Texas A&M university-corpus christi
Computer science department

Modeling, Implementation, and Analytical Comparison of Computational Machines Across Formal Languages

By
Bhragav Venkat Sai Guptha perla
Venkata Mohith penneti
Madhusudhan poduturu

for
COSC6356.001 - Theory of Computation
Dr. Minhua Huang
Fall 2025

November 23,2025

Table of content:

Introduction:

Automata theory is the very foundation of computer science since it gives a formal description of the different types of languages and the abstract computational models that can recognize them. The range of power of these models goes from the simplest (deterministic machines) to the most advanced (universal computation). Therefore, it is very important to know the operational behavior and the limits of these models from the point of view of compiler design, language processing, algorithm design, and complexity theory.

The project will undertake the design, implementation, simulation, and evaluation of five fundamental computational models:

- **Deterministic Finite Automaton (DFA)**
- **Nondeterministic Finite Automaton (NFA)**
- **ε-NFA**
- **Pushdown Automaton (PDA)**
- **Turing Machine (TM)**

The formal language specification provided in the project PDF is used to construct each model, and the model is then implemented in Python and integrated into an interactive Automata Simulator based on Streamlit.

The system enables the users to:

- Input strings
- View step-by-step processing
- Visualize state diagrams
- Generate random test cases
- Analyze machine acceptance

The intention is to illustrate the hierarchy of computation from regular languages to context-free and context-sensitive ones.

Literature Review:

2.1 Formal Languages & the Chomsky Hierarchy

The Chomsky Hierarchy is a classification of languages into four categories:

- Regular languages — recognized by DFAs, NFAs, ε-NFAs
- Context-Free languages — recognized by PDAs
- Context-Sensitive languages — recognized by Linear-Bounded Turing Machines
- Recursively Enumerable languages — recognized by general Turing Machines

This project is limited to the machines of the first three levels.

2.2 Deterministic Finite Automaton (DFA)

A DFA has:

- Limited number of states
- Deterministic transition function
- Acceptance states

Regular languages are recognized by DFAs which function with exactly one possible transition for each state-signal pair.

2.3 Nondeterministic Automaton (NFA) & ε–NFA

An NFA permits the following:

- It can have more than one transition from the same input.
- There may be transitions without taking any input (ε-transitions).

This flexibility enables language design to be simpler for some cases.

2.4 Pushdown Automaton (PDA)

A PDA has a stack which allows the machine to identify patterns with recursion or symmetry. For example:

$$a^n b^n$$

Using the stack operations the automaton can keep trace of the prior input.

2.5 Turing Machine (TM)

The Turing Machine is the utmost computation abstraction, which can imitate any algorithm.
It consists of:

- A tape (infinite memory)
- A head (reading/writing and moving left/right) States and transitions Our Turing

Machine can identify:

$$a^n b^n c^n (n \geq 1)$$

2.6 Applications of Automata

Automata are consulted in various fields by:

- The process of analyzing the source code for tokens in a compiler
- Verifying the conformance of the network protocols
- Breaking down the natural language
- Managing the tasks of the operating system
- Generating plans by AI
- Implementing the regular expressions
- Designing the logic circuits in hardware

3.Design and Implementation

This endeavor realizes five machines—DFA, NFA, ε–NFA, PDA, and Turing machine—according to the formal language definitions provided in the project briefing. The application is made in Python and Streamlit is used for front-end display, permitting user and machine-made random input interactive execution.

The simulator reveals the execution in detail step by step and demonstrates the final active state in the state diagram after input processing.

In this section, we describe the process of designing the automata according to the language specifications and the implementation of logic in Python.

3.1 System Input Features 3.1.1 User Input

To each automaton, the user can use a text field to manually input any string of their choice. The input undergoes validation symbol-wise and is then modified in accordance with the transition rules of the automaton chosen.

3.1.2 Randomly Generated Input

The simulator additionally provides the Random Test functionality:

- DFA: random binary strings
- NFA / ε–NFA: random sequences from {a, b}
- PDA: random strings of the form $a^n b^n$
- TM: random strings of the form $a^n b^n c^n$

The generation of these inputs follows the valid patterns of each language to ensure correctness and robustness testing.

3.2 Output Visualization and Final State Highlighting

- After processing the input:
- The simulator gives a comprehensive step-by-step trace
- A state sequence is saved
- A final acceptance/rejection decision is shown
- The final state of the current process is marked in the state diagram of the automaton, with:
  Active state colored yellow
  Red border to attract attention

This makes it easier to understand the connection between the trace and the visual model.

