Code Documentation

1. NFA to DFA conversion

I. Epsilon NFA to DFA

Step 1 – Consider M={Q, Σ, δ,q0,F) is NFA with ε. We have to convert this NFA with ε to equivalent DFA denoted by

M0=(Q0,Σ, δ0,q0,F0)

Then obtain,

ε-closure(q0) ={p1,p2,p3,……pn}

then [p1,p2,p2,….pn] becomes a start state of DFA

now[p1,p2,p3,….pn] ∈ Q0

```
epsilon_closures = {}
# Get all epsilon closure of all states
for state in self.transitions:
    epsilon_closures[state] = self.epsilon_closure(state)
print("Epsilon Closures:",epsilon_closures)
states = []
states_closures = []
ts = {}
newAlphabet = self.alphabet
newAlphabet.remove('eps')
i=0
# Add the epsilon closures of the first state
states_closures.append(list(epsilon_closures.values())[0])
j = len(states_closures)
newAcceptingStates = []
```

Step 2 – We will obtain δ transition on [p1,p2,p3,…pn] for each input.

δ 0([p1,p2,p3,..pn],a) = ε-closure(δ(p1,a) U δ(p2,a2)U……………. δ(pn,a))

= U (i=1 to n) ε-closure d(pi,a)

Where a is input ∈Σ

```python

# Loop on new added states
while j>0:

    closures = states_closures[i]

    l = list(closures)
    # If any state in accepting state is in this epsilon closures add it to the accepting states
    if len(set(self.acceptingStates).intersection(l)):
        newAcceptingStates.append('q'+str(i))


    ts['q'+str(i)] = {}
    states.append('q'+str(i))
    print('q'+str(i))
    print(closures)

    #  Loop on each alphabet
    for alpha in newAlphabet:
        nextState = set()
        # Get all next states based on the input alphabet
        for st in l:
            nextState |= set(self.transitions[st][alpha])
        ep = set()
        # loop on next states of the alphabet
        for n in nextState:
            if len(n)>0:
                # Get all epsilon closures for each next state
                ep |= epsilon_closures[n]
        print(alpha,ep)
        # If this state epsilon closure in not already added then add it
        if not(ep in states_closures) and len(ep)>0:
            states_closures.append(ep)
            j+=1
        # To handle when there is no edge between states or when there is no epsilon closures
        if len(ep)>0:
            ts['q'+str(i)][alpha] = 'q'+str(states_closures.index(ep))
        else:
            ts['q'+str(i)][alpha] = []


    print(ts)
    # Decrement j to loop
    j-=1
    # Increment the new states counter
    i+=1
```

Step 3 – The state obtained [p1,p2,p3,…pn] ∈ Q0 .

The states containing final state in pi is a final state in DFA

II. NFA to DFA without epsilon

Steps for converting NFA to DFA:

Step 1: Initially Q' = ϕ

Step 2: Add q0 of NFA to Q'. Then find the transitions from this start state.

```

states = [self.states[0]]
ts = {}
# Add first state
ts[self.states[0]] = getTransitionsOf(self.states[0],self.transitions)
i= len(states)
```

Step 3: In Q', find the possible set of states for each input symbol. If this set of states is not in Q', then add it to Q'.

```

# Loop whenever a state is added
while i>=0:
    # Loop to get (To states) from the transitions dictionary
    for a, toStates in ts[states[i- len(states)]].items():
            # If the same input alphabet goes to more than one state then name this state with '-' separated between the names of state
            temp = '-'.join(toStates)
            # If this state is not already added then add it
            if not(temp in states) and temp != '':
                newTransitions =  getTransitionsOf(temp,self.transitions)
                states.append(temp)
                ts[temp] = newTransitions
                i+=1
    i-=1
```

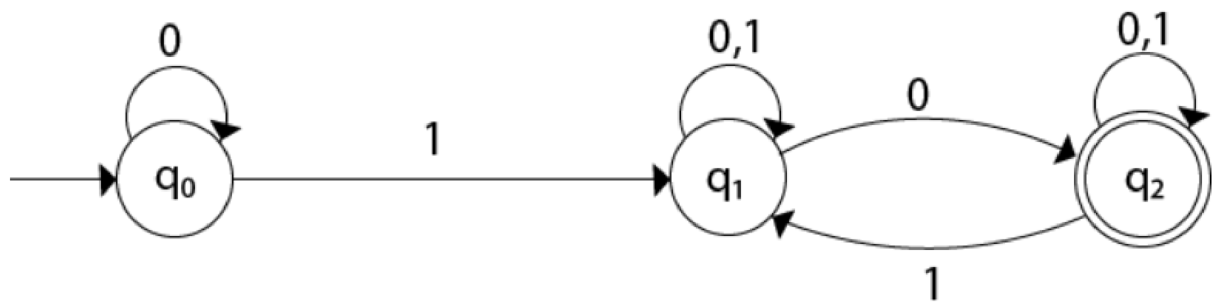
Step 4: In DFA, the final state will be all the states which contain F(final states of NFA)

