Implementation of acceptor finite-state automata

Ihor Orlenko

Research supervisors
associate professor, Candidate of Science in Physics and Mathematics O.S. Antonenko
senior lecturer G.P. Kuznetsova

FMPIT,
Odesa I.I. Mechnikov National University

All-Ukrainian competition of student scientific works, Feb 24
I degree diploma

Contents

Deterministic finite automaton (DFA)

Definition 1.1

A **deterministic finite automaton** (abbreviated DFA) is called an ordered quintuple $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$, which consists of the following elements:

(a) $Q = \{q_0, \ldots, q_m\}$ — a finite set of automaton states;

Deterministic finite automaton (DFA)

Definition 1.1

A **deterministic finite automaton** (abbreviated DFA) is called an ordered quintuple $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$, which consists of the following elements:

(a) $Q = \{q_0, \ldots, q_m\}$ — a finite set of automaton states;

(b) $\Sigma = \{a_0, \ldots, a_n\}$ — a finite set of the automaton's alphabet;

Deterministic finite automaton (DFA)

Definition 1.1

A **deterministic finite automaton** (abbreviated DFA) is called an ordered quintuple $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$, which consists of the following elements:

(a) $Q = \{q_0, \ldots, q_m\}$ — a finite set of automaton states;

(b) $\Sigma = \{a_0, \ldots, a_n\}$ — a finite set of the automaton's alphabet;

(c) $\delta : Q \times \Sigma \to Q$ — a transition function;

Deterministic finite automaton (DFA)

Definition 1.1

A **deterministic finite automaton** (abbreviated DFA) is called an ordered quintuple $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$, which consists of the following elements:

(a) $Q = \{q_0, \ldots, q_m\}$ — a finite set of automaton states;

(b) $\Sigma = \{a_0, \ldots, a_n\}$ — a finite set of the automaton's alphabet;

(c) $\delta : Q \times \Sigma \to Q$ — a transition function;

(d) $q_0 \in Q$ — an initial state;

Deterministic finite automaton (DFA)

Definition 1.1

A **deterministic finite automaton** (abbreviated DFA) is called an ordered quintuple $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$, which consists of the following elements:

(a) $Q = \{q_0, \ldots, q_m\}$ — a finite set of automaton states;

(b) $\Sigma = \{a_0, \ldots, a_n\}$ — a finite set of the automaton's alphabet;

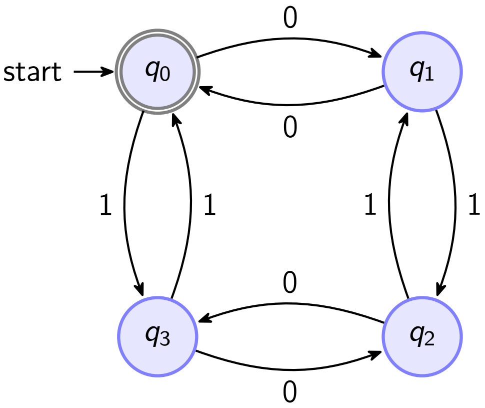
(c) $\delta : Q \times \Sigma \to Q$ — a transition function;

(d) $q_0 \in Q$ — an initial state;

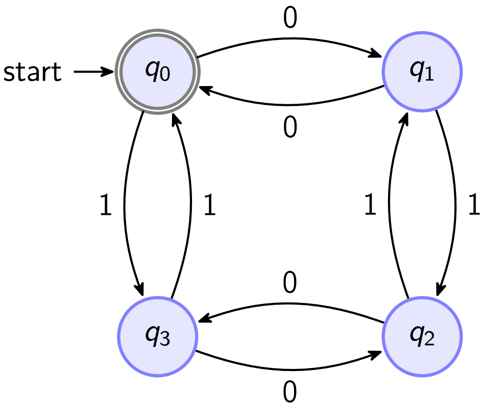
(e) $F \subseteq Q$ — a set of accepting (final) states.

Deterministic finite automaton (DFA)

Example of an automaton



$\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ - an automaton that recognizes words with an even number of 0s and an even number of 1s

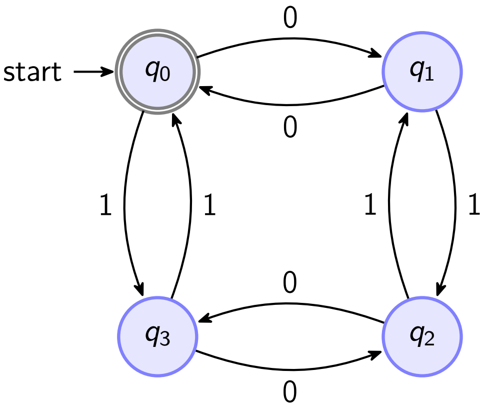


$\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ - an automaton that recognizes words with an even number of 0s and an even number of 1s

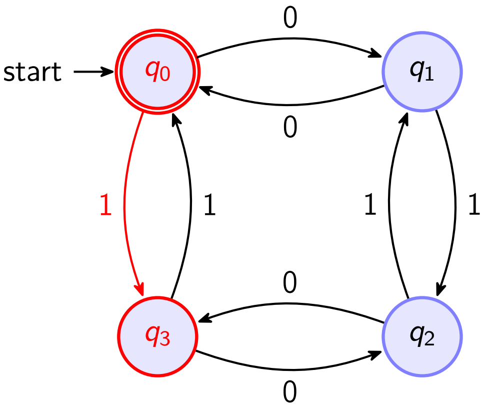
(a) $Q = \{q_0, q_1, q_2, q_3\}$
(b) $\Sigma = \{0, 1\}$
(c) $\delta : Q \times \Sigma \to Q$
(d) q_0 - the initial state
(e) $F = \{q_0\}$

