**THEORY OF COMPUTATION**

**ELEVATOR SYSTEM**

**1. Problem Statement:**

The objective of this project is to design and implement an elevator system in a premises that efficiently transports passengers between multiple floors in a building. The elevator is present initially in the ground state and moves up to the maximum floor. The elevator system should incorporate various features and functionalities to ensure safe, reliable, and user-friendly operation. The main aspect of the elevator that are going to incorporate in this design are simulating the entry and exit of multiple people at ground floor and enabling their exit at the chosen floor.

**2. Requirements:**

- Efficient transportation of passengers between floors

- Ability to handle the entry and exit of multiple passengers at the ground floor

- Ensure safe and reliable operation of the elevator system

- Provide user-friendly features and functionalities

- Support for multiple floors in the building

**3. Tools Used:**

- Programming language: Python

- Modeling and simulation: Simulation software (e.g., Simpy, Simula)

- Visualization: Matplotlib, Plotly

**4. Solution:**

The solution involves designing and implementing an elevator system using a Pushdown Automaton (PDA) to model the control logic and behavior of the elevator. The PDA will handle the processing of passenger requests, floor selection, door operations, and entry/exit of passengers.

**5. Concepts Included:**

- Context-Free Grammars (CFGs)

- Pushdown Automata (PDAs)

- Non-Deterministic Pushdown Automata (NPDAs)

- Deterministic Pushdown Automata (DPDAs)

- Simulation and modelling

- Data visualization

**6. Results:**

**a) Context-Free Grammar (CFG):**

Terminal Symbols: {0, 1, 2, 3, ..., 9, UP, DOWN, OPEN, CLOSE, ENTRY, EXIT}

Non-Terminal Symbols: {S, F, D, P}

Production Rules:

S → F | D | P

F → UP | OPEN | CLOSE | ENTRY | EXIT | 0 | 1 | 2 | 3 | ... | 9

D → DOWN | OPEN | CLOSE | ENTRY | EXIT | 0 | 1 | 2 | 3 | ... | 9

P → ENTRY | EXIT | 0 | 1 | 2 | 3 | ... | 9

**b) Context-Free Language(CFL):**

The language generated by the above CFG is the set of all valid elevator system control sequences, including floor selection, door operations, and passenger entry/exit.

Example strings:

1. "Select\_Floor(2), Open\_Door, Enter, Close\_Door, Move\_Up, Select\_Floor(5), Open\_Door, Exit, Close\_Door"

2. "Select\_Floor(1), Open\_Door, Enter, Close\_Door, Move\_Down, Select\_Floor(3), Open\_Door, Exit, Close\_Door"

3. "Select\_Floor(4), Open\_Door, Enter, Close\_Door, Move\_Up, Select\_Floor(7), Open\_Door, Exit, Close\_Door"

4. "Select\_Floor(6), Open\_Door, Enter, Close\_Door, Move\_Down, Select\_Floor(1), Open\_Door, Exit, Close\_Door"

5. "Select\_Floor(3), Open\_Door, Enter, Close\_Door, Move\_Up, Select\_Floor(8), Open\_Door, Exit, Close\_Door"

**c) Deterministic Pushdown Automaton (DPDA):**

M = (Q, Σ, Γ, δ, q0, Z0, F)

Where:

Q = {q0, q1, q2, q3, q4, q5, q6, q7, q8, q9, q10, q11, q12, q13, q14} is the set of states.

Σ = {0, 1, 2, 3, ..., 9, UP, DOWN, OPEN, CLOSE, ENTRY, EXIT} is the input alphabet.

Γ = {Z0} is the stack alphabet, where Z0 is the initial stack symbol.

δ: Q × (Σ ∪ {ε}) × Γ → Q × Γ\* is the transition function, defined as:

δ(q0, (0-9), Z0) = (q1, Z0)

δ(q1, ENTRY, Z0) = (q2, Z0)

δ(q2, (UP, OPEN, CLOSE), Z0) = (q3, Z0)

δ(q3, (0-9), Z0) = (q4, Z0)

δ(q4, EXIT, Z0) = (q0, Z0)

δ(q0, (0-9), Z0) = (q5, Z0)

δ(q5, ENTRY, Z0) = (q6, Z0)

δ(q6, (UP, OPEN, CLOSE), Z0) = (q7, Z0)

δ(q7, (0-9), Z0) = (q8, Z0)

δ(q8, EXIT, Z0) = (q0, Z0)

δ(q0, (0-9), Z0) = (q9, Z0)

