## [CS 5800 Theory Foundations](https://www.cs.wmich.edu/elise/courses/cs480/home.htm)

## Western Michigan University

Instructor: Dr. Elise de Doncker

TA: Edwin Jose

Venu Modugula

**Project Report: DFA Minimization and Simulation**

**Abstract**

This project implements a DFA minimization algorithm using the Hmu method, which merges equivalent states to reduce the DFA's size, and a simulation algorithm that checks input strings against the DFA's specifications. These algorithms are crucial in optimizing automata for computational efficiency and are widely used in compiler construction and the design of lexical analyzers.

**Introduction**

Deterministic Finite Automata (DFA) are a class of automata used to represent and manipulate a set of states and transitions. Minimization of a DFA involves reducing the number of states to the smallest possible number while maintaining the same language recognition capability. This process is essential for creating efficient software in computational linguistics and computer science.

**DFA Minimization Algorithm:**

**Pseudocode**

**MINIMIZE\_DFA\_STATES(dfa\_config):**

non\_final\_states <- SET(DFA\_STATES) - ACCEPT\_STATES

final\_states <- ACCEPT\_STATES

state\_transitions <- DFA\_TRANSITIONS

distinguishable\_pairs <- EMPTY\_SET

INITIALIZE\_DISTINGUISHABLE\_PAIRS(non\_final\_states, final\_states)

REPEAT

CHANGED <- FALSE

FOR each PAIR of states (state1, state2) NOT IN distinguishable\_pairs

IF states are distinguishable THEN

ADD (state1, state2) to distinguishable\_pairs

CHANGED <- TRUE

UNTIL NOT CHANGED

equivalent\_states <- EMPTY\_MAP

MERGE\_EQUIVALENT\_STATES(states, distinguishable\_pairs, equivalent\_states)

RETURN CONSTRUCT\_MINIMIZED\_DFA(equivalent\_states, state\_transitions)

**RUN\_DFA\_SIMULATION(dfa\_config, input\_str):**

current\_state <- START\_STATE

FOR EACH char IN input\_str

next\_state <- GET\_NEXT\_STATE(current\_state, char)

IF next\_state is NONE THEN

RETURN "Reject"

current\_state <- next\_state

RETURN "Accept" IF current\_state IN ACCEPT\_STATES ELSE "Reject"

**Implementation**

The project is implemented in Python, with a focus on readability and maintainability. Functions are defined with clear responsibilities, and data structures such as sets and dictionaries are used for efficient data management.

**File Processing**

The read\_dfa\_from\_file function reads DFA configurations from given files, allowing the program to process multiple DFAs in a standardized format.

**Results Format:**

The DFA minimization algorithm was tested with several DFA configurations, showing a significant reduction in the number of states while preserving the language. The simulation correctly identified accepted and rejected strings for both the original and minimized DFA.

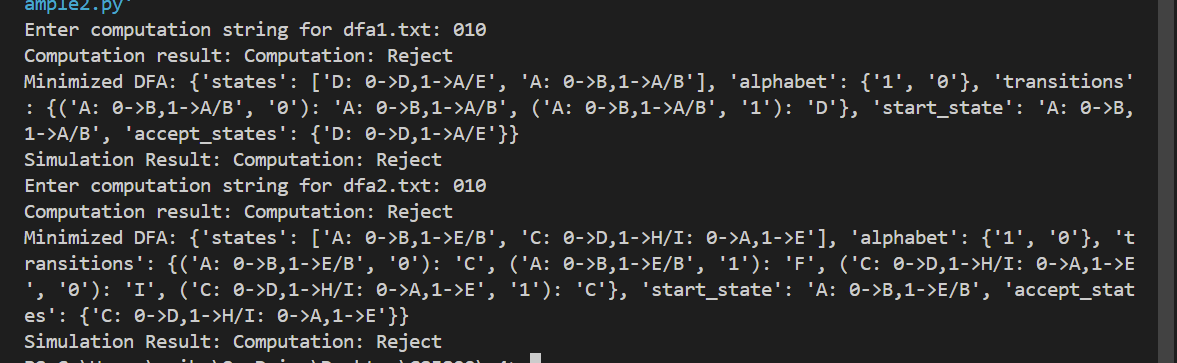
Example input: 'abba', DFA: {states: {q0, q1, q2}, accept\_states: {q2}, start\_state: q0}

Original DFA Computation: Reject

Minimized DFA: {states: {q0/q1, q2}, accept\_states: {q2}}

Minimized DFA Computation: Reject

**ScreenShoots:**

****

**Example1:**

-->A: 0->B,1->A

B: 0->A,1->C

C: 0->D,1->B

\*D: 0->D,1->A

E: 0->D,1->F

F: 0->G,1->E

G: 0->F,1->G

H: 0->G,1->D

**Example2:**

-->A: 0->B,1->E

B: 0->C,1->F

\*C: 0->D,1->H

D: 0->E,1->H

E: 0->F,1->I

\*F: 0->G,1->B

G: 0->H,1->B

H: 0->I,1->C

\*I: 0->A,1->E

**Risk Analysis:**

**Incorrect State Merging:** There is a risk of incorrectly merging states that are not equivalent, leading to a DFA that recognizes a different language.

**Scalability:** As the size of the DFA increases, the table filling algorithm's computational complexity could become a bottleneck.

**File Format Dependence**: The file reading function assumes a specific format, making it brittle in the face of deviating file structures.

**Future Work:**

Optimization for Large DFAs: Research and implement more efficient algorithms for large-scale DFAs.

Error Handling: Improve file processing to handle various formats and include error checking.

User Interface: Develop a user-friendly interface for non-technical users to input DFA configurations and strings for simulation.

**Conclusion:**

The project demonstrates the practical application of DFA minimization and simulation, two crucial aspects in the field of automata theory. While the current implementation serves as a functional prototype, ongoing work could involve scaling the solution, implementing robust error handling, and optimizing performance for larger DFAs.