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.

Elaborated:

st.gr. FAF-211 Grama Alexandru

Verified:

asist.univ. Vasile Drumea

Chișinău, 2023

Content

Introduction

Automata theory is a branch of computer science that studies abstract machines and their ability to recognize languages. One of the central concepts in automata theory is determinism, which refers to the property of a machine to transition to only one state for a given input symbol. Determinism plays a crucial role in ensuring predictability and consistency in computation, making it a fundamental concept in computer science.

In finite automata, the conversion from a nondeterministic finite automaton (NFA) to a deterministic finite automaton (DFA) is a critical process. It involves transforming an NFA, which may have multiple transition paths for a given input symbol, into an equivalent DFA, which can transition to only one state for a given input symbol. The conversion process typically involves constructing a state transition table or diagram that maps each input symbol to a unique state, thus eliminating ambiguity and ensuring determinism.

The Chomsky hierarchy is a classification of formal languages based on their generative power. It categorizes 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 limitations and expressive power of different types of formal languages, which has significant implications for natural language processing, computational linguistics, and artificial intelligence.

In this context, understanding determinism, conversion from NFAs to DFAs, and the Chomsky hierarchy is crucial in automata theory and 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

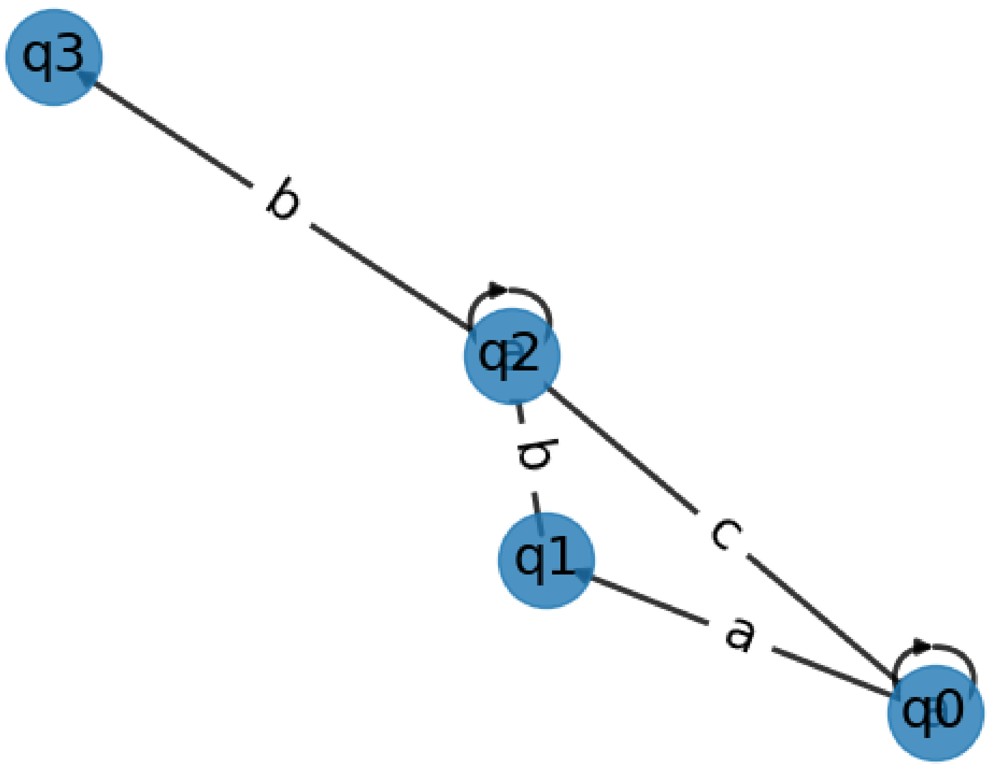
1. Not much, just to mention that it would be enough for you to implement the project just to work with your specific variant. Of course, it would be gr8 if you could make it as generic as possible. :)
2. In order to show the execution you can implement a client class/type, which is just a "Main" class/type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

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,
    # converting it to a finite automaton, and setting up a FiniteAutomaton object
    def __init__(self):
        self.productions = {
            'S': ['aS','bS','cA'],
            'A': ['aB',],
            'B': ['aB','bB','c'],
        }
        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__':
    # Create a Main object
    main = Main()

    # Generate and print 5 strings from the grammar
    main.generate_strings(5)

    # Convert the grammar to a finite automaton
    automatons = main.grammar.to_finite_automaton()

    # Define a finite automaton manually
    automaton = {
        'states': {'q0', 'q1', 'q2', 'q3', 'q4', 'q5', 'q6'},
        'alphabet': {'a', 'b', 'c'},
        'transition': {
            'q0': {'a': 'q1', 'b': 'q2', 'c': 'q3'},
            'q1': {'a': 'q1', 'b': 'q2', 'c': 'q3'},
            'q2': {'a': 'q4', 'b': 'q5', 'c': 'q6'},
            'q3': {'a': 'q1', 'b': 'q2', 'c': 'q3'},
            'q4': {'a': 'q4', 'b': 'q5', 'c': 'q6'},
            'q5': {'a': 'q4', 'b': 'q5', 'c': 'q6'},
            'q6': {'a': 'q4', 'b': 'q5', 'c': 'q6'}
        },
        'start_state': 'q0',
        'final_states': {'q1', 'q2', 'q3', 'q4', 'q5', 'q6'}
    }

    # Create a FiniteAutomaton object for the manually-defined automaton
    checker = FiniteAutomaton(automaton)

    # Check whether a list of strings are accepted by the finite automaton
    checker.check_strings(['aab', 'abcbb', 'bac', 'cab', 'ccaabb'])
    # Print the finite automaton
    print(automatons)

automation = Automaton()


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

# Check if automaton is deterministic
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()
```

Screenshot:



Conclusions

Determinism is a fundamental concept in computer science and automata theory. In finite automata, determinism refers to the property that a machine can transition to only one state for a given input symbol. The concept of determinism is essential because it allows for unique and unambiguous execution of a machine for a given input, thereby ensuring predictability and consistency in computation.

The conversion from a nondeterministic finite automaton (NFA) to a deterministic finite automaton (DFA) is a critical process in automata theory. It involves transforming an NFA, which may have multiple transition paths for a given input symbol, into an equivalent DFA, which can transition to only one state for a given input symbol. The conversion process typically involves constructing a state transition table or diagram that maps each input symbol to a unique state, thus eliminating ambiguity and ensuring determinism.

The Chomsky hierarchy is a classification of formal languages based on their generative power. It categorizes 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 study of automata theory and the concepts of determinism, nondeterminism, and conversion between NFAs and DFAs are fundamental in computer science and play a vital role in the design and analysis of algorithms, programming languages, and operating systems. The Chomsky hierarchy provides a framework for understanding the limitations and expressive power of different types of formal languages, which has significant implications for natural language processing, computational linguistics, and artificial intelligence.