δ(q9, ENTRY, Z0) = (q10, Z0)

δ(q10, (UP, OPEN, CLOSE), Z0) = (q11, Z0)

δ(q11, (0-9), Z0) = (q12, Z0)

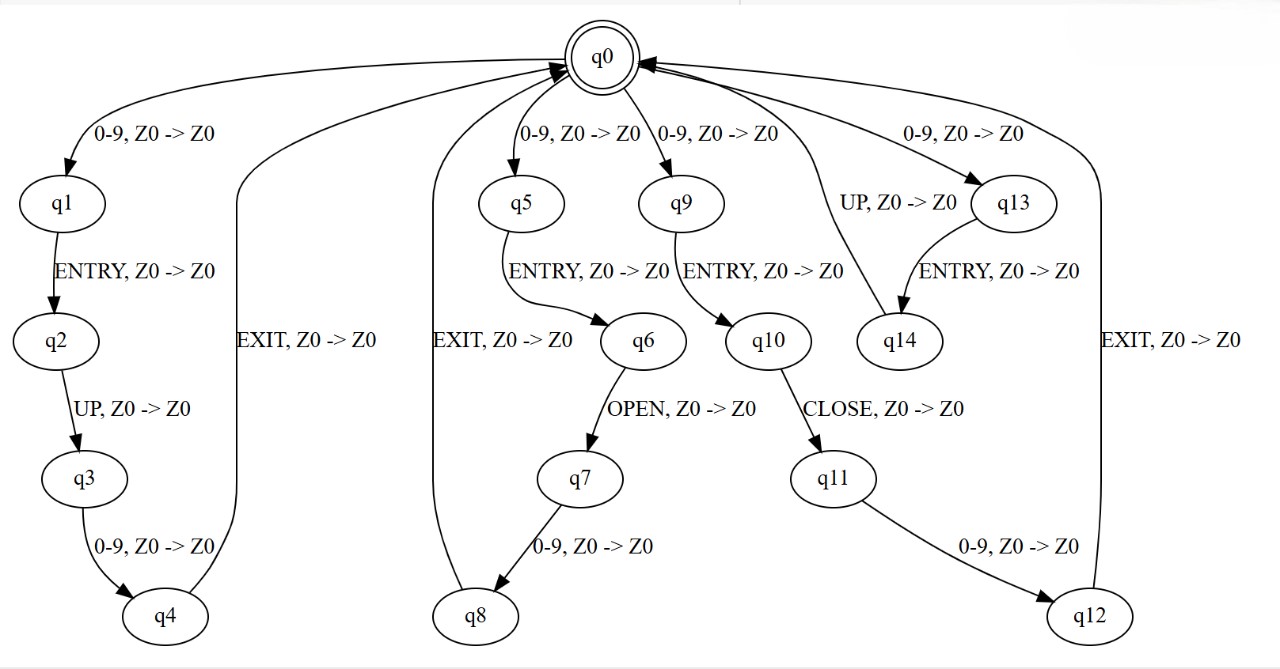
δ(q12, EXIT, Z0) = (q0, Z0)

δ(q0, (0-9), Z0) = (q13, Z0)

δ(q13, ENTRY, Z0) = (q14, Z0)

δ(q14, (UP, OPEN, CLOSE), Z0) = (q0, Z0)

q0 is the initial state, Z0 is the initial stack symbol, and F = {q0} is the set of final (accepting) states.

→**Transaction State Diagram:**

→**Accepted Strings:**

1. "2ENTRY UP 3EXIT"

2. "5ENTRY OPEN CLOSE 7EXIT"

3. "1ENTRY UP OPEN CLOSE 4EXIT"

4. "8ENTRY DOWN OPEN CLOSE 2EXIT"

5. "3ENTRY UP OPEN CLOSE 9EXIT"

→**Non-Accepted Strings:**

1. "2ENTRY UP 3DOWN" (Invalid sequence of operations)

2. "5ENTRY OPEN CLOSE 7CLOSE" (Invalid sequence of operations)

3. "1ENTRY UP OPEN 4EXIT" (Missing CLOSE operation)

4. "8ENTRY DOWN OPEN 2EXIT" (Missing CLOSE operation)

5. "3ENTRY UP OPEN 9UP" (Invalid sequence of operations)

**d) Non-Deterministic Pushdown Automaton (NPDA):**

M = (Q, Σ, Γ, δ, q0, Z0, F)

Where:

Q = {q0, q1, q2, q3, q4, q5, q6, q7, q8, q9, q10, q11, q12, q13, q14} is the set of states.