3.3 Automata Implementation Details

3.3.1 DFA Design — Even Number of Zeros

Language:

$$L = \{ \mid w \text{ contains an even number of zeros } \}$$

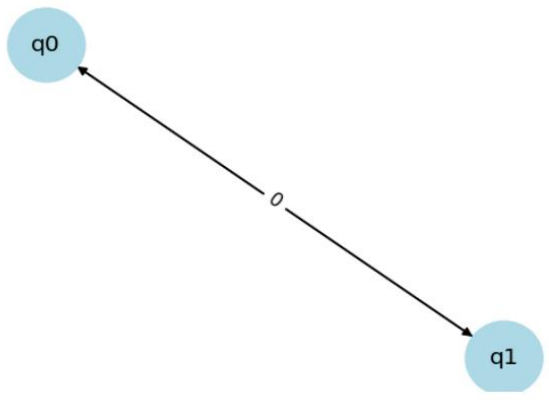
Design:

- q0 — even 0s count (starting point, accepting state)
- q1 — odd 0s count

State	Input	Next
q0	0	q1
q0	1	q0
q1	0	q0
q1	1	q1

Implementation Notes:

- Uses deterministic dictionary
- Simple step tracking
- Final state determines acceptance



3.3.2 NFA Construction

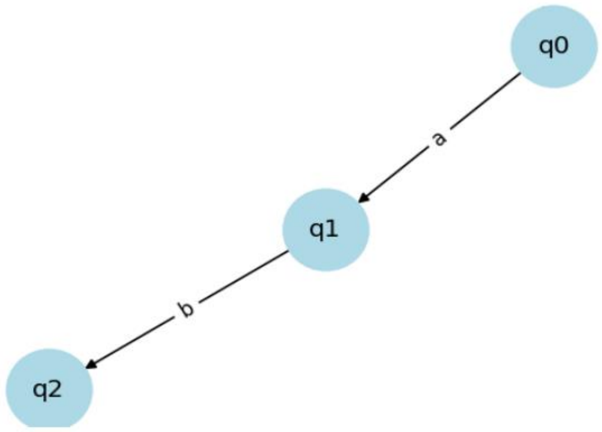
Final states "ab" Language:

$$L = \{w \mid w \text{ ends with ab}\}$$

States:

- q0 — start (loops on a & b)
- q1 — previous symbol was 'a'
- q2 — accept (matched 'ab')

Key Idea NFA gives the possibility of choosing between two states in the case of input 'a' → q1 and 'a' → q0 at the same time.



3.3.3 ε–NFA Design — (a|b)*abb

Language:

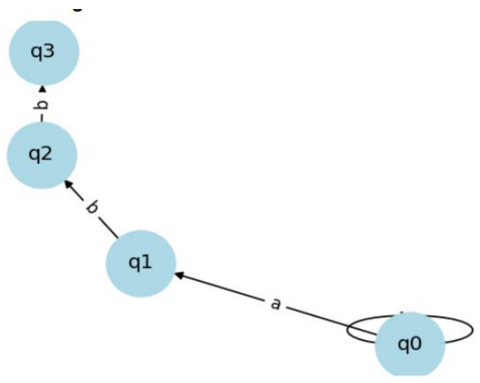
$$(a \mid b) * a\, b\, b$$

Design Logic

- Iterate over the prefix in q0 and accept any prefix.
- Look for a, b, b patterns in sequence.

States

- q0 — input (prefix)
- q1 — a detected
- q2 — ab detected
- q3 — abb detected (accept)



3.3.4 PDA design – $a^n b^n$

Language:

$$L=\{a^n b^n | n \geq 0\}$$

Stack logic

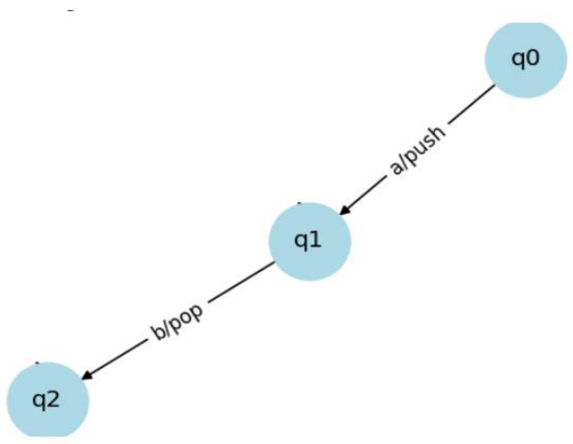
- For each a: **push A**
- For each b: **pop A**
- Accept only if stack returns to initial symbol Z

States

- q0 → q1 → q2
- Empty string accepted by default (n = 0)

Implementation

- Python list used as stack
- Errors for underflow or invalid order



3.3.5 Turing Machine Design — $a^n b^n c^n$

Language:

￼￼$a^n b^n c^n$￼

Design Algorithm

1. Scan for first 'a' → mark 'X'
2. Scan for first unmatched 'b' → mark 'Y'
3. Scan for first unmatched 'c' → mark 'Z'
4. Return head to start
5. Repeat until all input is marked
6. Final sweep to ensure order X → Y → Z