Deterministic finite automaton (DFA)

Word Recognition

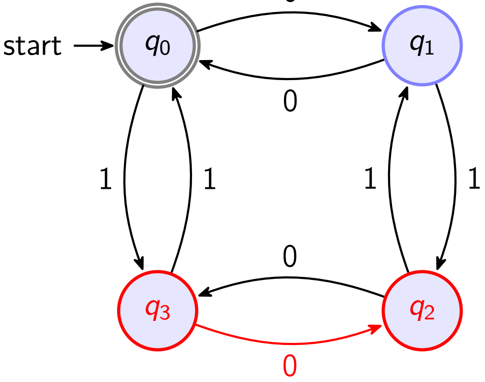


Let $\omega = 101011$, then the automaton $\mathcal{A}$ reads and recognizes the word ω as indicated below:



Let $\omega = 101011$, then the automaton $\mathcal{A}$ reads and recognizes the word ω as indicated below:

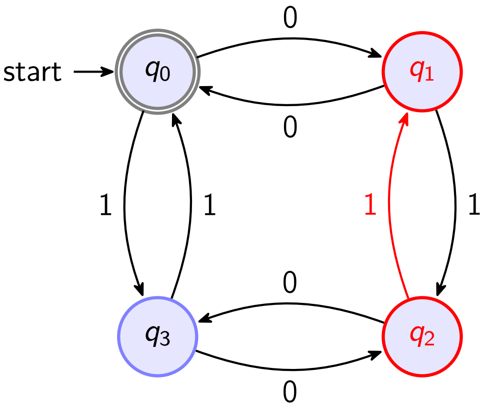
(1) $\delta : (q_0, 1) \to q_3$



Let $\omega = 101011$, then the automaton $\mathcal{A}$ reads and recognizes the word ω as indicated below:

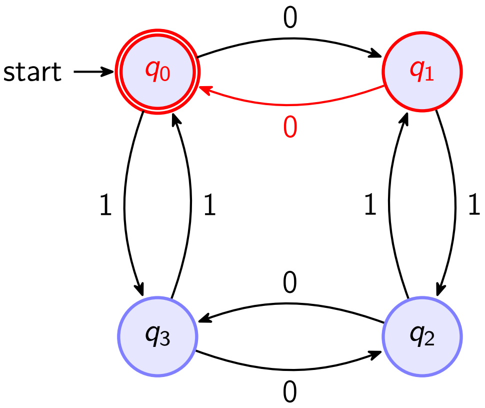
(1) $\delta : (q_0, 1) \rightarrow q_3$

(2) $\delta : (q_3, 0) \rightarrow q_2$



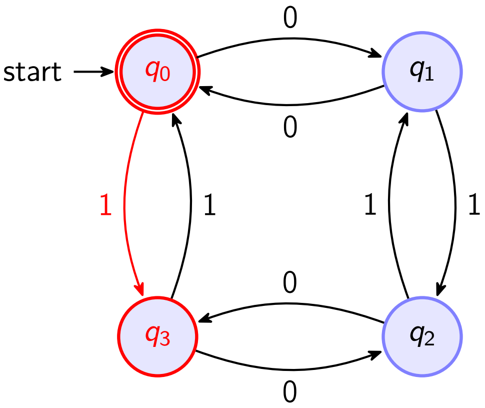
Let $\omega = 101011$, then the automaton $\mathcal{A}$ reads and recognizes the word ω as indicated below:

(1) $\delta : (q_0, 1) \to q_3$
(2) $\delta : (q_3, 0) \to q_2$
(3) $\delta : (q_2, 1) \to q_1$



Let $\omega = 101011$, then the automaton $\mathcal{A}$ reads and recognizes the word ω as indicated below:

(1) $\delta : (q_0, 1) \rightarrow q_3$
(2) $\delta : (q_3, 0) \rightarrow q_2$
(3) $\delta : (q_2, 1) \rightarrow q_1$
(4) $\delta : (q_1, 0) \rightarrow q_0$

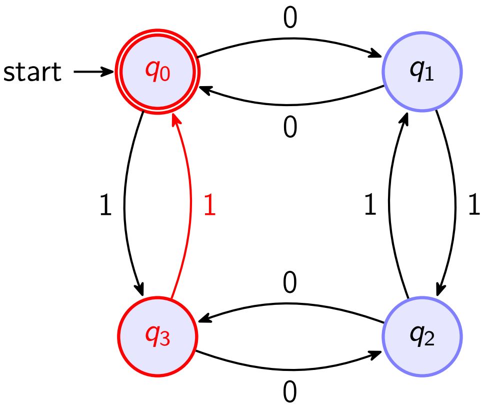


Let $\omega = 101011$, then the automaton $\mathcal{A}$ reads and recognizes the word ω as indicated below:

(1) $\delta : (q_0, 1) \rightarrow q_3$
(2) $\delta : (q_3, 0) \rightarrow q_2$
(3) $\delta : (q_2, 1) \rightarrow q_1$
(4) $\delta : (q_1, 0) \rightarrow q_0$
(5) $\delta : (q_0, 1) \rightarrow q_3$

