MINISTRY OF EDUCATION, CULTURE AND RESEARCH OF REPUBLIC OF MOLDOVA
TECHNICAL UNIVERSITY OF MOLDOVA
FACULTY OF COMPUTERS, INFORMATICS AND MICROELECTRONICS
DEPARTMENT OF SOFTWARE ENGINEERING AND AUTOMATICS

Formal Languages and Finite Automata

Laboratory work 2 : Determinism in Finite Automata. Conversion from NDFA 2 DFA. Chomsky Hierarchy.

Variant 13

Elaborated:
st.gr. FAF-211 Gazea Sandu
Verified:
asist.univ. Vasile Drumea

Chișinău, 2023

Content

Introduction

A finite automaton is a device that represents various processes and is similar to a state machine in both structure and function. The term ”finite” implies that an automaton has a starting and a set of final states, indicating that the process being modeled has a defined beginning and end. However, certain automata structures can result in non-determinism, where multiple states can be reached with a single transition. Determinism refers to how predictable a system is, and if random variables are present, the system becomes stochastic or non-deterministic. Automata can be classified as non-deterministic or deterministic, and it is possible to achieve determinism by using algorithms to modify the automaton's structure. Finite automata are widely used in computer science and mathematics for modeling and analyzing different kinds of processes. They are used in various applications such as regular expressions, lexical analysis, compilers, pattern recognition, and more. The finite automata theory is an essential part of computer science and helps in understanding the computational complexity of different algorithms.

A deterministic finite automaton (DFA) is an automaton where every input symbol leads to a unique state transition. On the other hand, a non-deterministic finite automaton (NFA) is an automaton where a given input symbol can lead to multiple state transitions, causing non-determinism. While DFAs are more intuitive and easy to understand, NFAs are often used in more complex situations where a DFA would require too many states to model the process accurately.

There are various algorithms and techniques available to convert an NFA to a DFA, such as the subset construction algorithm and the powerset construction algorithm. These algorithms are used to transform an NFA into an equivalent DFA with a reduced number of states, thus achieving determinism.

In summary, finite automata are powerful tools that have many real-world applications, and their study and analysis are essential for computer scientists and mathematicians alike. Understanding the differences between deterministic and non-deterministic automata and the algorithms for converting between them is crucial for designing efficient and effective algorithms for various tasks.

Objectives

1. Understand what an automaton is and what it can be used for.
2. Continuing the work in the same repository and the same project, the following need to be added: a. Provide a function in your grammar type/class that could classify the grammar based on Chomsky hierarchy.
   (a) For this you can use the variant from the previous lab.
3. According to your variant number (by universal convention it is register ID), get the finite automaton definition and do the following tasks:
   (a) Implement conversion of a finite automaton to a regular grammar.
   (b) Determine whether your FA is deterministic or non-deterministic.
   (c) Implement some functionality that would convert an NDFA to a DFA.
   (d) Represent the finite automaton graphically (Optional, and can be considered as a bonus point).

1 Implementation

1. Not much, just to mention that it would be enough for you to implement the project just to work with your specific variant. Of course, it would be gr8 if you could make it as generic as possible.
2. In order to show the execution you can implement a client class/type, which is just a ”Main” class/type in which you can instantiate the types/classes. Another approach would be to write unit tests if you are familiar with them.

