Mehmet Akif Elem

Student No:2526283

HW1- CNG280

Instructor: Prof.Dr.Ahmet Coşar

Code:

```
class NFA:
    def __init__(self, states, alphabet, transition, start_state, final_states):
        self.states = states
        self.alphabet = alphabet
        self.transition = transition
        self.start_state = start_state
        self.final_states = final_states

    def track_string(self, string):

        current_states = {self.start_state}
        print(f"Start state: {current_states}")

        for symbol in string:
            next_states = set()

            for state in current_states:
                if state in self.transition and symbol in self.transition[state]:
                    next_states.update(self.transition[state][symbol])
            current_states = next_states
            print(f"After input '{symbol}', reached states: {current_states}")

        if current_states & self.final_states:
            print("Accepted")

        else:
            print("Rejected")


class DFA:
    def __init__(self, states, alphabet, transition, start_state, final_states):
        self.states = states
        self.alphabet = alphabet
        self.transition = transition
        self.start_state = start_state
        self.final_states = final_states

    def track_string(self, string):

        current_state = self.start_state
        print(f"Start state: {current_state}")


        for symbol in string:
            if current_state in self.transition and symbol in self.transition[current_state]:
                current_state = self.transition[current_state][symbol]
                print(f"After input '{symbol}', current state: {current_state}")
            else:
                print("Rejected")
                return

            if current_state in self.final_states:
                print("Accepted")
            else:
                print("Rejected")


def nfa_to_dfa(nfa):

    dfa_states = []
    dfa_transitions = {}

    dfa_start_state = tuple([nfa.start_state])
    dfa_states.append(dfa_start_state)


    index = 0
    while index < len(dfa_states):
        current_set = dfa_states[index]
        index += 1

        dfa_transitions[current_set] = {}

        for symbol in nfa.alphabet:
            next_state = set()
            for state in current_set:

                if state in nfa.transition and symbol in nfa.transition[state]:
                    next_state.update(nfa.transition[state][symbol])

            next_state_tuple = tuple(sorted(next_state))
            if next_state_tuple not in dfa_states and next_state_tuple:
                dfa_states.append(next_state_tuple)

            dfa_transitions[current_set][symbol] = next_state_tuple

    dfa_final_states = {state for state in dfa_states if any(s in nfa.final_states for s in state)}

    return DFA(set(dfa_states), nfa.alphabet, dfa_transitions, dfa_start_state, dfa_final_states)


def reverse_nfa(nfa):
    reversed_transitions = {state: {} for state in nfa.states}

    for state, transitions in nfa.transition.items():
        for symbol, next_states in transitions.items():

            for next_state in next_states:
                if next_state not in reversed_transitions:

                    reversed_transitions[next_state] = {}
                if symbol not in reversed_transitions[next_state]:
                    reversed_transitions[next_state][symbol] = set()
                reversed_transitions[next_state][symbol].add(state)

    return NFA(
        states=nfa.states,
        alphabet=nfa.alphabet,
        transition=reversed_transitions,
        start_state=list(nfa.final_states)[0],
        final_states={nfa.start_state}
    )


nfa = NFA(
    states={'q0', 'q1', 'q2'},
    alphabet={'a', 'b'},
    transition={
        'q0': {'a': {'q0', 'q1'}, 'b': {'q1'}},
        'q1': {'b': {'q0'}, 'a': {'q2'}},
        'q2': {'a': {'q2'}, 'b': {'q1'}}
    },
    start_state='q0',
    final_states={'q2'}
)

dfa = nfa_to_dfa(nfa)

reverse_nfa = reverse_nfa(nfa)

reverse_dfa = nfa_to_dfa(reverse_nfa)

while True:
    user_input = input("\nEnter a string composed of a and b (enter exit to quit): ").strip()

    if user_input.lower() == 'exit':
        break

    print("\n--- Testing on nfa ---")
    nfa.track_string(user_input)


    print("\n--- Testing on dfa ---")
    dfa.track_string(user_input)

    reversed_input = user_input[::-1]
    print(f"\n--- testing on dfaa for L reverse (reversed input: {reversed_input}) ---")
    reverse_dfa.track_string(reversed_input)
```

Output of the code:

a) Run your programs on DFAs of the following two regular languages on sample inputs: {a^nb} for n>=0, {ba}.

```
(.venv) (base) mac@Mac-MacBook-Pro 280HWTRY % python3 main.py
Enter a string composed of a and b (enter exit to quit): b

--- Testing on nfa ---
Start state: {'q0'}
After input 'b', reached states: {'q1'}
Rejected

--- Testing on dfa ---
Start state: ('q0',)
After input 'b', current state: ('q1',)
Rejected

--- testing on dfaa for L reverse (reversed input: b) ---
Start state: ('q2',)
After input 'b', current state: ()
Rejected

Enter a string composed of a and b (enter exit to quit): ab

--- Testing on nfa ---
Start state: {'q0'}
After input 'a', reached states: {'q1', 'q0'}
After input 'b', reached states: {'q1', 'q0'}
Rejected

--- Testing on dfa ---
Start state: ('q0',)
After input 'a', current state: ('q0', 'q1')
After input 'b', current state: ('q0', 'q1')
Rejected

--- testing on dfaa for L reverse (reversed input: ba) ---
Start state: ('q2',)
After input 'b', current state: ()
Rejected

Enter a string composed of a and b (enter exit to quit): aab

--- Testing on nfa ---
```