Deterministic finite automaton (DFA)

Word Recognition



Let $\omega = 101011$, then the automaton $\mathcal{A}$ reads and recognizes the word ω as indicated below:

(1) $\delta : (q_0, 1) \to q_3$

(2) $\delta : (q_3, 0) \to q_2$

(3) $\delta : (q_2, 1) \to q_1$

(4) $\delta : (q_1, 0) \to q_0$

(5) $\delta : (q_0, 1) \to q_3$

(6) $\delta : (q_3, 1) \to q_0$

Deterministic finite automaton (DFA)

Automaton Implementation

```
class DFA:

    class TransitionFunction:
        #  ... description of the class ...

    #  constructor of the DFA class
    def __init__(self):
        self.states = set()
        self._start_state = None
        self.alphabet = set()
        self.transition_function = self.TransitionFunction()
        self._accept_states = None
```

Deterministic finite automaton (DFA)

Implementation of the transition function class

```python
class TransitionFunction:

    # constructor of the class
    def __init__(self):
        self.__data = {}

    # function of adding to the dictionary __data
    # iterates through the template state>symbol>next_state
    def add(self, transition):
        state, symbol, next_state = transition.split('>')
        if state not in self.__data:
            self.__data[state] = {}
        if symbol in self.__data[state]:
            raise AutomatonInputError("DFA mustn't contain"
                                      " non-deterministic transitions")
        self.__data[state][symbol] = next_state
```

Deterministic finite automaton (DFA)

Implementation of the recognition mechanism

```
class DFA:

    # The recognition function, which takes a string word,
    # executes transitions in the automaton according to each symbol,
    # and determines the final state
    def recognize(self, word):
        current_state = self.start_state
        for symbol in word:
            if symbol not in self.alphabet \
                    or current_state not in self.transition_function \
                    or symbol not in self.transition_function[current_state]:
                return False
            current_state = self.transition_function[current_state][symbol]
        return current_state in self.accept_states
```

Deterministic finite automaton (DFA)

Example of algorithm operation

```python
dfa = DFA()
dfa._start_state = "q0"
dfa._accept_states = {"q0"}
dfa.update_transition([
    "q0>0>q1", "q0>1>q3",
    "q1>0>q0", "q1>1>q2",
    "q2>0>q3", "q2>1>q1",
    "q3>0>q2", "q3>1>q0",
])

print(dfa)

word = "101011"
print(f"The word {word} is recognized:
↪  {dfa.recognize(word)}")
```

```
<class 'automata.DFA'>
States: {'q2', 'q0', 'q3', 'q1'}
Start state: q0
Alphabet: {'1', '0'}
Accept states: {'q0'}
Transition function:
q0: {'0': 'q1', '1': 'q3'}
q1: {'0': 'q0', '1': 'q2'}
q2: {'0': 'q3', '1': 'q1'}
q3: {'0': 'q2', '1': 'q0'}

The word 101011 is recognized: True
```

Nondeterministic finite automaton (NFA)

Definition 2.1

A **nondeterministic finite automaton** (abbreviated as NFA) is called an ordered quintuple $\mathcal{A} = (Q, \Sigma, \Delta, q_0, F)$, in which Q, Σ, q_0, F are defined as in the deterministic case, but the transition function Δ is of the form $\Delta : Q \times \Sigma \to \mathcal{P}(Q)$, where $\mathcal{P}(Q)$ is the power set of Q (the set of all subsets of Q).

Nondeterministic finite automaton (NFA)

Definition 2.1

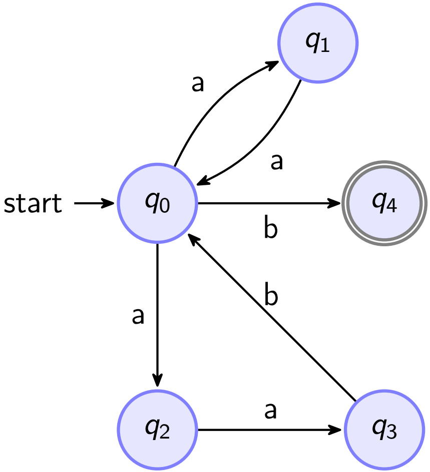
A **nondeterministic finite automaton** (abbreviated as NFA) is called an ordered quintuple $\mathcal{A} = (Q, \Sigma, \Delta, q_0, F)$, in which Q, Σ, q_0, F are defined as in the deterministic case, but the transition function Δ is of the form $\Delta : Q \times \Sigma \to \mathcal{P}(Q)$, where $\mathcal{P}(Q)$ is the power set of Q (the set of all subsets of Q).

Definition 2.2

The language $\mathcal{L}(\mathcal{A})$ defined by an automaton $\mathcal{A}$ is $\{\omega \in \Sigma^* | \delta^*(q_0, \omega) \in F\}$, the set of all words that are recognized by the deterministic automaton $\mathcal{A}$.
The language $\mathcal{L} \subseteq \Sigma^*$ is said to be **automaton-recognizable**, and there exists a deterministic finite automaton $\mathcal{A}$ such that $\mathcal{L} = \mathcal{L}(\mathcal{A})$.