1.1 Code:

```
import matplotlib.pyplot as plt
import networkx as nx
from Automaton import Automaton
from FiniteAutomaton import FiniteAutomaton
from Grammar import Grammars
class Main:
    # Initialize the Main class by setting up a grammar,
    def __init__(self):
        self.productions = {
            'S': ['aA', 'aB'],
            'A': ['bS'],
            'B': ['aC'],
            'C': ['a', 'bS'],
        }
        self.start_symbol = 'S'
        self.grammar = Grammars(self.productions, self.start_symbol)
        self.finite_automaton = self.grammar.to_finite_automaton()
        self.automaton = FiniteAutomaton

    # Generates strings from the grammar
    def generate_strings(self, num_strings):
        for i in range(num_strings):
            string = self.grammar.generate_string()
            print(string)
```

```
if __name__ == '__main__':
    main = Main()

    main.generate_strings(5)

    automatons = main.grammar.to_finite_automaton()

    # Define a finite automaton manually
    automaton = {
        'states': {'q0', 'q1', 'q2', 'q3', 'q4', 'q5'},
        'alphabet': {'a', 'b', 'c'},
        'transitions': {
        'q0': {'a': 'q1'},
        'q1': {'a': 'q4', 'b': 'q2', 'c': 'q5'},
        'q2': {'a': 'q3', 'b': 'q2', 'c': 'q5'},
        'q3': {'a': 'q3', 'b': 'q4', 'c': 'q5'},
        'q4': {'a': 'q3', 'b': 'q2', 'c': 'q5'},
        'q5': {'a': 'q5', 'b': 'q0', 'c': 'q5'}
    },
        'start_state': 'q0',
        'final_states': {'q3', 'q5'}
    }

    checker = FiniteAutomaton(automaton)

    checker.check_strings(['aaa', 'abaaa', 'ababaa', 'aa', 'abababa'])

    print(automatons)

automation = Automaton()

automation.states = ['q0', 'q1', 'q2', 'q3']
automation.alphabet = ['a', 'b']
automation.transitions = {
```

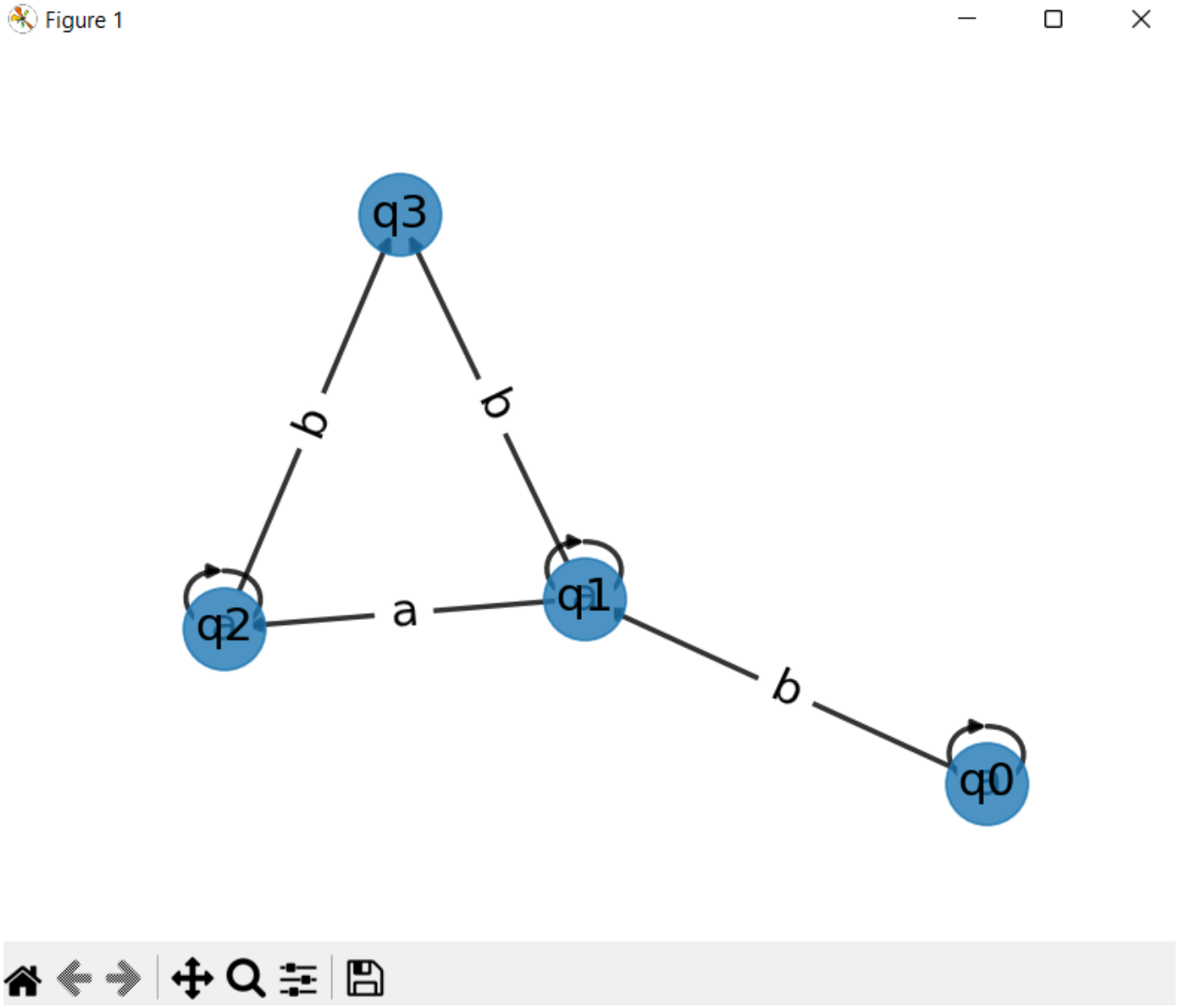
```
                    ('q0', 'a'): ['q0'],
                    ('q0', 'b'): ['q1'],
                    ('q1', 'a'): ['q1','q2'],
                    ('q1', 'b'): ['q3'],
                    ('q2', 'a'): ['q2'],
                    ('q2', 'b'): ['q3']}

automation.start_state = 'q0'
automation.accept_states = ['q3']
print('')
print('')
print('')
print('------------------------------------------------------------------
------------------------------------------------------------------
----------------------------------------')
print('
LAB2')
print('------------------------------------------------------------------
------------------------------------------------------------------
----------------------------------------')
is_deterministic = automation.is_deterministic()
print(f"Is automaton deterministic? {is_deterministic}")

# Convert NDFA to DFA
dfa = automation.to_dfa()
print(f"DFA states: {dfa.states}")
print(f"DFA transition function: {dfa.transitions}")
print(f"DFA initial state: {dfa.start_state}")
print(f"DFA final states: {dfa.accept_states}")

# Convert automaton to regular grammar
grammar = automation.to_grammar()
print(f"Regular grammar productions: {grammar}")
print(main.grammar.chomsky_classification())
```

```
automation.render()
```

