Amrita Vishwa Vidyapeetham, Amaravati

Department of Computer Science and Engineering

Project Report

on

Automata-Based Smart Door Lock Control System

Submitted by

G. Mohnish Krishna Sai Kumar
Roll. No: AV.SC.U4CSE23013

K. Mokshagna
Roll. No: AV.SC.U4CSE22502

Under the Supervision of

Mr. Dontha Madhusudhana Rao
Assistant Professor (Sr. Grade)

Academic Year 2025 (CSE – A)

Aim

To design and implement an automata-based Smart Door Lock and Access Control System. The objective is to provide a secure mechanism for managing door access using a six PIN-based authentication method. The system ensures:

- Only authorized users with the correct PIN can unlock the door.
- Users are provided with an option to reset the PIN securely.
- Unauthorized access attempts are restricted and denied.

Introduction

The Door Lock Management System is a modern security solution that ensures controlled access through password authentication. Instead of relying on traditional mechanical locks, this system uses a 6-digit PIN code as the key for unlocking, providing greater flexibility and reliability. The system is designed to validate user input against a stored PIN, unlocking the door only when the correct sequence of digits is entered. If an incorrect PIN is entered, the system immediately rejects the input and transitions into an error state, preventing unauthorized access. To enhance security, the system also features limited attempts, PIN reset functionality, and an automated lockout after multiple failed entries. This makes the Door Lock Management System both practical and secure, closely resembling real-world digital lock mechanisms.

To model this system formally, we employ **context-free grammars** and a **Pushdown Automaton (PDA)**. The grammar defines the valid structure of the PIN sequence, while the PDA represents the state transitions involved in validating and accepting or rejecting an entered PIN. This approach demonstrates how concepts from formal language and automata theory can be applied to real-world security problems, ensuring correctness, reliability, and structured validation in system design.

Grammar:

In formal language theory, a **grammar** is a structured system used to define the strings of a language. Formally, a grammar is represented as a 4-tuple **(V, T, P, S)**, where:

- **V (Variables/Non-terminals):** These are symbols that can be replaced or expanded using production rules, e.g., {S, A, B, C…}.
- **T (Terminals):** These are the basic symbols of the language that appear in the final strings, e.g., {a, b}.
- **P (Productions/Rules):** A set of rules that specify how variables can be replaced by combinations of terminals and variables, e.g., $S \rightarrow aA \mid b$.
- **S (Start Symbol):** A special variable from which string generation begins.

Grammars are essential in **automata theory**, as they provide a blueprint for generating strings in a language, while **automata** are the computational machines that recognize or accept these strings.

Grammars are classified according to the **Chomsky Hierarchy**:

1. **Type 0 – Unrestricted Grammar:** Can generate any recursively enumerable language.
2. **Type 1 – Context-Sensitive Grammar (CSG):** Rules are of the form $\alpha A \beta \rightarrow \alpha \gamma \beta$, ensuring the length of the string does not decrease.
3. **Type 2 – Context-Free Grammar (CFG):** Rules have a single non-terminal on the left-hand side: $A \rightarrow \gamma$.
4. **Type 3 – Regular Grammar:** Rules are very restricted, e.g., $A \rightarrow aB$ or $A \rightarrow a$, suitable for defining regular languages.

Here in this, we use a **Context-Free Grammar** (CFG) to formally define valid PIN entry sequences for the smart door lock system. The CFG generates the precise structure required for successful authentication, ensuring only the correct PIN is accepted.

Grammar Steps:

S = Start symbol
Terminals: 5, 0, 2, 0, 1, 3 (the valid PIN digits in order)
Non-terminals: A, B, C, D, E, F (intermediate validation steps)
ε = Empty string (successful completion)

1. S → 5A
2. A → 0B
3. B → 2C
4. C → 0D
5. D → 1E
6. E → 3F
7. F → ε

Language:

Languages represent the problems that automata, grammars, or computational machines can recognize or solve.

- Let Σ be an alphabet.
- A string is a finite sequence of symbols from Σ.
- A language L over Σ is any set of such strings.

Types of Languages (Chomsky Hierarchy)

1. **Regular Languages:**
   These are the simplest class of languages generated by regular grammars. They are recognized by finite automata (DFA or NFA). The production rules are restricted such that a single non-terminal is replaced by either a terminal or a terminal followed by a non-terminal.
2. **Context-Free Languages (CFLs):**
   Generated by context-free grammars, these languages are recognized by pushdown automata (which use a stack). The production rules allow a single non-terminal to be replaced by a combination of terminals and/or non-terminals, enabling the description of nested or recursive structures.
3. **Context-Sensitive Languages (CSLs):**
   These languages are generated by context-sensitive grammars and are recognized by linear bounded automata. The production rules are dependent on the surrounding context, with constraints ensuring that the output string is never shorter than the input string.
4. **Recursively Enumerable Languages:**
   The most general class of languages, generated by unrestricted grammars and recognized by Turing machines. These languages include any that can be computed or enumerated by a general algorithm.

The language used in this project is a **Context-Free Language (CFL)**, generated by a context-free grammar and accepted by a pushdown automaton, suitable for modelling the smart door lock PIN validation system.

Automata Used

In our project, a Pushdown Automaton (PDA) is employed to model the authentication process of the smart door lock. Similar to the way states govern configurations in puzzles, each state in the PDA corresponds to a specific stage in the PIN validation sequence.