Nondeterministic finite automaton (NFA)

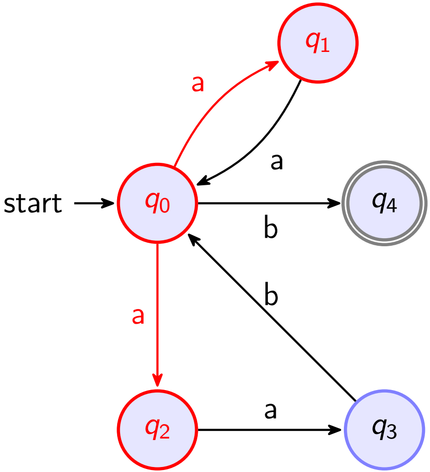
Example of automaton



$$\mathcal{A} = (Q, \Sigma, \Delta, q_0, F)$$

Nondeterministic finite automaton (NFA)

Example of automaton

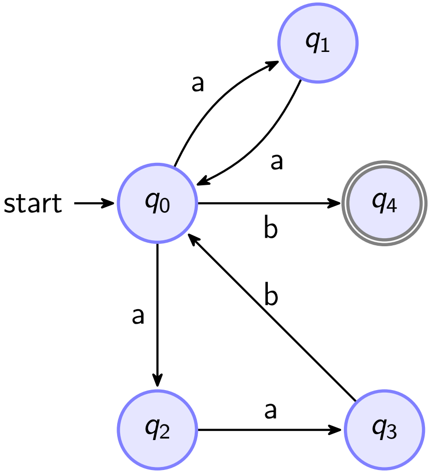


$$\mathcal{A} = (Q, \Sigma, \Delta, q_0, F)$$

$$\delta : (q_0, a) \to \{q_1, q_2\} \in \Delta$$

Nondeterministic finite automaton (NFA)

Example of automaton



$$\mathcal{A} = (Q, \Sigma, \Delta, q_0, F)$$

$$\delta : (q_0, a) \to \{q_1, q_2\} \in \Delta$$

$$\mathcal{L}(\mathcal{A}) = \{aa, aab\}^*\{b\}$$

$$\omega = aabaab \in \mathcal{L}(\mathcal{A})$$

```python
# NFA class, inherited as a subclass of the DFA class
class NFA(DFA):
    # Overriding the DFA.TransitionFunction class
    class TransitionFunction(DFA.TransitionFunction):
        def add(self, transition):
            state, symbol, next_state = transition.split('>')
            if state not in self.__data:
                self.__data[state] = {}
            if symbol not in self.__data[state]:
                self.__data[state][symbol] = set()
            self.__data[state][symbol].add(next_state)

    # Constructor of the NFA class
    def __init__(self):
        super().__init__()
        self.transition_function = self.TransitionFunction()
```

Nondeterministic finite automaton (NFA)

Implementation of the recognition mechanism

```
class NFA(DFA):
    # Overriding DFA's recognize function
    def recognize(self, word):
        current_states = {self._start_state}

        for symbol in word:
            next_states = set()
            for state in current_states:
                if symbol in self.transition_function[state]:
                    next_states.update(self.transition_function[state][symbol])
            current_states = next_states

            if not current_states:
                return False

        return any(state in self._accept_states for state in current_states)
```

Nondeterministic finite automaton (NFA)

Example of algorithm operation

```python
nfa = NFA()
nfa._start_state = "q0"
nfa._accept_states = {"q4"}
nfa.update_transition([
    "q0>a>q1", "q0>a>q2", "q0>b>q4",
    "q1>a>q0",
    "q2>a>q3",
    "q3>b>q0",
])

print(nfa)

word = "aabaab"
print(f"The word {word} is recognized:
↪  {nfa.recognize(word)}")
```

```
<class 'automata.NFA'>
States: {'q1', 'q0', 'q4', 'q2', 'q3'}
Start state: q0
Alphabet: {'a', 'b'}
Accept states: {'q4'}
Transition function:
q0: {'a': {'q2', 'q1'}, 'b': {'q4'}}
q1: {'a': {'q0'}}
q2: {'a': {'q3'}}
q3: {'b': {'q0'}}

The word aabaab is recognized: True
```

ε - Nondeterministic finite automaton (ε-NFA)

Definition 3.1

A **nondeterministic finite automaton with ε-transitions** (abbreviated as ε-NFA) is called an ordered quintuple $\mathcal{A} = (Q, \Sigma, \Delta, q_0, F)$, where Q, Σ, q_0, F are defined as in the nondeterministic automaton without ε-transitions, but the transition function Δ is extended to $\Delta : Q \times (\Sigma \cup \{\varepsilon\}) \to \mathcal{P}(Q)$.

ε - Nondeterministic finite automaton (ε-NFA)

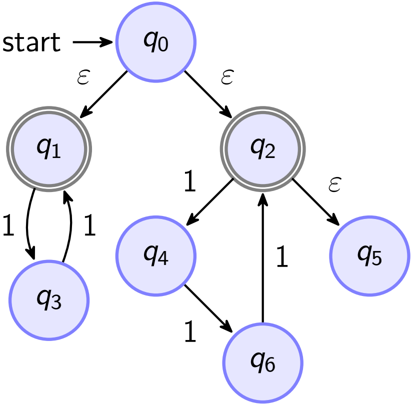
Definition 3.1

A **nondeterministic finite automaton with ε-transitions** (abbreviated as ε-NFA) is called an ordered quintuple $\mathcal{A} = (Q, \Sigma, \Delta, q_0, F)$, where Q, Σ, q_0, F are defined as in the nondeterministic automaton without ε-transitions, but the transition function Δ is extended to $\Delta : Q \times (\Sigma \cup \{\varepsilon\}) \to \mathcal{P}(Q)$.