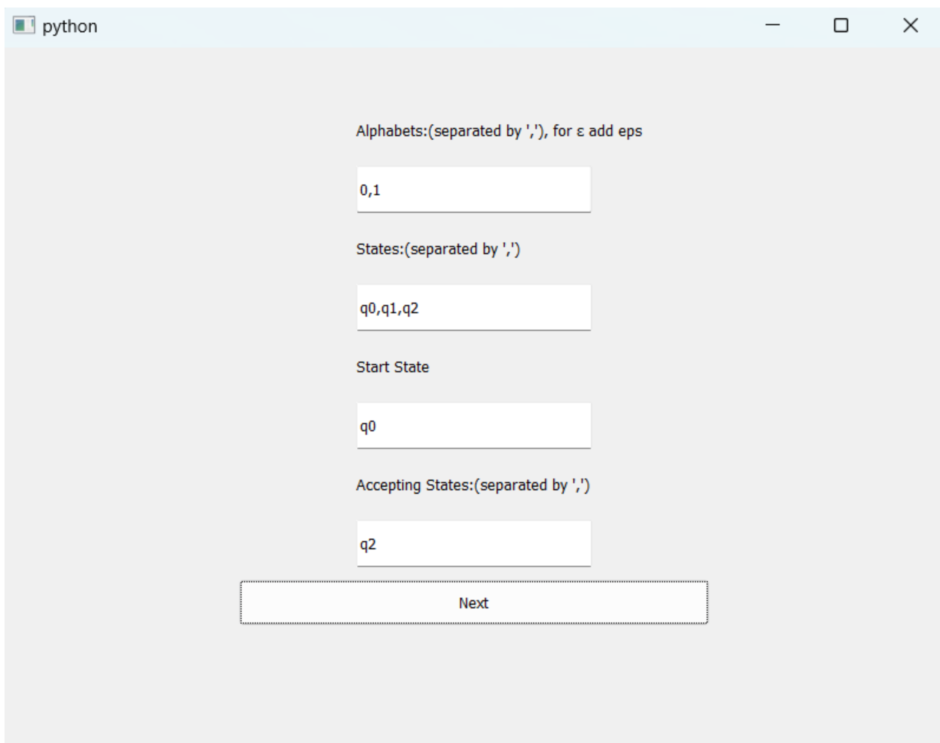
```
 # Function to get Accepting states
acceptingStates = self.getAcceptingStates(self.acceptingStates,states)

def getAcceptingStates(self,oldStates,newStates):
        states = []
        for s in newStates:
             if s in oldStates or len(set(oldStates).intersection(set(s.split('-
'))))>0:
                states.append(s)
        return states
```

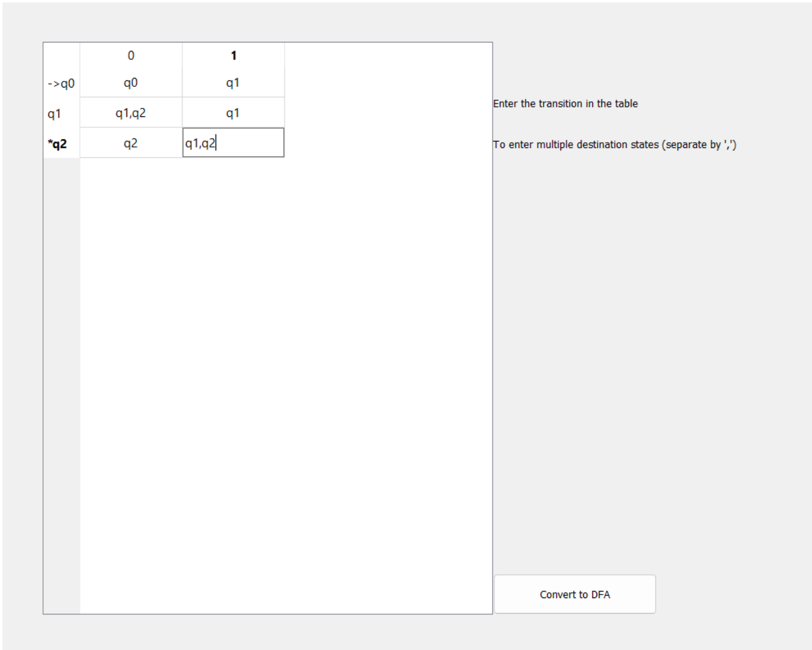
Screenshots:

For example:



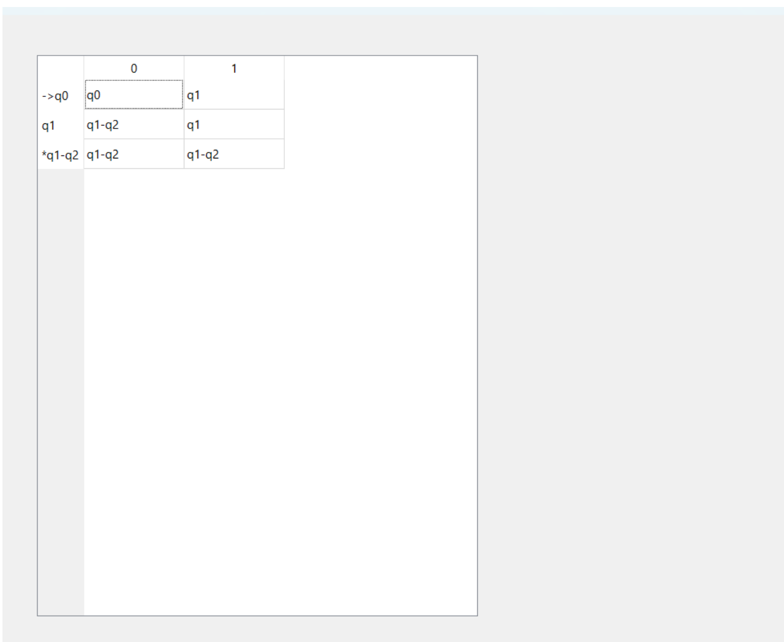


Input transition table:



	0	1
->q0	q0	q1
q1	q1,q2	q1
*q2	q2	q1,q2

Output:



	0	1
->q0	q0	q1
q1	q1-q2	q1
*q1-q2	q1-q2	q1-q2

2. CFG to PDA conversion

Step 1: The code defines a PDA (Pushdown Automaton) class, which simulates a context-free grammar (CFG) using a pushdown automaton.
Initialize the PDA instance: The __init__ method initializes a new PDA instance with the following parameters:
variables: A set of variables (non-terminal symbols) in the CFG.
terminals: A set of terminal symbols in the CFG.
start_variable: The start variable for the CFG.
production_rules: A dictionary containing the production rules for each variable.

```
class PDA:
    def __init__(self, variables, terminals, start_variable, production_rules):
        self.states = ['q_start', 'q_push_start_var', 'q_loop', 'q_final']
        self.variables = variables
        self.terminals = terminals
        self.start_variable = start_variable
        self.production_rules = production_rules
        self.transitions = self.create_transitions()
```

Step 2: The create_transitions method generates the transition rules for the PDA, based on the given CFG. The transitions are created for the following scenarios:

- Transition from q_start to q_push_start_var, pushing the stack symbol $.
- Transition from q_push_start_var to q_loop, pushing the start variable.
- Transitions for production rules, in the q_loop state. If a variable A produces a string w, the PDA pushes the symbols of w in reverse order onto the stack.
- Transitions for terminal symbols, in the q_loop state, reading the terminal symbol from the input and popping it from the stack.
- Transition from q_loop to q_final, popping the stack symbol $.

```
def create_transitions(self):
    transitions = {}

    # Transition from q_start to q_push_start_var, pushing the stack symbol $
    transitions[('q_start', '', '')] = [('q_push_start_var', '$')]

    # Transition from q_push_start_var to q_loop, pushing the start variable
    transitions[('q_push_start_var', '', '')] = [('q_loop', self.start_variable)]

    # Transitions for production rules, in q_loop state
    state_counter = 1

    # Loop through the production rules for each variable
    for variable in self.variables:
        for prod_rule in self.production_rules[variable]:
            previous_state = 'q_loop'

            # Loop through the symbols in the production rule, in reverse order
            for symbol in enumerate(prod_rule[::-1]):
                new_state = f'q_alt_{state_counter}'
                state_counter += 1
                transitions[(previous_state, '', variable)] = [(new_state, symbol)]
                previous_state = new_state
                variable = ''  # Clear the variable after the first iteration

            # Transition back to q_loop after pushing all symbols
            transitions[(previous_state, '', '')] = [('q_loop', '')]
```

```python
# Transitions for terminal symbols, in q_loop state
for terminal in self.terminals:
    # For each terminal symbol a, with input a and pop a, push ε
    transitions[('q_loop', terminal, terminal)] = [('q_loop', '')]

# Transition from q_loop to q_final, popping the stack symbol $
transitions[('q_loop', '', '$')] = [('q_final', '')]

return transitions
```