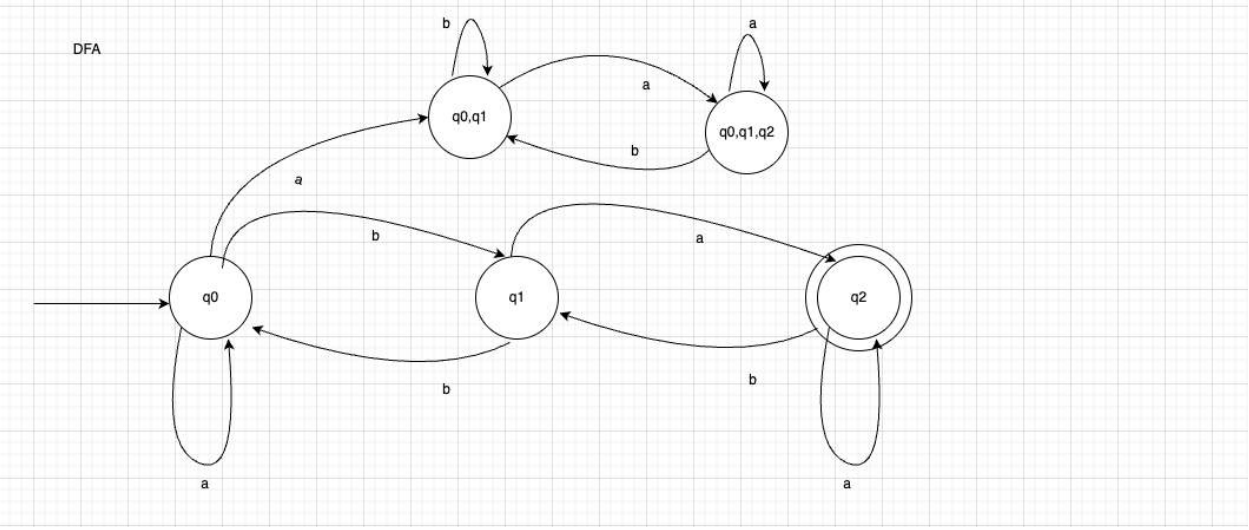
```
Start state: {'q0'}
After input 'a', reached states: {'q1', 'q0'}
After input 'a', reached states: {'q1', 'q2', 'q0'}
After input 'b', reached states: {'q1', 'q0'}
Rejected

--- Testing on dfa ---
Start state: ('q0',)
After input 'a', current state: ('q0', 'q1')
After input 'a', current state: ('q0', 'q1', 'q2')
After input 'b', current state: ('q0', 'q1')
Rejected

--- testing on dfaa for L reverse (reversed input: baa) ---
Start state: ('q2',)
After input 'b', current state: ()
Rejected

Enter a string composed of a and b (enter exit to quit): aaaab

--- Testing on nfa ---
Start state: {'q0'}
After input 'a', reached states: {'q1', 'q0'}
After input 'a', reached states: {'q1', 'q2', 'q0'}
After input 'a', reached states: {'q1', 'q2', 'q0'}
After input 'a', reached states: {'q1', 'q2', 'q0'}
After input 'b', reached states: {'q1', 'q0'}
Rejected

--- Testing on dfa ---
Start state: ('q0',)
After input 'a', current state: ('q0', 'q1')
After input 'a', current state: ('q0', 'q1', 'q2')
After input 'a', current state: ('q0', 'q1', 'q2')
After input 'a', current state: ('q0', 'q1', 'q2')
After input 'b', current state: ('q0', 'q1')
Rejected

--- testing on dfaa for L reverse (reversed input: baaaa) ---
Start state: ('q2',)
After input 'b', current state: ()
Rejected

Enter a string composed of a and b (enter exit to quit): ba
```

```
--- Testing on nfa ---
Start state: {'q0'}
After input 'b', reached states: {'q1'}
After input 'a', reached states: {'q2'}
Accepted

--- Testing on dfa ---
Start state: ('q0',)
After input 'b', current state: ('q1',)
After input 'a', current state: ('q2',)
Accepted

--- testing on dfaa for L reverse (reversed input: ab) ---
Start state: ('q2',)
After input 'a', current state: ('q1', 'q2')
After input 'b', current state: ('q0', 'q2')
Accepted


Enter a string composed of a and b (enter exit to quit): a

--- Testing on nfa ---
Start state: {'q0'}
After input 'a', reached states: {'q1', 'q0'}
Rejected

--- Testing on dfa ---
Start state: ('q0',)
After input 'a', current state: ('q0', 'q1')
Rejected

--- testing on dfaa for L reverse (reversed input: a) ---
Start state: ('q2',)
After input 'a', current state: ('q1', 'q2')
Rejected

Enter a string composed of a and b (enter exit to quit): ba

--- Testing on nfa ---
Start state: {'q0'}
After input 'b', reached states: {'q1'}
After input 'a', reached states: {'q2'}
Accepted
```