Definition 3.2

Let $\mathcal{A} = (Q, \Sigma, \Delta, q_0, F)$ be an ε-NFA. The **closure** of a state $q \in Q$ denoted as $Orb(q)$ is the set of all states that are reachable from state q through ε-transitions and is defined by the following conditions:

- $q \in P$,
- If $p \in P$ i $\Delta(p, \varepsilon) \cap Q \neq \varnothing$, then $\Delta(p, \varepsilon) \subseteq P$.

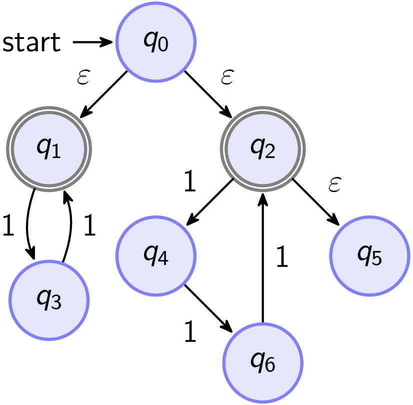


$\mathcal{A}$ - ε-NFA, such that

$$\mathcal{L}(\mathcal{A}) = \{1^n : n - \text{is even, or divisible by } 3\}$$

ε - Nondeterministic finite automaton (ε-NFA)

Example of automaton and closure

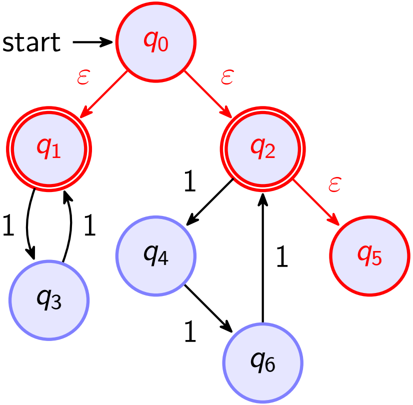


$\mathcal{A}$ - ε-NFA, such that

$$\mathcal{L}(\mathcal{A}) = \{1^n : n - \text{is even, or divisible by } 3\}$$

$$\omega = 111111 \in \mathcal{L}(\mathcal{A})$$

$$\omega = 11111 \notin \mathcal{L}(\mathcal{A})$$



$\mathcal{A}$ - ε-NFA, such that

$$\mathcal{L}(\mathcal{A}) = \{1^n : n - \text{is even, or divisible by 3}\}$$

$$\omega = 111111 \in \mathcal{L}(\mathcal{A})$$

$$\omega = 11111 \notin \mathcal{L}(\mathcal{A})$$

$$Orb(q_0) = \{q_0, q_1, q_2, q_5\}$$

```
# Defined class NFA
class eNFA(NFA):

    # Defined constructor of class NFA
    def __init__(self):
        super().__init__()
        self.epsilon = 'eps'

    # Method that computes the epsilon-closure
    def epsilon_closure(self, states):
        # ... description of the function ...

    # Method that recognizes the passed word.
    # Uses NFA.epsilon_closure()
    def recognize(self, word):
        # ... description of the function ...
```

ε - Nondeterministic finite automaton (ε-NFA)

Implementation of the closure operation

```
class eNFA(NFA):
    def epsilon_closure(self, states):
        closure = set(states)  # initialize the set of closure states
        stack = list(states)   # initialize a LIFO-stack of states

        while stack:           # DFS-like traversal of transitions
            state = stack.pop()
            if self.epsilon not in self.transition_function[state]:
                continue
            for next_state in self.transition_function[state][self.epsilon]:
                if next_state not in closure:
                    closure.add(next_state)
                    stack.append(next_state)

        return closure
```

ε - Nondeterministic finite automaton (ε-NFA)

Implementation of the recognition mechanism

```
class eNFA(NFA):
    def recognize(self, word):
        current_states = self.epsilon_closure({self._start_state})

        for symbol in word:
            next_states = set()
            for state in current_states:
                if symbol in self.transition_function[state]:
                    next_states.update(self.transition_function[state][symbol])
            next_states.update(next_states)
            current_states = self.epsilon_closure(next_states)

            if not current_states:
                return False

        return any(state in self._accept_states for state in current_states)
```

ε - Nondeterministic finite automaton (ε-NFA)

Example of algorithm operation

```python
enfa = eNFA()
enfa._start_state = "q0"
enfa._accept_states = {"q1", "q2"}
enfa.update_transition([
    "q0>eps>q1", "q0>eps>q2",
    "q1>1>q3", "q2>1>q4",
    "q2>eps>q5", "q3>1>q1",
    "q4>1>q6", "q6>1>q2"
])

print(enfa)
print((f"The word 111111 is recognized:
↪  {enfa.recognize('111111')}")
print(f"The word 11111 is recognized:
↪  {enfa.recognize('11111')}")
```

```
<class 'automata.eNFA'>
States: {'q3', 'q4', 'q1', 'q5', 'q6',
↪  'q2', 'q0'}
Start state: q0
Alphabet: {'1'}
Accept states: {'q1', 'q2'}
Transition function:
q0: {'eps': {'q1', 'q2'}}
q1: {'1': {'q3'}}
q2: {'1': {'q4'}, 'eps': {'q5'}}
q3: {'1': {'q1'}}
q4: {'1': {'q6'}}
q6: {'1': {'q2'}}
The word 111111 is recognized: True
The word 11111 is recognized: False
```

Elimination of ε-moves

Theorem 4.1 (about constructing an equivalent DFA)

For any nondeterministic finite automaton with ε-moves $\mathcal{A}$ there exists an equivalent nondeterministic finite automaton $\mathcal{A}'$ such that $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}')$.