Formal Mathematical Steps of the PDA for PIN Validation

Given the PDA **M=(Q,Σ,Γ,δ,q0,Z0,F)** where:

- **Q** = {qstart,qlocked,q1,q2,q3,q4,q5,q6,qopen,qerror} (set of states)
- **Σ** = {5,0,2,1,3}(input alphabet, digits of PIN)
- **Γ** ={Z0}(stack alphabet, with base symbol)
- **q0** = qstart (start state)
- **Z0** (initial stack symbol)
- **F** = {qopen} (accepting state)

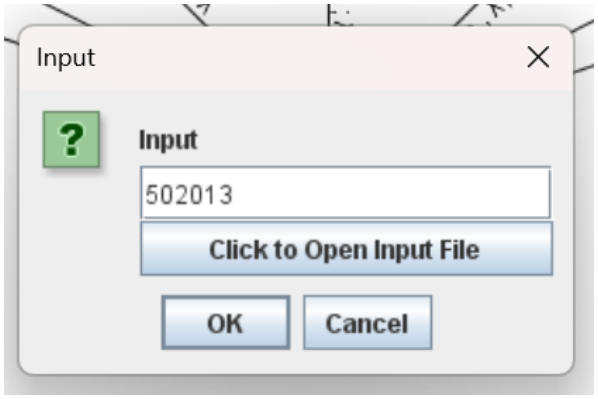
Step-by-Step State Transition

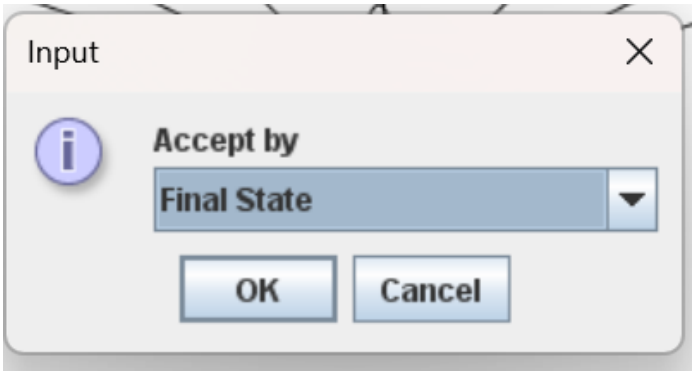
- **Step 1:** The system starts in the initial state **qstart**, where the stack is initialized with a special bottom marker.
- **Step 2:** Upon receiving the first digit 5, the PDA moves to state q1.
- **Step 3:** Inputting the next digit 0 causes a transition to state q2.
- **Step 4:** The third digit 2 is verified as the automaton transitions to state q3.
- **Step 5:** The fourth digit 0 advances the PDA to state q4.
- **Step 6:** For the fifth digit 1, the PDA enters state q5.
- **Step 7:** After the sixth digit 3, the PDA reaches state q6.
- **Step 8:** Finally, the stack symbol is popped, and the PDA transitions into the accepting **qopen** state, confirming the PIN is valid and unlocking the door.

If any digit does not match the stored PIN at any step, the PDA transitions to an error state **qerror**, rejecting the input and denying access.

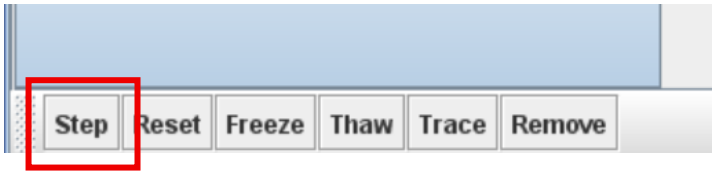
In order to start the step-by-step movement of State:

We need to start the testing on the top of JFF App Click the **Input Option** inside that click on **Step by State (**in simple you can use simple shortcut **"Ctrl + Shift + R")** then you will open the input box like this later click ok by selecting the **final state.**





After clicking okay below the jff chart you will find the box with q_start where the top of stack we will have the **no stack symbols** as there are **no inputs** this is where step by step states will be checked in order to move the next state, we need to click the step button which marked with red box.



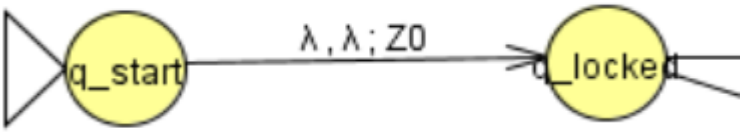
Detailed Explanation of PDA States & Transitions for PIN Validation

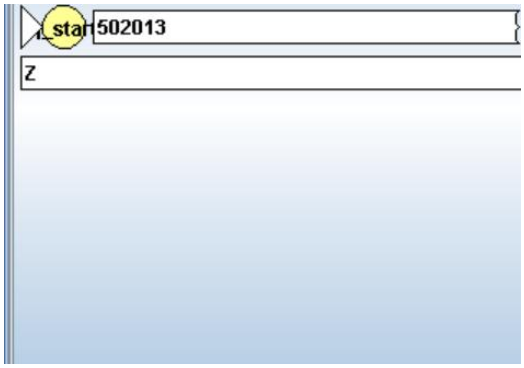
Step 1: System Initialization and Stack Setup at qstart

- **Input:** No input symbol consumed (ε transition).
- **Stack Operation:** Push symbol "Z0" onto the stack as the base marker.
- **Description:** This initializes the stack to mark the bottom before PIN entry begins. The stack starts with base marker "Z0", no input read.
- **Reason for transition:** At start state with empty input and empty stack, push Z0 onto stack and move to locked state.

$$\delta(qstart,\varepsilon,\varepsilon)=\{(qlocked,Z0)\}$$

- **Current Stack:** [Z0]





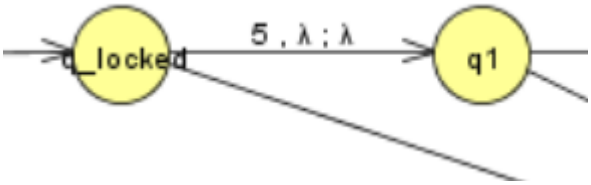
- **Next State:** Initially, at **state q_start**, on an empty input with an empty stack, the PDA pushes the base symbol "Z0" and moves to **state q_locked**, preparing for PIN entry.