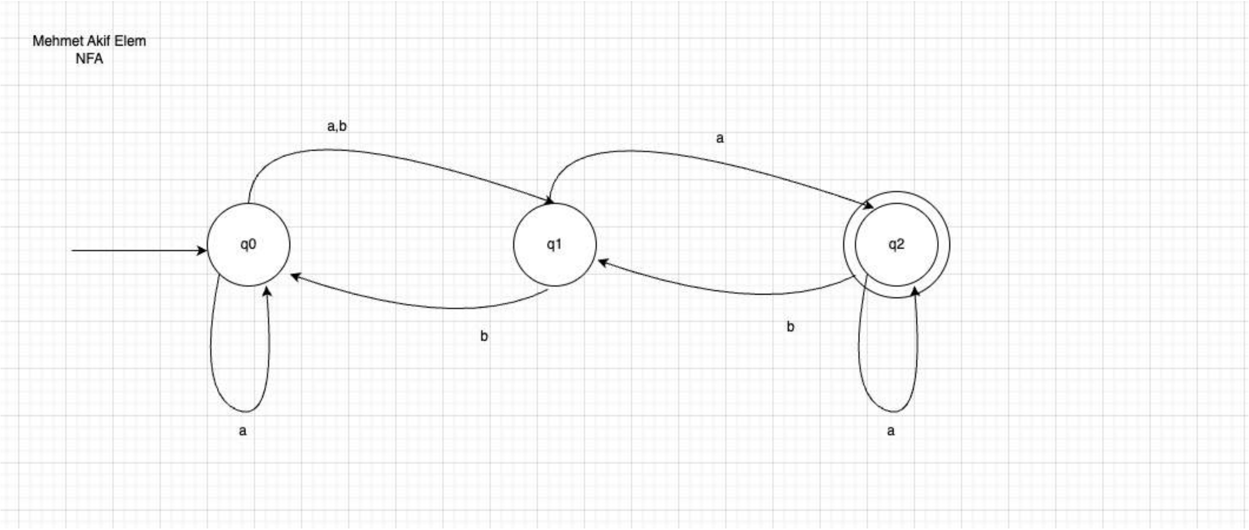
```
--- Testing on dfa ---
Start state: ('q0',)
After input 'b', current state: ('q1',)
After input 'a', current state: ('q2',)
Accepted

--- testing on dfaa for L reverse (reversed input: ab) ---
Start state: ('q2',)
After input 'a', current state: ('q1', 'q2')
After input 'b', current state: ('q0', 'q2')
Accepted
```

b) Draw the input DFAs and generated NFAs for corresponding reversed languages.

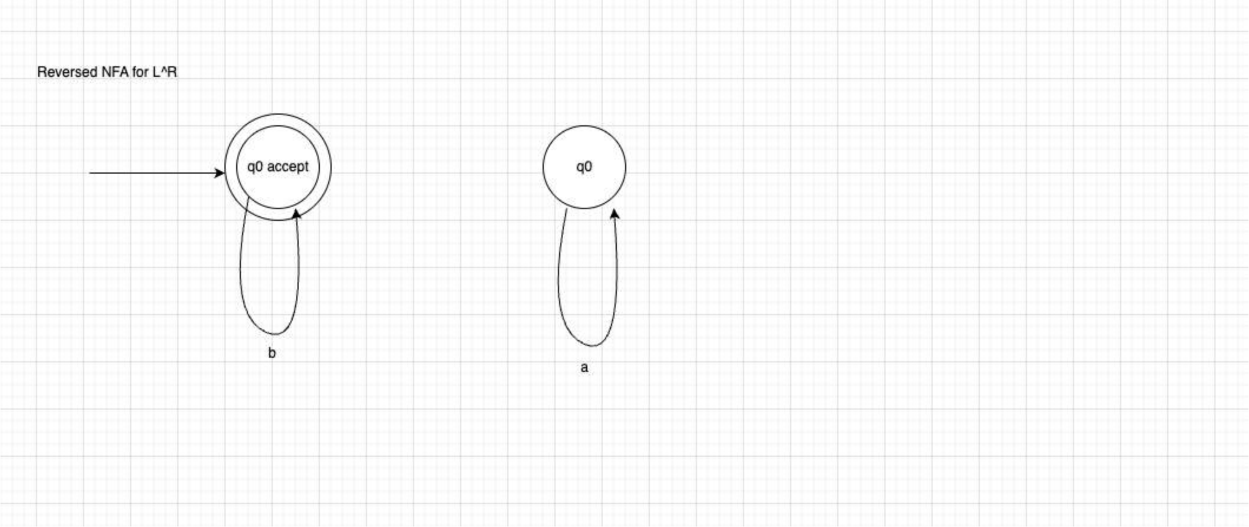
DFA:



NFA:



Reversed NFA for L^R:


Reversed NFA for L^R
q0 accept
q0
b
a