Elimination of ε-moves

Theorem 4.1 (about constructing an equivalent DFA)

For any nondeterministic finite automaton with ε-moves $\mathcal{A}$ there exists an equivalent nondeterministic finite automaton $\mathcal{A}'$ such that $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}')$.

For any state $q \in Q$, we define its closure

$$Orb(q) = \{q\} \cup \{p \in Q \mid \text{ in } \mathcal{A} \text{ there exists a path } q \xrightarrow{\varepsilon} q_1 \xrightarrow{\varepsilon} \ldots \xrightarrow{\varepsilon} q_k \xrightarrow{\varepsilon} p\},$$

Elimination of ε-moves

Theorem 4.1 (about constructing an equivalent DFA)

For any nondeterministic finite automaton with ε-moves $\mathcal{A}$ there exists an equivalent nondeterministic finite automaton $\mathcal{A}'$ such that $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}')$.

For any state $q \in Q$, we define its closure

$$Orb(q) = \{q\} \cup \{p \in Q \mid \text{ in } \mathcal{A} \text{ there exists a path } q \xrightarrow{\varepsilon} q_1 \xrightarrow{\varepsilon} \ldots \xrightarrow{\varepsilon} q_k \xrightarrow{\varepsilon} p\},$$

Then $\mathcal{A}' = (Q, \Sigma, \Delta', q_0, F')$ - NFA, where:

Elimination of ε-moves

Theorem 4.1 (about constructing an equivalent DFA)

For any nondeterministic finite automaton with ε-moves $\mathcal{A}$ there exists an equivalent nondeterministic finite automaton $\mathcal{A}'$ such that $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}')$.

For any state $q \in Q$, we define its closure

$$Orb(q) = \{q\} \cup \{p \in Q \mid \text{ in } \mathcal{A} \text{ there exists a path } q \xrightarrow{\varepsilon} q_1 \xrightarrow{\varepsilon} \dots \xrightarrow{\varepsilon} q_k \xrightarrow{\varepsilon} p\},$$

Then $\mathcal{A}' = (Q, \Sigma, \Delta', q_0, F')$ - NFA, where:

- For every state $q \in Q$ and symbol $a \in \Sigma$, the transition function Δ' is defined as $\Delta'(q, a) = \bigcup\limits_{p \in Orb(q)} \Delta(p, a)$, where $Orb(q)$ – is the closure of state q.

Elimination of ε-moves

Theorem 4.1 (about constructing an equivalent DFA)

For any nondeterministic finite automaton with ε-moves $\mathcal{A}$ there exists an equivalent nondeterministic finite automaton $\mathcal{A}'$ such that $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}')$.

For any state $q \in Q$, we define its closure

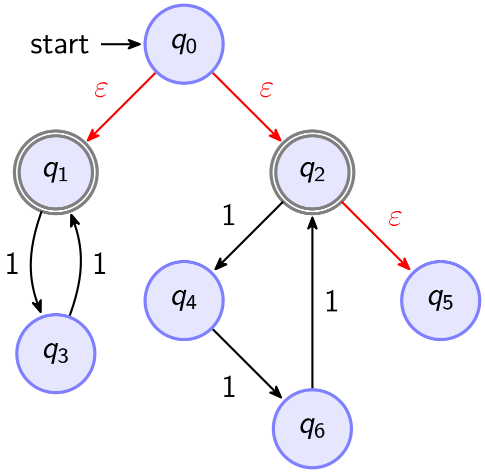
$$Orb(q) = \{q\} \cup \{p \in Q \mid \text{ in } \mathcal{A} \text{ there exists a path } q \xrightarrow{\varepsilon} q_1 \xrightarrow{\varepsilon} \ldots \xrightarrow{\varepsilon} q_k \xrightarrow{\varepsilon} p\},$$

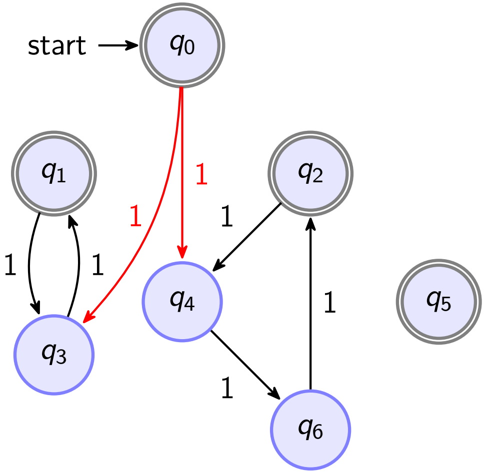
Then $\mathcal{A}' = (Q, \Sigma, \Delta', q_0, F')$ - NFA, where:

- For every state $q \in Q$ and symbol $a \in \Sigma$, the transition function Δ' is defined as $\Delta'(q, a) = \bigcup\limits_{p \in Orb(q)} \Delta(p, a)$, where $Orb(q)$ – is the closure of state q.
- The set of selected states F' is defined as $F' = \{q \in Q \mid Orb(q) \cap F \neq \varnothing\}$.

