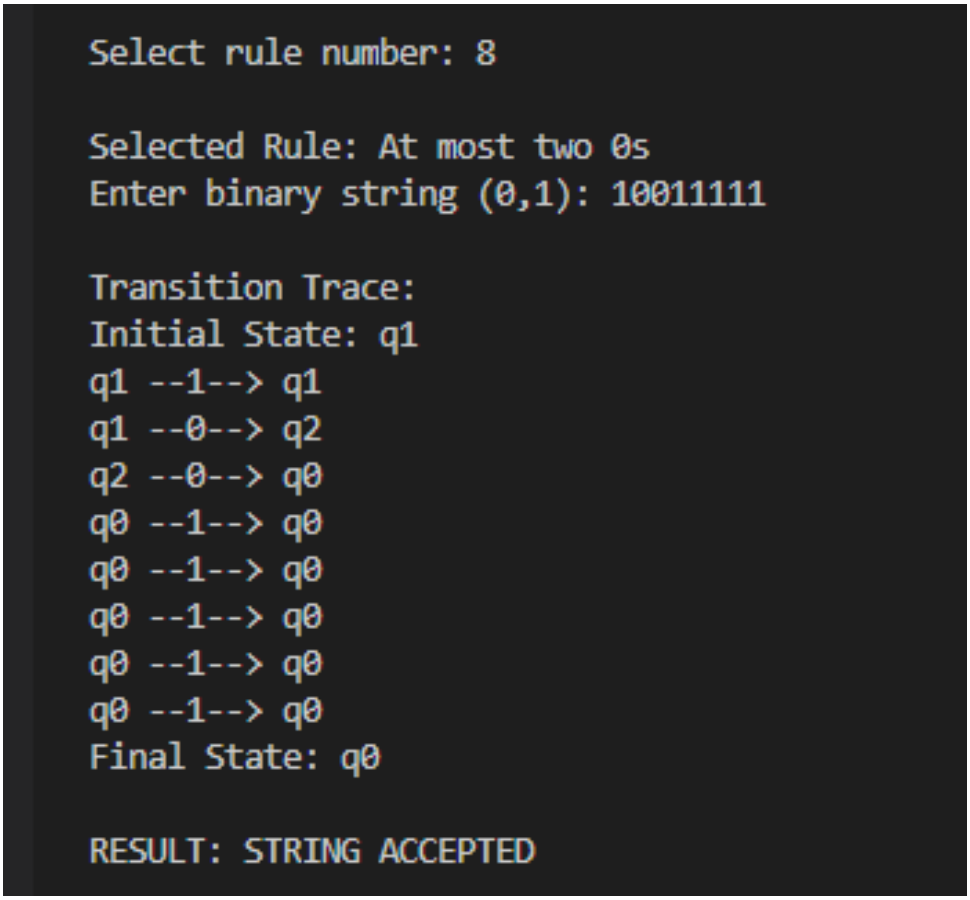

Figure 4: Output Display 4

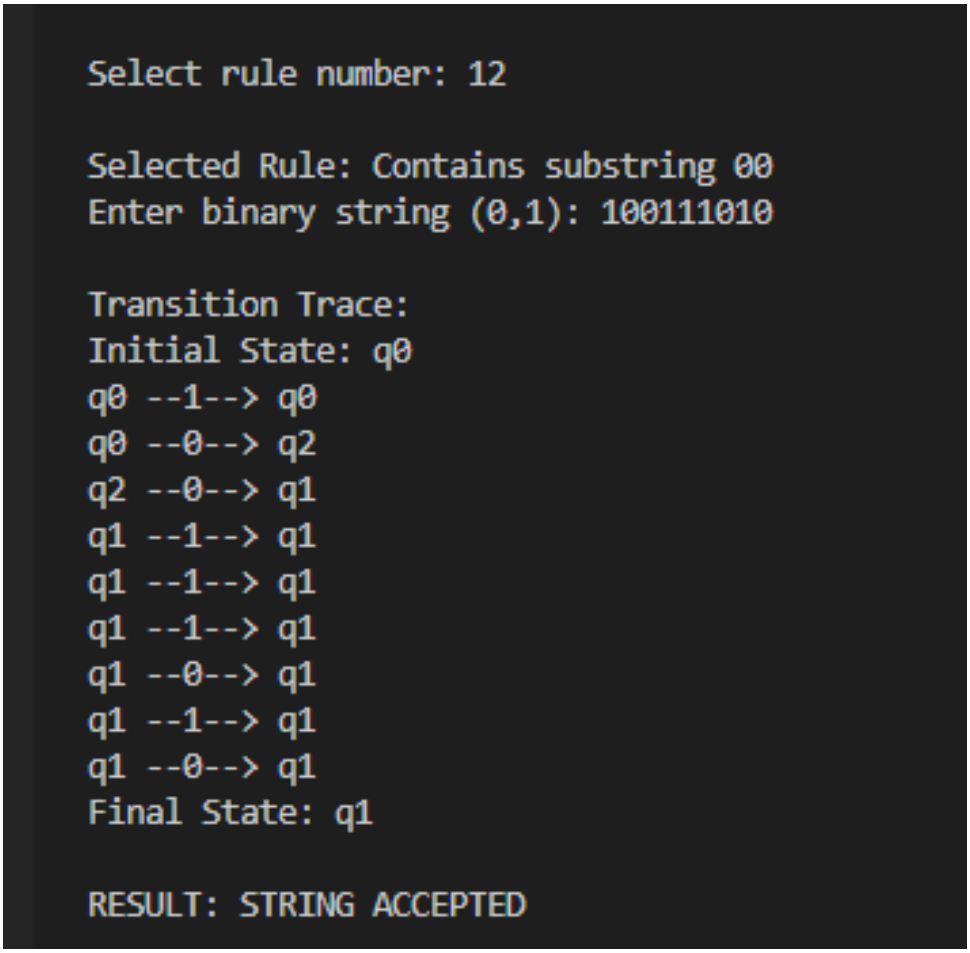

Figure 5: Output Display 5

```
Select rule number: 15

Selected Rule: Contains both 0 and 1
Enter binary string (0,1): 1111111111111111111

Transition Trace:
Initial State: q0
q0 --1--> q3
q3 --1--> q3
q3 --1--> q3
q3 --1--> q3
q3 --1--> q3
q3 --1--> q3
q3 --1--> q3
q3 --1--> q3
q3 --1--> q3
q3 --1--> q3
q3 --1--> q3
q3 --1--> q3
q3 --1--> q3
q3 --1--> q3
q3 --1--> q3
q3 --1--> q3
q3 --1--> q3
q3 --1--> q3
q3 --1--> q3
q3 --1--> q3
q3 --1--> q3
Final State: q3

RESULT: STRING REJECTED
```

Figure 6: Output Display 6

5 Conclusion

This project successfully demonstrates the practical implementation of DFA-based binary pattern detection. By combining regular expressions, DFA construction, and transition tracing, the system provides both educational insight and real-world applicability. The modular design allows future extensions such as DFA minimization and graphical visualization.

References

[1] Sipser, M. *Introduction to the Theory of Computation*. Cengage Learning.

[2] Hopcroft, J. E., Motwani, R., Ullman, J. D. *Introduction to Automata Theory, Languages, and Computation*. Pearson.

[3] Automata-lib Documentation. https://pypi.org/project/automata-lib/

[4] DFA Applications – GeeksforGeeks. https://www.geeksforgeeks.org/theory-of-computation/application-of-deterministic-finite-automata-dfa/

[5] Regular Expressions and Finite Automata. https://www.geeksforgeeks.org/regular-expressions-regular-grammar-and-regular-languages/

[6] Python Official Documentation. https://docs.python.org/3/

[7] DFA and NFA Conversion Concepts. https://www.javatpoint.com/dfa-and-nfa

[8] Formal Languages and Automata Theory Notes. https://www.tutorialspoint.com/automata_theory/index.htm

[9] Neural Networks as Universal Finite-State Machines. https://arxiv.org/html/2505.11694v2

[10] Efficient String Matching Using DFA. IJCA. https://research.ijcaonline.org/volume44/number18/pxc3878750.pdf

[11] Symbolic String Verification: An Automata-based Approach. https://sites.cs.ucsb.edu/~bultan/publications/spin08.pdf

[12] Memory-Efficient DFA-Based Bit-Split. https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0126517

[13] Algorithm Design for Deterministic Finite Automata. https://www.bhu.ac.in/research_pub/jsr/Volumes/JSR_66_02_2022/3.pdf