Step 3: The main function processes the given CFG by creating a PDA instance and printing the generated transition rules.

```python
# Main function for testing the PDA class
def main():
    # Define variables, terminals, start variable, and production rules
    variables = {'S','B'}
    terminals = {'a', 'b','c'}
    start_variable = 'S'
    production_rules = {
        'S': ['aBc', 'ab'],
        'B': ['SB', '']
    }

    # Create a PDA object with the given parameters
    pda = PDA(variables, terminals, start_variable, production_rules)

    # Print the transitions in a human-readable format
    for key, values in pda.transitions.items():
        for value in values:
            print(f"From {key[0]} with input '{key[1]}' and pop '{key[2]}': to {value[0]} and push '{value[1]}'")

    print(f"")
    print(f"another format\n")

    # Print the transitions in another format
    for key, values i (variable) values: Any ):
        for value in values:
            print(f"{key[0]}, '{key[1]}', '{key[2]}': --> {value[0]}, '{value[1]}'")
```

past cfg

```python
variables = {'S','B'}
terminals = {'a', 'b','c'}
start_variable = 'S'
production_rules = {
    'S': ['aBc', 'ab'],
    'B': ['SB', '']
}

# Create a PDA object with the given parameters
pda = PDA(variables, terminals, start_variable, production_rules)
```

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL

PS C:\Users\ziada> & C:/Users/ziada/AppData/Local/Programs/Python/Python311/python.exe "c:/Users/ziada/OneDrive/
From q_start with input '' and pop '': to q_push_start_var and push '$'
From q_push_start_var with input '' and pop '': to q_loop and push 'S'
From q_loop with input '' and pop 'S': to q_alt_1 and push '(0, 'c')'
From q_alt_1 with input '' and pop '': to q_alt_2 and push '(1, 'B')'
From q_alt_2 with input '' and pop '': to q_alt_3 and push '(2, 'a')'
From q_alt_3 with input '' and pop '': to q_loop and push ''
From q_loop with input '' and pop '': to q_loop and push ''
From q_alt_4 with input '' and pop '': to q_alt_5 and push '(1, 'a')'
From q_alt_5 with input '' and pop '': to q_loop and push ''
From q_loop with input '' and pop 'B': to q_alt_6 and push '(0, 'B')'
From q_alt_6 with input '' and pop '': to q_alt_7 and push '(1, 'S')'
From q_alt_7 with input '' and pop '': to q_loop and push ''
From q_loop with input 'c' and pop 'c': to q_loop and push ''
From q_loop with input 'b' and pop 'b': to q_loop and push ''
From q_loop with input 'a' and pop 'a': to q_loop and push ''
From q_loop with input '' and pop '$': to q_final and push ''
```

Another Format

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL

From q_loop with input '' and pop '$': to q_final and push

another format

q_start, '', '': --> q_push_start_var, '$'
q_push_start_var, '', '': --> q_loop, 'S'
q_loop, '', 'S': --> q_alt_1, '(0, 'c')'
q_alt_1, '', '': --> q_alt_2, '(1, 'B')'
q_alt_2, '', '': --> q_alt_3, '(2, 'a')'
q_alt_3, '', '': --> q_loop, ''
q_loop, '', '': --> q_loop, ''
q_alt_4, '', '': --> q_alt_5, '(1, 'a')'
q_alt_5, '', '': --> q_loop, ''
q_loop, '', 'B': --> q_alt_6, '(0, 'B')'
q_alt_6, '', '': --> q_alt_7, '(1, 'S')'
q_alt_7, '', '': --> q_loop, ''
q_loop, 'c', 'c': --> q_loop, ''
q_loop, 'b', 'b': --> q_loop, ''
q_loop, 'a', 'a': --> q_loop, ''
q_loop, '', '$': --> q_final, ''
```