Elimination of ε-moves

Example of ε - moves elimination





Elimination of ε-moves

Implementation

```
class eNFA(NFA):
    def eliminate_epsilon(self):
        nfa = NFA()
        nfa.start_state = self.start_state
        nfa.accept_states = set()
        for state in self.states:
            orbit = self.epsilon_closure({state})
            for satellite in orbit:
                for symbol in self.transition_function[satellite]:
                    if symbol == self.epsilon:
                        continue
                    next_states = self.transition_function[satellite][symbol]
                    for next_state in next_states:
                        nfa.add_transition(f"{state}>{symbol}>{next_state}")
            if orbit & self.accept_states != set():
                nfa.accept_states |= orbit
```

NFA determinization

Theorem 5.1 (about constructing an equivalent DFA)

For any NFA $\mathcal{A}$ there exists a DFA $\mathcal{A}'$ such that $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}')$.

NFA determinization

Theorem 5.1 (about constructing an equivalent DFA)

For any NFA $\mathcal{A}$ there exists a DFA $\mathcal{A}'$ such that $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}')$.

Let $\mathcal{A} = (Q, \Sigma, \Delta, q_0, F)$ — be an NFA. We define the DFA $\mathcal{A}' = (Q', \Sigma, \delta', \{q_0\}, F')$ in the following way:

NFA determinization

Theorem 5.1 (about constructing an equivalent DFA)

For any NFA $\mathcal{A}$ there exists a DFA $\mathcal{A}'$ such that $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}')$.

Let $\mathcal{A} = (Q, \Sigma, \Delta, q_0, F)$ — be an NFA. We define the DFA $\mathcal{A}' = (Q', \Sigma, \delta', \{q_0\}, F')$ in the following way:

(a) $Q' = \mathcal{P}(Q)$ — the power set of Q;

NFA determinization

Theorem 5.1 (about constructing an equivalent DFA)

For any NFA $\mathcal{A}$ there exists a DFA $\mathcal{A}'$ such that $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}')$.

Let $\mathcal{A} = (Q, \Sigma, \Delta, q_0, F)$ — be an NFA. We define the DFA $\mathcal{A}' = (Q', \Sigma, \delta', \{q_0\}, F')$ in the following way:

(a) $Q' = \mathcal{P}(Q)$ — the power set of Q;

(b) $\delta'(q', a) = \bigcup_{r \in q'} \Delta(r, a)$, where $q' \in Q'$ and $a \in \Sigma$;

NFA determinization

Theorem 5.1 (about constructing an equivalent DFA)

For any NFA $\mathcal{A}$ there exists a DFA $\mathcal{A}'$ such that $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}')$.

Let $\mathcal{A} = (Q, \Sigma, \Delta, q_0, F)$ — be an NFA. We define the DFA $\mathcal{A}' = (Q', \Sigma, \delta', \{q_0\}, F')$ in the following way:

(a) $Q' = \mathcal{P}(Q)$ — the power set of Q;

(b) $\delta'(q', a) = \bigcup_{r \in q'} \Delta(r, a)$, where $q' \in Q'$ and $a \in \Sigma$;

(c) The initial state $q_0' = \{q_0\}$;

NFA determinization

Theorem 5.1 (about constructing an equivalent DFA)

For any NFA $\mathcal{A}$ there exists a DFA $\mathcal{A}'$ such that $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}')$.

Let $\mathcal{A} = (Q, \Sigma, \Delta, q_0, F)$ — be an NFA. We define the DFA $\mathcal{A}' = (Q', \Sigma, \delta', \{q_0\}, F')$ in the following way:

(a) $Q' = \mathcal{P}(Q)$ — the power set of Q;

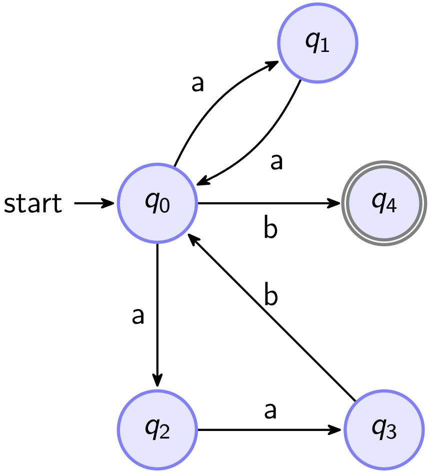
(b) $\delta'(q', a) = \bigcup_{r \in q'} \Delta(r, a)$, where $q' \in Q'$ and $a \in \Sigma$;

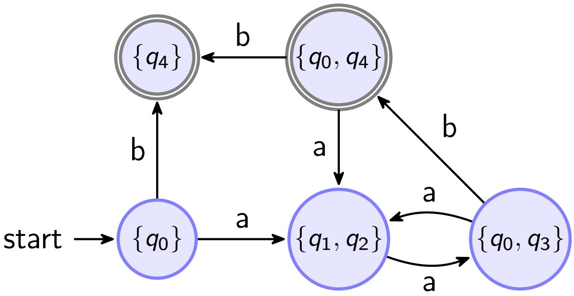
(c) The initial state $q_0' = \{q_0\}$;

(d) $F' = \{q' \in Q' \mid q' \cap F \neq \varnothing\}$.