Step 2: Transition from qlocked (State 1) to q1 (State 2)

- **Input:** The input symbol '5' is read.
- **Stack Operation:** No change to the stack (top remains "Z0").
- **Description:** The PDA recognizes the first correct digit of the PIN ("5") and moves forward in the sequence.
- **Invalid Input Handling:** At **state q_locked**, on reading input '5' with stack top "Z0", the PDA keeps the stack unchanged and moves to **state q1**, confirming the first digit of the PIN.

$$\delta(qlocked,5,Z0)=\{(q1,Z0)\}$$

- **Current Stack:** [Z0]



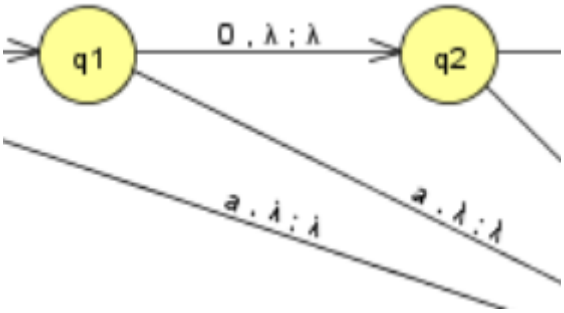
- **Next State:** On reading input '5' with stack top 'Z0' at **state q_locked**, the PDA does not modify the stack and moves to **state q1** to validate the second digit.

Step 3: Transition from q1 (State 2) to q2 (State 3)

- **Input:** The input symbol '0' is read.
- **Stack Operation:** No stack manipulation; the "Z0" remains at the bottom of the stack.
- **Description:** After validating the first digit, the PDA now verifies the second digit '0'. Successful reading of this digit moves the system one step closer to complete PIN validation.
- **Invalid Input Handling:** At **state q1**, on reading input '0' with stack top "Z0", the PDA performs no stack modification and transitions to **state q2**, validating the second digit.

$$\delta(q1,0,Z0)=\{(q2,Z0)\}$$

- **Current Stack:** [Z0]



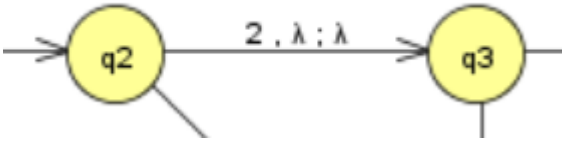
- **Next State:** On reading input '0' with stack unchanged at **state q1**, the PDA moves to **state q2**, confirming the second digit.

Step 4: Transition from q2 (State 3) to q3 (State 4)

- **Input:** The input symbol '2' is read.
- **Stack Operation:** No changes are made to the stack; the base symbol "Z0" remains intact.
- **Description:** The PDA verifies the third digit of the PIN. Upon successful reading, it advances to the next state.
- **Invalid Input Handling:** At **state q2**, on reading input '2' with stack top "Z0", the PDA leaves the stack unchanged and proceeds to **state q3**, confirming the third digit.

$$\delta(q2,2,Z0)=\{(q3,Z0)\}$$

- **Current Stack:** [Z0]



- **Next State:** On reading input '2' with stack unchanged at **state q2**, the PDA progresses to **state q3** to validate the third digit.

Step 5: Transition from q3 (State 4) to q4 (State 5)

- **Input:** The input symbol '0' is read.
- **Stack Operation:** Stack remains unchanged; "Z0" remains at the bottom.
- **Description:** This state confirms the fourth digit in the PIN sequence. A successful match results in moving to the next validation state.
- **Invalid Input Handling:** At **state q3**, on reading input '0' with stack top "Z0", the PDA keeps the stack intact and moves to **state q4**, validating the fourth digit.

$$\delta(q3,0,Z0)=\{(q4,Z0)\}$$

- **Current Stack:** [Z0]

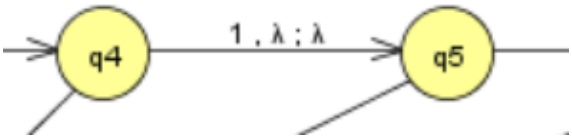
- **Next State:** On reading input '0' with stack unchanged at **state q3**, the PDA transitions to **state q4** to check the fourth digit.

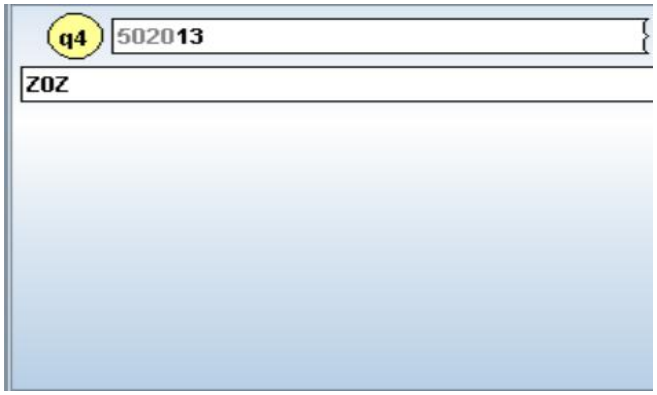
Step 6: Transition from q4(State 5) to q5 (State 6)

- **Input:** The input symbol '1' is read.
- **Stack Operation:** The stack remains unchanged with "Z0" at the bottom.
- **Description:** The PDA verifies the fifth digit of the PIN. If the input is correct, it transitions to the next state for the final digit validation.
- **Invalid Input Handling:** At **state q4**, on reading input '1' with stack top "Z0", the PDA does not modify the stack and proceeds to **state q5**, confirming the fifth digit.

$$\delta(q4,1,Z0)=\{(q5,Z0)\}$$

- **Current Stack:** Z0





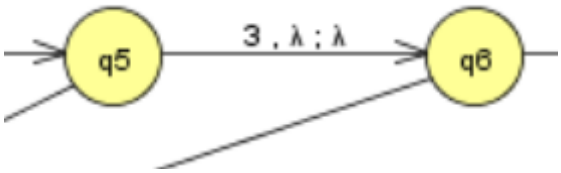
- **Next State:** On reading input '1' with stack unchanged at **state q4**, the PDA advances to **state q5** for the fifth digit validation.