Above is atached only the main folder of the project,
all classes you always can find on my github:GSandu1.

1.2 Screenshot:

```
--------------------------------------------------------------------------------------------------------------------------------------------------------------
                                                                    LAB2
--------------------------------------------------------------------------------------------------------------------------------------------------------------
Is automaton deterministic? False
DFA states: {frozenset({'q0'}), frozenset({'q1'}), frozenset({'q3'}), frozenset({'q1', 'q2'})}
DFA transition function: {(frozenset({'q0'}), 'a'): frozenset({'q0'}), (frozenset({'q0'}), 'b'): frozenset({'q1'}), (frozenset({'q1'}), 'a'): frozenset({'q1', 'q2'}), (frozenset(
DFA initial state: q0
DFA final states: {frozenset({'q3'})}
Regular grammar productions: ('q0', {'q1': {'aS', 'b', 'bq0', 'aq1'}, 'q2': {'aS', 'b', 'aq2', 'aq1'}, 'S': {'aq0'}})
Type 1: Context-Sensitive Grammar
: frozenset({'q3'}), (frozenset({'q1', 'q2'}), 'a'): frozenset({'q1', 'q2'}), (frozenset({'q1', 'q2'}), 'b'): frozenset({'q3'})}
```



Conclusions

Automata theory is a crucial topic in computer science, dealing with abstract computing devices capable of generating and recognizing languages. Finite automata are a type of automaton frequently used to represent various processes.

Finite automata can be considered as mathematical models of machines that can transition between a finite number of states based on input symbols and their current state. They have a start state and a set of final states, which dictate the beginning and end of the modeled process. State machines and finite automata have similar structures, but the latter is designed specifically for processes that can be modeled using a finite number of states.

Non-determinism is a significant challenge when working with finite automata. It occurs when a single input symbol can cause multiple state transitions, leading to unpredictable system behavior. To address this, algorithms can modify the automaton structure to create a deterministic version. Determinism is a crucial concept in systems theory, defining the predictability of a system. If the system includes random variables, it becomes stochastic or non-deterministic. The structure and behavior of finite automata determine their classification as deterministic or non-deterministic.

Finite automata are a powerful tool in computer science, used to model various processes. Although non-determinism can cause unpredictability, algorithms can modify the automaton structure to make it deterministic. Understanding automata theory is essential in parsing, formal language theory, compilers, and other areas of computer science. Finite automata are a fundamental concept that can help us comprehend the nature of computation and the behavior of complex systems.