NFA determinization

Example of determinization





NFA determinization

Example of algorithm operation

```
nfa = NFA()
nfa._start_state = "q0"
nfa._accept_states = {"q4"}
nfa.update_transition([
    "q0>a>q1", "q0>a>q2", "q0>b>q4",
    "q1>a>q0",
    "q2>a>q3",
    "q3>b>q0",
])

print(nfa.determinize())
```

```
<class 'automata.DFA'>
States: {'q4', 'q3,q0', 'q0', 'q4,q0',
↪   'q1,q2'}
Start state: q0
Alphabet: {'b', 'a'}
Accept states: {'q4', 'q4,q0'}
Transition function:
q0: {'b': 'q4', 'a': 'q1,q2'}
q1,q2: {'a': 'q3,q0'}
q3,q0: {'b': 'q4,q0', 'a': 'q1,q2'}
q4,q0: {'b': 'q4', 'a': 'q1,q2'}
```

DFA minimization

Definition 6.1

Let $\mathcal{D} = (Q, \Sigma, \delta, q_0, F)$ be a DFA (the definition for NFA and ε-NFA is analogous). The automaton **reversed** relative to $\mathcal{D}$ is called such an automaton $\mathcal{D}^{-1} = (Q, \Sigma, \Delta, F, \{q_0\})$, where:

(a) Q and Σ remain unchanged;

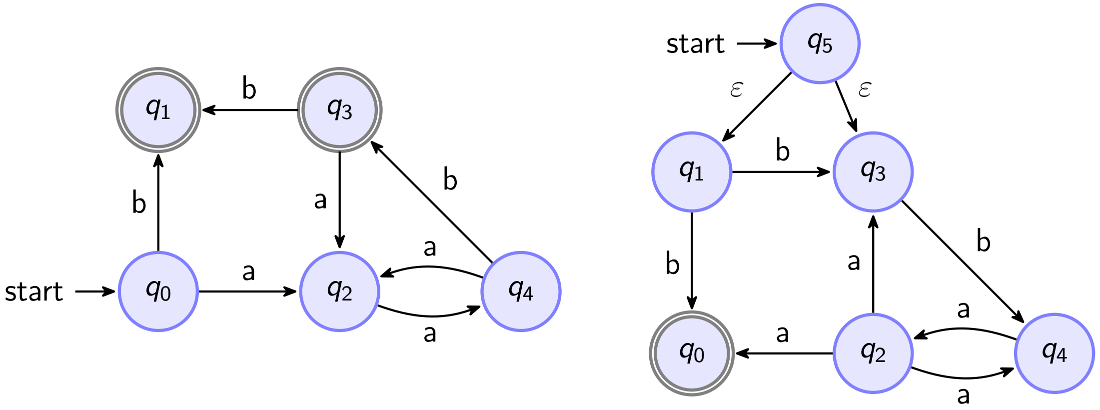
(b) The transition function Δ is defined as $\Delta(q, a) = \{p \in Q \mid \delta(p, a) = q\}$ for all $q \in Q$ and $a \in \Sigma$;

(c) The set of initial states $\mathcal{D}^{-1}$ corresponds to the set of accept states F of the automaton $\mathcal{D}$;

(d) The set of accept states $\mathcal{D}^{-1}$ consists only of the initial state q_0 of the automaton $\mathcal{D}$.

DFA minimization

Reversal of the automaton



DFA minimization

Automaton reversal implementation

```python
class DFA:
    def reverse(self):
        enfa = eNFA()
        enfa.alphabet = self.alphabet

        terminal_state = f"q{len(self.states)}"
        enfa.start_state = terminal_state
        enfa.accept_states = {self.start_state}

        for state in self.states:
            for symbol, next_state in self.transition_function[state].items():
                enfa.add_transition(f"{next_state}>{symbol}>{state}")
            if state in self.accept_states:
                enfa.add_epsilon_transition(terminal_state, state)

        return enfa
```

Theorem 6.2

Let $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ — be a deterministic finite automaton. **The Brzozowski minimization** is defined as follows:

1. Define the reversal operator r, such that for a given automaton $\mathcal{A}$, the automaton $r(\mathcal{A})$ is the reversed automaton of $\mathcal{A}$.
2. Define the determinization operator d, such that for any nondeterministic finite automaton $\mathcal{A}'$, the automaton $d(\mathcal{A}')$ is the deterministic automaton $\mathcal{A}'$.
3. The minimized automaton is obtained as $drdr(\mathcal{A})$.

DFA minimization

Brzozowski's Algorithm implementation

```
class DFA:
    def minimize(self):
        r = self.reverse()
        dr = r.determinize()
        rdr = dr.reverse()
        drdr = rdr.determinize()
        return drdr
```

Bibliography

1. Orlenko I. automata_builder GitHub repository. URL: https://github.com/ihorlenko/automata_builder.
2. Hopcroft J. E. Introduction to automata theory, languages, and computation. Reading, Mass : Addison-Wesley, 1979. 418 p.
3. Hopcroft J. E. Formal languages and their relation to automata. Reading, Mass : Addison-Wesley Pub. Co., 1969. 242 p.
4. Sipser. Introduction to the Theory of Computation. Cengage India, 2014. 458 p.
5. H. Rodger S. JFLAP. Version 7.1. 2018. URL: https://www.jflap.org
6. Теорія цифрових автоматів та формальних мов. Вступний курс / С. Гавриленко та ін. Харків, 2011. 176 с.
7. Биков М., Черв‘яков В. Дискретний аналіз і теорія автоматів : Навч. посіб. Суми, 2016.