Step 7: Transition from q5 (State 6) to q6 (State 7)

- **Input:** The input symbol '3' is read.
- **Stack Operation:** The stack remains unchanged with the bottom symbol "Z0".
- **Description:** This step validates the sixth and last digit of the PIN sequence. A correct input leads to the final check before acceptance.
- **Invalid Input Handling:** At state q5, on reading input '3' with stack top "Z0", the PDA maintains the stack unchanged and transitions to state q6, completing the digit verification.

$$\delta(q5,3,Z0)=\{(q6,Z0)\}$$

- **Current Stack:** Z0



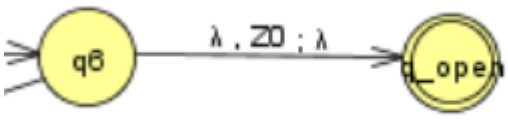
- **Next State:** On reading input '3' with stack unchanged at **state q5**, the PDA moves to **state q6** towards final acceptance.

Step 8: Transition from q6 (State 7) to qopen (State 8)

- **Input:** No input symbol is consumed in this ε-transition.
- **Stack Operation:** The PDA pops the bottom stack symbol "Z0", indicating completion of PIN input.
- **Description:** This marks the acceptance of the complete PIN sequence. The PDA transitions to the accepting (open) state, signalling successful validation.
- **Invalid Input Handling:** At **state q6**, on ε (no input) with stack top "Z0", the PDA pops the base symbol from the stack and moves to **state q_open**, accepting the input and unlocking the door.

$$\delta(q6,\ \varepsilon,\ Z0)=\{(qopen\ ,\varepsilon)\}$$

- **Current Stack:** (Stack emptied)

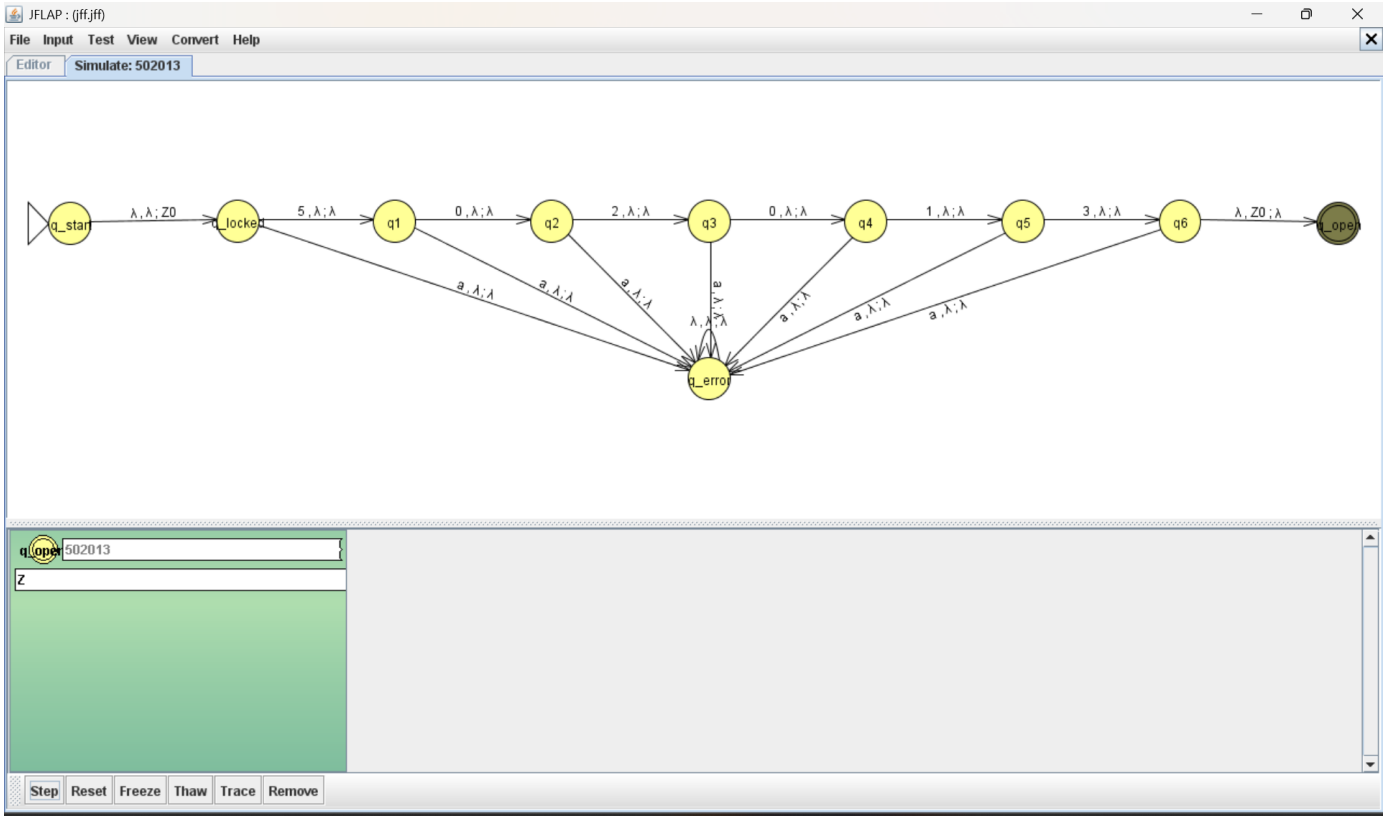


- **Next State:** With no input and stack top 'Z0' at **state q6**, the PDA pops the base symbol from the stack and transitions into **state q_open**, confirming the correct PIN entry.

How to Conclude that our automata has reached final state?

To conclude that the **PDA automata** has reached the **final state** in JFLAP for the PIN validation system, two main conditions must be fulfilled:

- The **input sequence** must be exactly **"502013"**—all 6 digits must be entered in the correct order and fully consumed.
- Upon processing the last digit, the **stack** must have no remaining symbols (i.e., the base symbol **"Z0"** must be popped), and the automaton must move to the **accepting state q_open**.