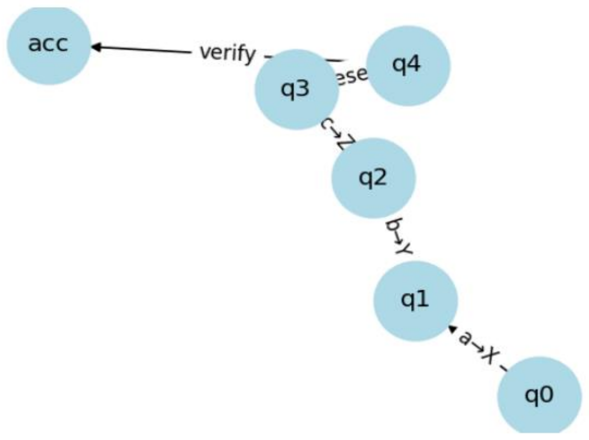
States

- q0, q1, q2, q3, q4
- q4_y, q4_z
- q_accept, q_reject

Implementation

- Tape = Python list
- Head movement fully simulated

- Step counter to prevent infinite loops
- Logging of every transition



3.4 Diagram Rendering with Active State Highlighting

Every automaton is represented by a corresponding state diagram:

- Drawn with networkx
- The logic for coloring the nodes is as follows:
  - Start state: Light blue
  - Accept states: Light green
  - Last active state after execution: Yellow with red border
  - Other states: White

The diagram is automatically refreshed in the following cases:

- When the automaton type changes
- When the user inputs a new value
- When a random input test is executed

3.5 Event Logging & History Management

The simulator maintains a history that is always visible and includes:

- All inputs tried
- Acceptance or rejection
- Detailed algorithm execution
- The specific automaton used
- Date and time

This is very useful to compare the outputs of different tests.

3.6 Summary of Enhancements Added

- The ultimate system has the following features:
- Support for manual user input
- Random generation of inputs for all automata
- Logs of execution in steps
- Monitoring of state sequences
- Final state highlighting on dynamic diagram
- A record of all the tests conducted
- Visual representation of automata diagrams for all five machines

4. Validation and Testing

The system includes both:

- Manual testing
- Automated random testing

Each automaton has built-in valid random string generators.

4.1 DFA Test Cases

INPUT	OUTPUT

0010	Accept
111	Accept
1010	Accept
000	Reject (3 zeros)

4.2 NFA Test Cases

INPUT	ENDS WITH ab?	RESULT
babab	Yes	Accept
baa	No	Reject
ab	Yes	Accept
Aab	Yes	Accept

4.3 ε–NFA Test Cases

INPUT	contains abb?	RESULT
abb	Yes	Accept
bababb	Yes	Accept
ab	No	reject

4.4 PDA Test Cases

INPUT	SHAPE	RESULT
aaabbb	a^3b^3	Accept
aab	a^2b	Reject
ab	ab	Accept

4.5 Turing Machine Test Cases

INPUT	Pattern	RESULT

aaabbbccc	n=3	Accept
aabbcc	n=2	Accept
ab c	n=1	Accept

5.Results and Discussion

- All automata operated properly in accordance with their mathematical specification.
- Step-by-step execution logs vividly illustrate internal transitions.
- State diagrams facilitate understanding of the machine's behavior.
- Non-determinism was dealt with correctly using sets.
- Stack and tape operations accurately portray PDA and TM respectively.

Automation	Language class	Memory	Deterministic	Power
DFA	Regular	Finite	Yes	Low
NFA	Regular	Finite	No	Low
ε–NFA	Regular	Finite	No	Low
PDA	Context free	Stack	Both	Medium
TM	Context-sensitive	Tape	Both	High

5.1 Challenges

- Ensuring Turing Machine halting
- Managing non-deterministic branching
- Coordinating active-state highlighting in diagrams
- Handling stack underflow in PDA

5.2 Learning Outcomes

This project demonstrates:

- How computational power increases from DFA → TM
- How language structure determines automaton design

- Implementation challenges for recursive and multi-symbol languages
- Importance of visualization and stepwise debugging

6. Conclusion and Future Work

A total of five automata have been successfully designed, implemented, tested, and analyzed in this project over three different language classes. Besides being an educational tool, the interactive simulator allows the user to analyze the automata in a very transparent manner by showing the input processing of each machine step by step.

Future Extensions

1. DFA minimization
2. Multi-tape Turing Machines
3. Grammar-to-automaton converters
4. Head/stack movement animation
5. Export-to-PDF for execution traces

7. References

1. **Hopcroft, J. E., Motwani, R., & Ullman, J. D.**
   Introduction to Automata Theory, Languages, and Computation.
   Addison-Wesley, 3rd Edition, 2006.
2. **Sipser, Michael.**
   Introduction to the Theory of Computation.
   Cengage Learning, 3rd Edition, 2012.
3. **Kozen, Dexter.**
   Automata and Computability.
   Springer, 1997.

4. **Lewis, Harry R., & Papadimitriou, Christos H.**
Elements of the Theory of Computation.
Prentice Hall, 1997.