Σ = {0, 1, 2, 3, ..., 9, UP, DOWN, OPEN, CLOSE, ENTRY, EXIT} is the input alphabet.

Γ = {Z0} is the stack alphabet, where Z0 is the initial stack symbol.

δ: Q × (Σ ∪ {ε}) × Γ → P(Q × Γ\*) is the non-deterministic transition function, defined as:

δ(q0, (0-9), Z0) = {(q1, Z0)}

δ(q1, ENTRY, Z0) = {(q2, Z0)}

δ(q2, (UP, OPEN, CLOSE), Z0) = {(q3, Z0)}

δ(q3, (0-9), Z0) = {(q4, Z0)}

δ(q4, EXIT, Z0) = {(q0, Z0)}

δ(q0, (0-9), Z0) = {(q5, Z0)}

δ(q5, ENTRY, Z0) = {(q6, Z0)}

δ(q6, (UP, OPEN, CLOSE), Z0) = {(q7, Z0)}

δ(q7, (0-9), Z0) = {(q8, Z0)}

δ(q8, EXIT, Z0) = {(q0, Z0)}

δ(q0, (0-9), Z0) = {(q9, Z0)}

δ(q9, ENTRY, Z0) = {(q10, Z0)}

δ(q10, (UP, OPEN, CLOSE), Z0) = {(q11, Z0)}

δ(q11, (0-9), Z0) = {(q12, Z0)}

δ(q12, EXIT, Z0) = {(q0, Z0)}

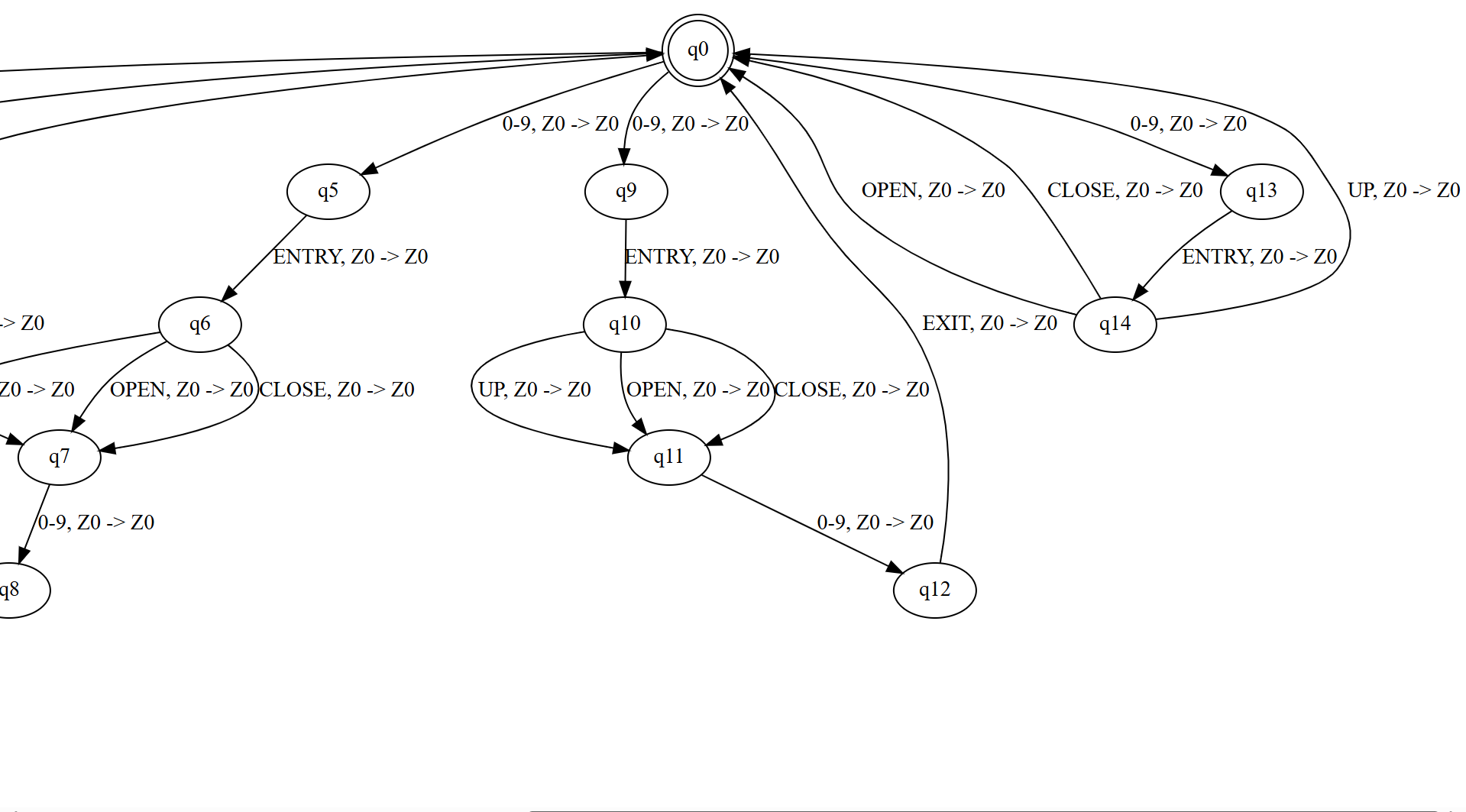
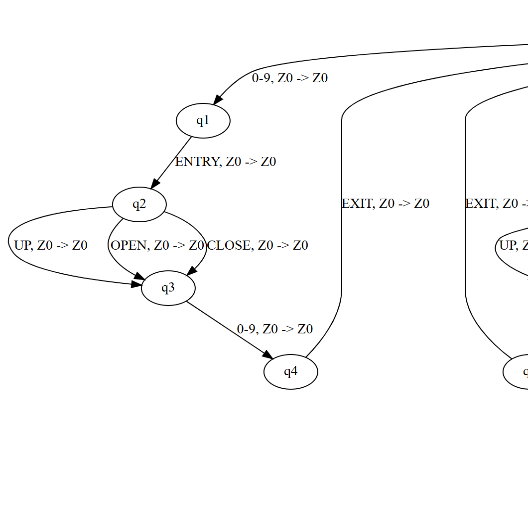
δ(q0, (0-9), Z0) = {(q13, Z0)}