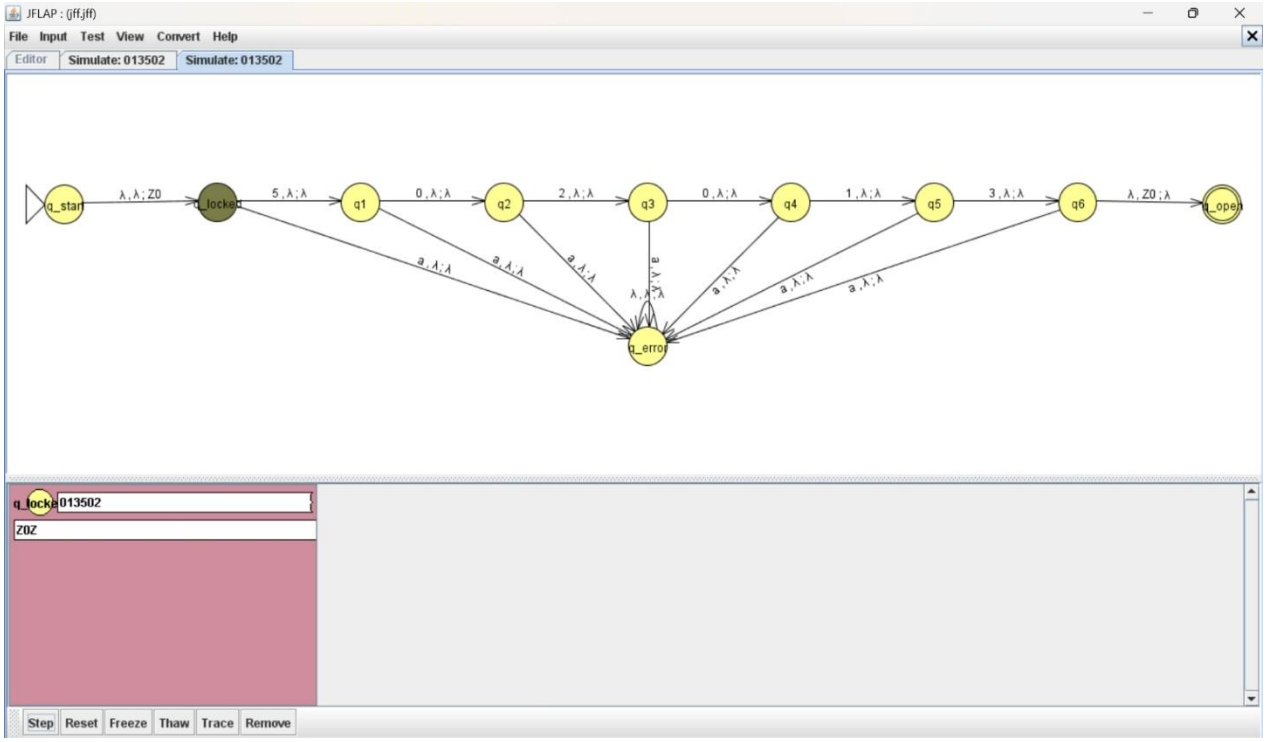
If both the **input is exhausted** and the PDA is in the **final accepting state** with an **empty stack**, JFLAP visually indicates this success (the state turns **green**), confirming that the model has accepted the string and the lock would open. Any violation—incorrect PIN, unconsumed input, or stack not empty—results in **rejection**.

How does our Automata get reject ?

The automata implemented for PIN validation will reject the input in the following two scenarios:

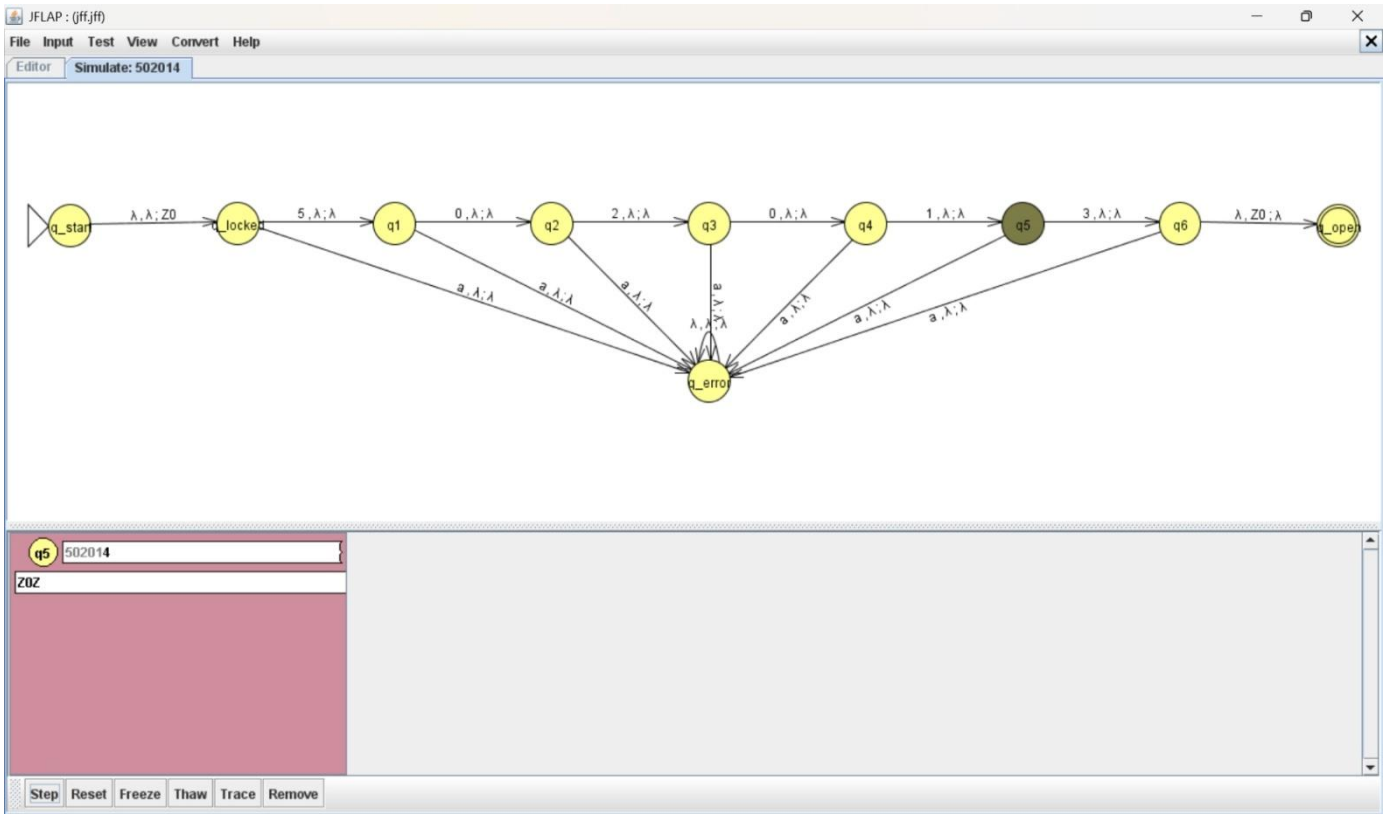
Case 1: Wrong Starting Digit

- If the first digit is incorrect (not '5'), the PDA cannot leave state **q_locked**. Since no transition exists for an invalid starting digit, the PDA halts immediately at this state, and the input is rejected without any progress through further states.
- *Example:* If you try the PIN '802013' instead of '502013', the automaton stays in **q_locked** after reading '8'. JFLAP will display no further transitions, and the simulation visually stops at the starting locked state. The door remains locked, and no stack change occurs. This proves the automaton verifies the PIN right from the first entry.



Case 2: Correct Start but Wrong Digit in Middle

- If the PIN starts correctly (first digit '5'), but any later digit is wrong (e.g., entering '502113'), the PDA moves through the expected states until it reads the wrong input. At that moment, it transitions directly to **state q_error**.
- *Example:* Entering '502113' causes the PDA to run through **q_locked → q1 → q2 → q3 → q4**, but when the fifth digit '1' is read where '0' was expected, the transition leads instantly to **q_error**. JFLAP shows the simulation ending in the red error state with the remaining input unused. The stack remains unchanged except for the base symbol. This demonstrates the automaton's strict check at every step and instant halt on any error in the sequence.



How is the interfacing between Python code & the JFLAP .jff file handled?

- ✓ The user is presented with a menu offering three options:
  1. Enter the PIN to unlock the door.
  2. Reset the PIN securely.
  3. Exit the application.

```
--- 🔒 Smart Door Lock System ---
1. Enter PIN
2. Reset PIN
3. Exit
Choose option:
```

- ✓ **Enter the PIN to unlock the door:**
  1. The system prompts the user to input a 6-digit PIN.
  2. If the entered PIN matches the stored PIN, the system prints:
     "✅ Correct PIN - Door Unlocked"

```
--- 🔒 Smart Door Lock System ---
1. Enter PIN
2. Reset PIN
3. Exit
Choose option: 1
Enter 6-digit PIN: 502013
✅ Correct PIN - Door Unlocked
```

  3. If the PIN is incorrect, it increments the failed attempt count and prints:
     "❌ Wrong PIN - Access Denied (attempts/maximum attempts)"

```
--- 🔒 Smart Door Lock System ---
1. Enter PIN
2. Reset PIN
3. Exit
Choose option: 1
Enter 6-digit PIN: 013502
❌ Wrong PIN - Access Denied (1/3)
```

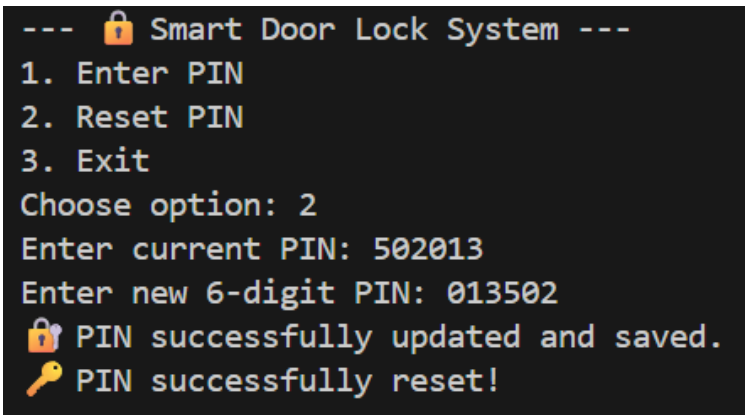
  4. After 3 failed attempts, the system shows:
     "🚨 Security Alert: System Locked!" and denies further access until reset.

```
--- 🔒 Smart Door Lock System ---
1. Enter PIN
2. Reset PIN
3. Exit
Choose option: 1
Enter 6-digit PIN: 241412
❌ Wrong PIN - Access Denied (3/3)
🚨 Security Alert: System Locked!
```

✓ **Reset the PIN securely:**

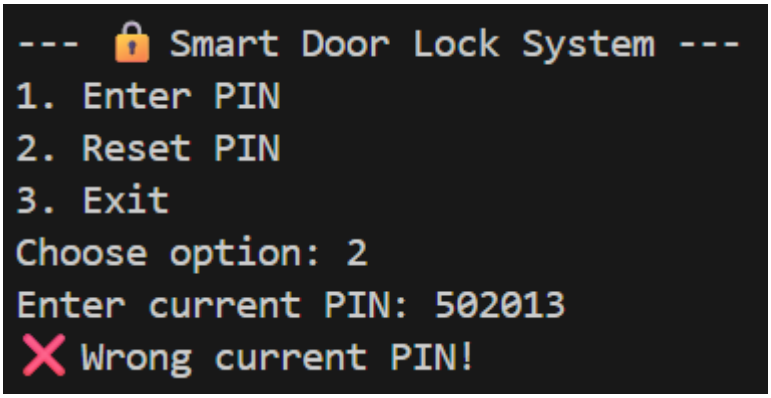
1. The user is asked to enter the current PIN for verification.
2. Upon successful verification, the user can input a new 6-digit PIN.
3. If valid, the PIN is updated inside the .jff file and the program states:

"🔑 PIN successfully reset!"



