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 to DFA. Chomsky Hierarchy.

Elaborated:

st.gr. FAF-213 Mihailiuc Igor

Verified:

asist.univ. Vasile Drumea

Chișinău, 2023

Introduction

Automata theory is a subfield of computer science that investigates the behavior of abstract machines and their ability to recognize languages. Determinism is one of the core concepts in automata theory, which describes the property of a machine to move to a unique state for a given input symbol. Determinism is crucial in ensuring that computations are predictable and consistent, and it is a fundamental concept in computer science. In finite automata, converting from a nondeterministic finite automaton (NFA) to a deterministic finite automaton (DFA) is a crucial process. This conversion transforms an NFA that may have multiple transition paths for a given input symbol into an equivalent DFA that only moves to one state for a given input symbol. The conversion process typically involves creating a state transition table or diagram that maps each input symbol to a unique state, which eliminates ambiguity and ensures determinism. The Chomsky hierarchy is a classification of formal languages based on their generative power. The hierarchy organizes languages into four levels: regular languages, context-free languages, context-sensitive languages, and recursively enumerable languages. Regular languages are the simplest and can be recognized by finite automata, while recursively enumerable languages are the most complex and can be recognized by Turing machines. The Chomsky hierarchy provides a framework for understanding the expressive power and limitations of different types of formal languages, which has significant implications for natural language processing, computational linguistics, and artificial intelligence.

Therefore, in the context of automata theory, comprehending determinism, converting from NFAs to DFAs, and the Chomsky hierarchy is crucial. This knowledge has broad implications for computer science and related fields.

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):
      - You can use external libraries, tools or APIs to generate the figures/diagrams.
      - Your program needs to gather and send the data about the automaton and the lib/tool/API return the visual representation.

Implementation

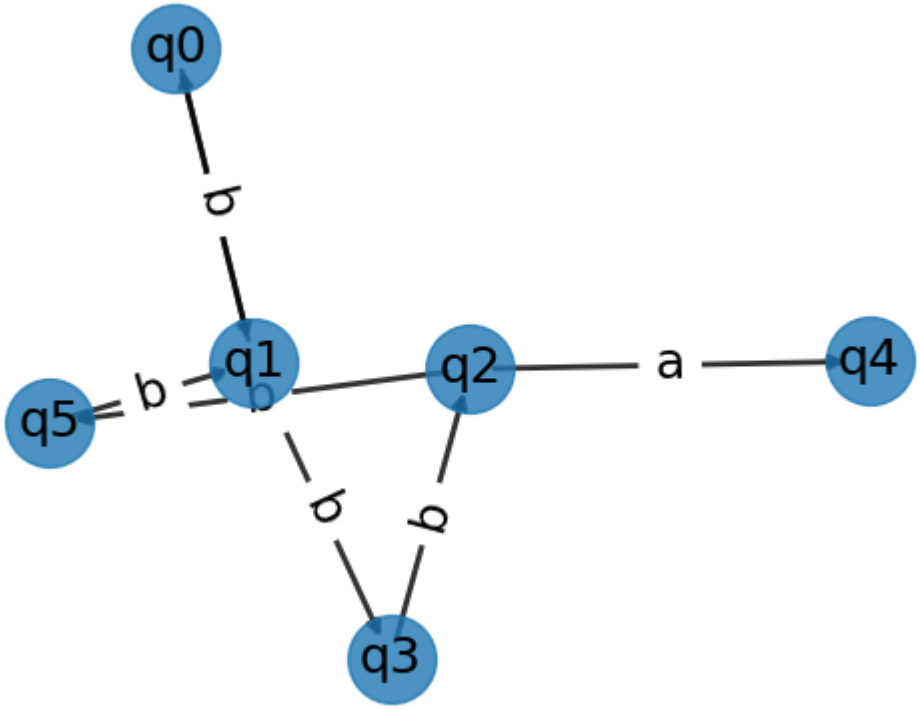
1. Not much, just to mention that it would be enough for you to implement the project just to work with your specific variant. Of course, it would be great if you could make it as generic as possible.
2. In order to show the execution you can implement a client class/type, which is just a ”Main” class/type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

Code:

```
import matplotlib.pyplot as plt
import networkx as nx
from Automaton import Automaton
from FiniteAutomaton import FiniteAutomaton
from Grammar import Grammars
class Main:
    print(
        '------------------------------------------------------------------------LAB1----------------
-------------------------------------------------------')


    def __init__(self):
        self.productions = {
            'S': ['aA'],
            'A': ['bS','bD'],
            'C': ['bA','a'],
            'D': ['bC','aD']
        }
        self.start_symbol = 'S'
        self.grammar = Grammars(self.productions, self.start_symbol)
        self.finite_automaton = self.grammar.to_finite_automaton()
        self.automaton = FiniteAutomaton


    def generate_strings(self, num_strings):
        for i in range(num_strings):
            string = self.grammar.generate_string()
            print(string)

if __name__ == '__main__':
    # Create a Main object
    main = Main()


    main.generate_strings(5)

    automatons = main.grammar.to_finite_automaton()

    automaton = {
        'states': {'q0', 'q1', 'q2', 'q3', 'q4', 'q5', 'q6'},
    'alphabet': {'a', 'b'},
    'transition': {
```

```
        'q0': {'a': 'q1'},
        'q1': {'b': 'q0', 'b': 'q3'},
        'q2': {'a': 'q4', 'b': 'q5'},
        'q3': {'b': 'q2', 'a': 'q3'},
        'q4': {'b': 'q2', 'a': 'q4'},
        'q5': {'a': 'q6', 'b': 'q1'},
        'q6': {'a': 'q3', 'b': 'q5'}

    },
    'start_state': 'q0',
    'final_states': {'q1', 'q2', 'q3', 'q4', 'q5', 'q6'}
    }

    checker = FiniteAutomaton(automaton)

    checker.check_strings(['aba', 'ababa', 'aaba', 'abbaba', 'abaaba'])

    print(automatons)

automation = Automaton()

automation.states = ['q0', 'q1', 'q2', 'q3']
automation.alphabet = ['a', 'b']
automation.transitions = {('q0', 'a'): ['q1'],
                          ('q1', 'b'): ['q0','q3'],
                          ('q2', 'a'): ['q4'],
                          ('q2', 'b'): ['q5'],
                          ('q3', 'b'): ['q2'],
                          ('q5', 'b'): ['q1']}
automation.start_state = 'q0'
automation.accept_states = ['q3']
print('')
print('')
print('')
print('-------------------------------------------------------------------------LAB2-----------------
-------------------------------------------------------')
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}")

grammar = automation.to_grammar()
print(f"Regular grammar productions: {grammar}")
print(main.grammar.chomsky_classification())
automation.render()
```

Screenshot:



Conclusions

Determinism is a crucial concept in automata theory and computer science, ensuring predictability and consistency in computation. The conversion from an NFA to a DFA is a critical process that eliminates ambiguity and ensures determinism, involving the construction of a state transition table or diagram. Understanding these concepts, along with the Chomsky hierarchy's language classification, is fundamental to computer science and has implications for natural language processing, computational linguistics, and artificial intelligence.