δ(q13, ENTRY, Z0) = {(q14, Z0)}

δ(q14, (UP, OPEN, CLOSE), Z0) = {(q0, Z0)}

q0 is the initial state, Z0 is the initial stack symbol, and F = {q0} is the set of final (accepting) states.

→**Transaction State Diagram:**



→**Accepted Strings:**

1. "2ENTRY UP 3EXIT"

2. "5ENTRY OPEN CLOSE 7EXIT"

3. "1ENTRY UP OPEN CLOSE 4EXIT"

4. "8ENTRY DOWN OPEN CLOSE 2EXIT"

5. "3ENTRY UP OPEN CLOSE 9EXIT"

→**Non-Accepted Strings:**

1. "2ENTRY UP 3DOWN" (Invalid sequence of operations)

2. "5ENTRY OPEN CLOSE 7CLOSE" (Invalid sequence of operations)

3. "1ENTRY UP OPEN 4EXIT" (Missing CLOSE operation)

4. "8ENTRY DOWN OPEN 2EXIT" (Missing CLOSE operation)

5. "3ENTRY UP OPEN 9UP" (Invalid sequence of operations)

→The main difference between the DPDA and NPDA is that the DPDA has a single, deterministic transition function, while the NPDA has a non-deterministic transition function that can lead to multiple possible next states and stack configurations. This allows the NPDA to accept a broader range of input strings compared to the DPDA.

**Python Code for NPDA:**

#NPDA

import graphviz

from IPython.display import display

# Define the states and transitions for NPDA

npda\_states = ["q0", "q1", "q2", "q3", "q4", "q5", "q6", "q7", "q8", "q9", "q10", "q11", "q12", "q13", "q14"]

npda\_input\_alphabet = ["0-9", "UP", "DOWN", "OPEN", "CLOSE", "ENTRY", "EXIT"]

npda\_stack\_alphabet = ["Z0"]