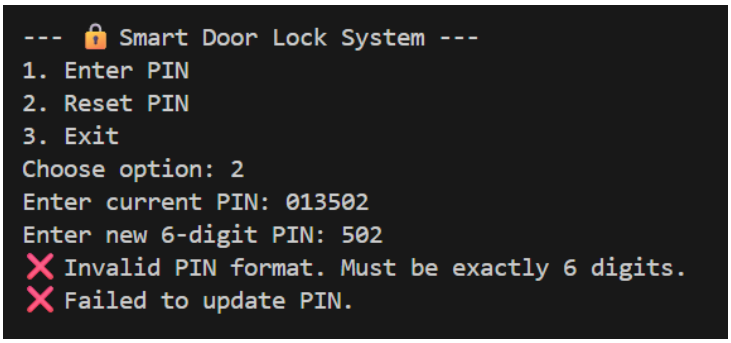
4. If the current PIN entered is incorrect, the system displays:

"❌ Wrong current PIN!"



5. If the new PIN format is invalid (not 6 digits), the system shows:

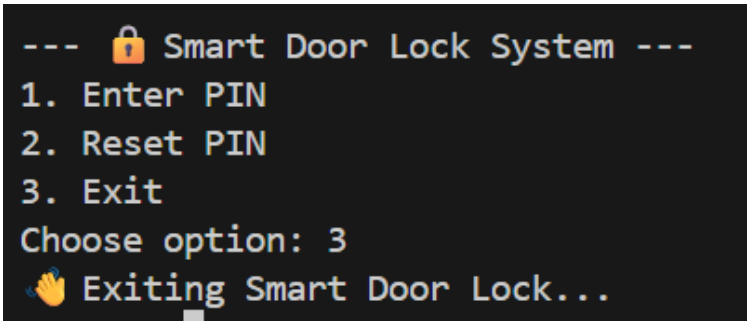
"❌ Invalid PIN format. Must be exactly 6 digits."



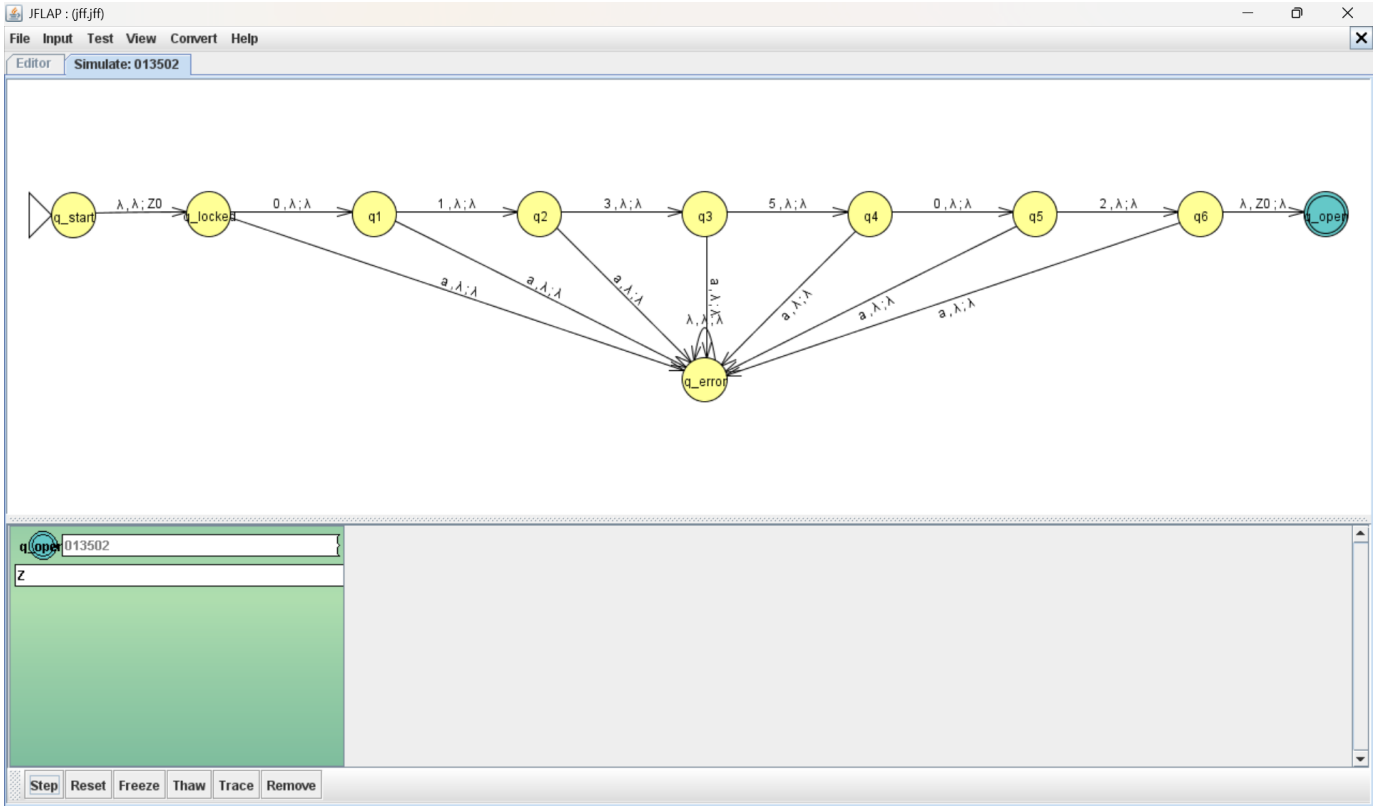
✓ **Exit the application:**

1. The program displays:

"👋 Exiting Smart Door Lock..."



After updating the PIN, the changes are automatically reflected in the **JFLAP** .jff file by modifying the **PDA** transitions to match the new PIN digits. This ensures that the automaton's behavior stays synchronized with the current valid PIN stored in the system.



Python Code:

```
import xml.etree.ElementTree as ET
import shutil
import os

JFF_FILE = r"E:\jff.jff"
BACKUP_FILE = JFF_FILE + ".bak"

def backup_jff():
    """Create a backup copy of the current .jff file."""
    try:
        shutil.copy2(JFF_FILE, BACKUP_FILE)
    except Exception as e:
        print(f"⚠️ Backup failed: {e}")

def get_current_pin():
    """
    Extract the stored PIN from JFLAP transitions in correct PDA order.
    Expected transitions: 1→2, 2→3, 3→4, 4→5, 5→6, 6→7
    """
    tree = ET.parse(JFF_FILE)
    automaton = tree.getroot().find("automaton")
    pin_digits = []
    expected_pairs = [("1", "2"), ("2", "3"), ("3", "4"), ("4", "5"), ("5", "6"), ("6", "7")]

    for (frm, to) in expected_pairs:
        digit = None
        for trans in automaton.findall("transition"):
```

```
            from_state = (trans.find("from").text or "").strip()
            to_state = (trans.find("to").text or "").strip()
            read = (trans.find("read").text or "").strip() if trans.find("read") is not None else ""
            if from_state == frm and to_state == to and read.isdigit():
                digit = read
                break
        if digit:
            pin_digits.append(digit)

    return "".join(pin_digits)

def update_jff_pin(new_pin):
    """Update the JFLAP automaton transitions with the new 6-digit PIN."""
    new_pin = new_pin.strip()
    if len(new_pin) != 6 or not new_pin.isdigit():
        print("❌ Invalid PIN format. Must be exactly 6 digits.")
        return False
    backup_jff()
    tree = ET.parse(JFF_FILE)
    root = tree.getroot()
    automaton = root.find("automaton")

    # Remove old numeric transitions (1–6)
    for trans in list(automaton.findall("transition")):
        frm = trans.find("from")
        read = trans.find("read")
        if frm is None or read is None:
            continue
        from_state = (frm.text or "").strip()
        read_val = (read.text or "").strip()
        if from_state.isdigit() and 1 <= int(from_state) <= 6 and read_val.isdigit():
            automaton.remove(trans)

    # Add new transitions for each digit
    states = ["1", "2", "3", "4", "5", "6", "7"]
    for i in range(6):
        t = ET.SubElement(automaton, "transition")
        ET.SubElement(t, "from").text = states[i]
        ET.SubElement(t, "to").text = states[i + 1]
        ET.SubElement(t, "read").text = new_pin[i]
        ET.SubElement(t, "pop").text = ""
        ET.SubElement(t, "push").text = ""
    tree.write(JFF_FILE, encoding="UTF-8", xml_declaration=True)
    print("🔐 PIN successfully updated and saved.")
    return True

def verify_pin_jff(pin):
    """Simulate JFLAP PDA transitions to verify the entered PIN."""
    tree = ET.parse(JFF_FILE)
    automaton = tree.getroot().find("automaton")
    current_state = "1"
    for digit in pin:
        found_transition = False
        for trans in automaton.findall("transition"):
            from_state = (trans.find("from").text or "").strip()
```

```
            to_state = (trans.find("to").text or "").strip()
            read_val = (trans.find("read").text or "").strip() if trans.find("read") is not None else ""
            if from_state == current_state and read_val == digit:
                current_state = to_state
                found_transition = True
                break
        if not found_transition:
            return False
    return current_state == "7"

def door_lock():
    """Main interactive door lock interface."""
    attempts = 0
    max_attempts = 3

    while True:
        stored_pin = get_current_pin()
        print("\n--- 🔒 Smart Door Lock System ---")
        print("1. Enter PIN")
        print("2. Reset PIN")
        print("3. Exit")

        choice = input("Choose option: ").strip()
        if choice == "1":
            if attempts >= max_attempts:
                print("🚨 Too many failed attempts. Access locked!")
                continue
            user_pin = input("Enter 6-digit PIN: ").strip()
            if verify_pin_jff(user_pin):
                print("✅ Correct PIN - Door Unlocked")
                attempts = 0
            else:
                attempts += 1
                print(f"❌ Wrong PIN - Access Denied ({attempts}/{max_attempts})")
                if attempts >= max_attempts:
                    print("🚨 Security Alert: System Locked!")

        elif choice == "2":
            old_pin = input("Enter current PIN: ").strip()
            if old_pin == stored_pin and stored_pin != "":
                new_pin = input("Enter new 6-digit PIN: ").strip()
                if update_jff_pin(new_pin):
                    print("🔑 PIN successfully reset!")
                    attempts = 0
                else:
                    print("❌ Failed to update PIN.")
            else:
                print("❌ Wrong current PIN!")

        elif choice == "3":
            print("👋 Exiting Smart Door Lock...")
            break

        else:
```

```
            print("❌ Invalid option! Please choose 1, 2, or 3.")

if __name__ == "__main__":
    if not os.path.exists(JFF_FILE):
        print(f"Error: JFF file not found at {JFF_FILE}")
    else:
        door_lock()
```

Problematic way:

Starting configuration: (qstart,w,ε)

Processing input w=502013 PDA performs the following transitions:

1. (qstart, 502013, ε) = (qlocked, 502013, Z0)
2. (qlocked, 502013,Z0) = (q1, 02013, Z0)
3. (q1, 02013, Z0) = (q2, 2013, Z0)
4. (q2, 2013, Z0) = (q3, 013, Z0)
5. (q3, 013, Z0) = (q4, 13, Z0)
6. (q4, 13, Z0) = (q5, 3, Z0)
7. (q5, 3, Z0) = (q6, ε, Z0)
8. (q6, ε, Z0) = (qopen, ε, ε)

Since the PDA ends in an accepting state qopen with empty input and empty stack, the input is accepted and the door will be opened safely.

Conclusion

This project successfully implemented an **Automata-Based Smart Door Lock Control System** using a **Pushdown Automaton (PDA)** designed through **Context-Free Grammar** principles for PIN validation. The system was constructed and tested using **JFLAP**, demonstrating deterministic state transitions across ten states that we used with stack-based memory management using the top element "Z0". The PDA follows strict validation of the PIN "**502013**", immediately rejecting invalid inputs through error state transitions, which ensuring security. The mathematical approach using formal transition functions guarantees that behavior of the automata is correct and the output process accurate. Testing automata with the system correctly accepts the valid PIN while rejecting any incorrect sequences at the first wrong digit or in b/w the process are leading to stop the automata. The project helps to establishes a real time prototype using the specified automata like **PDA** and create a secure & reliable system based on the **grammar, automata, language** that we implement in it.