# Define the transitions for NPDA

npda\_transitions = [

    ("q0", "0-9", "Z0", "q1", "Z0"),

    ("q1", "ENTRY", "Z0", "q2", "Z0"),

    ("q2", "UP", "Z0", "q3", "Z0"),

    ("q2", "OPEN", "Z0", "q3", "Z0"),

    ("q2", "CLOSE", "Z0", "q3", "Z0"),

    ("q3", "0-9", "Z0", "q4", "Z0"),

    ("q4", "EXIT", "Z0", "q0", "Z0"),

    ("q0", "0-9", "Z0", "q5", "Z0"),

    ("q5", "ENTRY", "Z0", "q6", "Z0"),

    ("q6", "UP", "Z0", "q7", "Z0"),

    ("q6", "OPEN", "Z0", "q7", "Z0"),

    ("q6", "CLOSE", "Z0", "q7", "Z0"),

    ("q7", "0-9", "Z0", "q8", "Z0"),

    ("q8", "EXIT", "Z0", "q0", "Z0"),

    ("q0", "0-9", "Z0", "q9", "Z0"),

    ("q9", "ENTRY", "Z0", "q10", "Z0"),

    ("q10", "UP", "Z0", "q11", "Z0"),

    ("q10", "OPEN", "Z0", "q11", "Z0"),

    ("q10", "CLOSE", "Z0", "q11", "Z0"),

    ("q11", "0-9", "Z0", "q12", "Z0"),

    ("q12", "EXIT", "Z0", "q0", "Z0"),

    ("q0", "0-9", "Z0", "q13", "Z0"),

    ("q13", "ENTRY", "Z0", "q14", "Z0"),

    ("q14", "UP", "Z0", "q0", "Z0"),

    ("q14", "OPEN", "Z0", "q0", "Z0"),

    ("q14", "CLOSE", "Z0", "q0", "Z0"),

]

# Create a new directed graph for NPDA

npda\_dot = graphviz.Digraph(comment="Nondeterministic Pushdown Automaton")

# Add states to the graph

for state in npda\_states:

    if state == "q0":

        npda\_dot.node(state, state, shape="doublecircle")  # Accepting state

    else:

        npda\_dot.node(state, state)

# Add transitions to the graph

for (start, input\_symbol, stack\_symbol, end, new\_stack\_symbol) in npda\_transitions:

    label = f"{input\_symbol}, {stack\_symbol} -> {new\_stack\_symbol}"

    npda\_dot.edge(start, end, label=label)

# Display the graph in the notebook

display(npda\_dot)

**Python Code for DPDA:**

#DPDA

import graphviz

from IPython.display import display

# Define the states and transitions for DPDA

dpda\_states = ["q0", "q1", "q2", "q3", "q4", "q5", "q6", "q7", "q8", "q9", "q10", "q11", "q12", "q13", "q14"]

dpda\_input\_alphabet = ["0-9", "UP", "DOWN", "OPEN", "CLOSE", "ENTRY", "EXIT"]

dpda\_stack\_alphabet = ["Z0"]

# Define the transitions for DPDA

dpda\_transitions = [

    ("q0", "0-9", "Z0", "q1", "Z0"),

    ("q1", "ENTRY", "Z0", "q2", "Z0"),

    ("q2", "UP", "Z0", "q3", "Z0"),  # Ensuring determinism with only one transition for each input

    ("q3", "0-9", "Z0", "q4", "Z0"),

    ("q4", "EXIT", "Z0", "q0", "Z0"),

    ("q0", "0-9", "Z0", "q5", "Z0"),

    ("q5", "ENTRY", "Z0", "q6", "Z0"),

    ("q6", "OPEN", "Z0", "q7", "Z0"),  # Ensuring determinism with only one transition for each input

    ("q7", "0-9", "Z0", "q8", "Z0"),

    ("q8", "EXIT", "Z0", "q0", "Z0"),

    ("q0", "0-9", "Z0", "q9", "Z0"),

    ("q9", "ENTRY", "Z0", "q10", "Z0"),

    ("q10", "CLOSE", "Z0", "q11", "Z0"),  # Ensuring determinism with only one transition for each input

    ("q11", "0-9", "Z0", "q12", "Z0"),

    ("q12", "EXIT", "Z0", "q0", "Z0"),

    ("q0", "0-9", "Z0", "q13", "Z0"),

    ("q13", "ENTRY", "Z0", "q14", "Z0"),

    ("q14", "UP", "Z0", "q0", "Z0"),  # Ensuring determinism with only one transition for each input

]

# Create a new directed graph for DPDA

dpda\_dot = graphviz.Digraph(comment="Deterministic Pushdown Automaton")

# Add states to the graph

for state in dpda\_states:

    if state == "q0":

        dpda\_dot.node(state, state, shape="doublecircle")  # Accepting state

    else:

        dpda\_dot.node(state, state)

# Add transitions to the graph

for (start, input\_symbol, stack\_symbol, end, new\_stack\_symbol) in dpda\_transitions:

    label = f"{input\_symbol}, {stack\_symbol} -> {new\_stack\_symbol}"

    dpda\_dot.edge(start, end, label=label)

# Display the graph in the notebook

display(dpda\_dot)