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 1 : Regular Grammars and Finite Automata

Elaborated:
st.gr. FAF-211 Grama Alexandru
Verified:
asist.univ. Vasile Drumea

Chișinău, 2023

Content

Introduction

Regular Grammars and Finite Automata are fundamental concepts in theoretical computer science and play a crucial role in the study of formal languages and automata theory.

A Regular Grammar is a set of production rules used to generate strings in a formal language. A formal language is a set of strings made up of symbols from a given alphabet, and regular grammars describe a subset of those languages that can be recognized by Finite Automata. Regular grammars are defined by a set of production rules, which specify how to rewrite a symbol in the language to a string of other symbols. The production rules are applied repeatedly to generate all the valid strings in the language.

A Finite Automaton is a mathematical model used to recognize patterns in strings, and it consists of a set of states, an initial state, a set of accepting states, and a transition function that maps from a current state and input symbol to a next state. Finite automata are classified as either deterministic or non-deterministic, depending on whether or not multiple transitions are possible for a given input symbol and state.

The relationship between regular grammars and finite automata is very close, and they are often used interchangeably. In particular, every regular grammar can be associated with a deterministic finite automaton, and every deterministic finite automaton can be associated with a regular grammar.

Regular grammars and finite automata have numerous applications in computer science, including parsing, pattern matching, lexical analysis, and text processing. They provide a foundation for more advanced topics, such as context-free grammars, pushdown automata, and Turing machines, which are essential in the study of computational complexity and algorithm design.

Objectives

1. Understand what a language is and what it needs to have in order to be considered a formal one.
2. Provide the initial setup for the evolving project that you will work on during this semester. I said project because usually at lab works, I encourage/impose students to treat all the labs like stages of development of a whole project. Basically you need to do the following:
   (a) Create a local remote repository of a VCS hosting service (let us all use Github to avoid unnecessary headaches);
   (b) Choose a programming language, and my suggestion would be to choose one that supports all the main paradigms;
   (c) Create a separate folder where you will be keeping the report. This semester I wish I won't see reports alongside source code files, fingers crossed;
3. According to your variant number (by universal convention it is register ID), get the grammar definition and do the following tasks:
   (a) Implement a type/class for your grammar;
   (b) Add one function that would generate 5 valid strings from the language expressed by your given grammar;
   (c) Implement some functionality that would convert and object of type Grammar to one of type Finite Automaton;
   (d) For the Finite Automaton, please add a method that checks if an input string can be obtained via the state transition from it;

1 Implementation

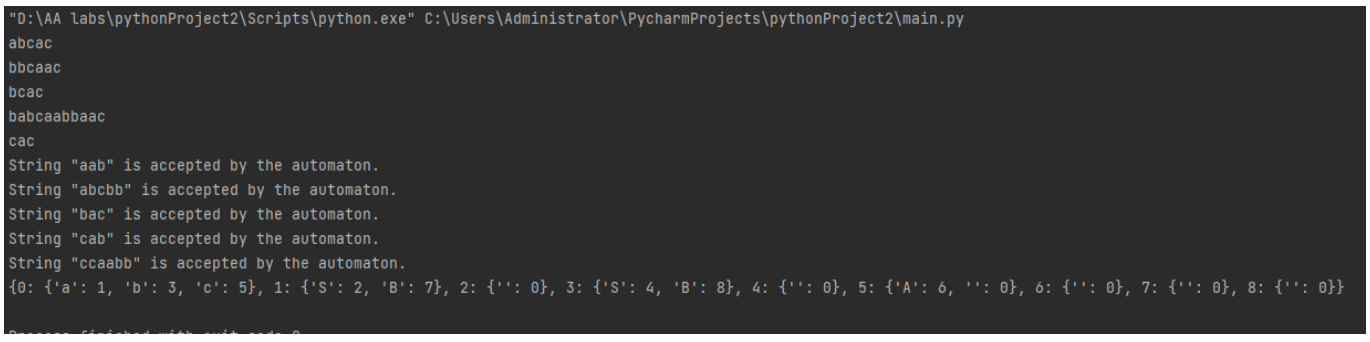
1. You can use 2 classes in order to represent the 2 main object which are the grammar and finite automaton. Additional data model, helper classes etc. can be added but should be used (i.e. you shouldn't have source code file that are not used).
2. In order to show the execution you can implement a client 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:

```
from FiniteAutomaton import FiniteAutomaton
from Grammar import Grammars
class Main:


    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
    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()
```

```
automaton = {
    'states': {'q0', 'q1', 'q2', 'q3', 'q4', 'q5', 'q6'},
'alphabet': {'a', 'b', 'c'},
'transitions': {
    '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'}
}
checker = FiniteAutomaton(automaton)
checker.check_strings(['aab', 'abcbb', 'bac', 'cab', 'ccaabb'])
print(automatons)
```

1.2 Screenshot:



Conclusions

Regular grammars and finite automata are essential tools for understanding the capabilities and limitations of computers. By studying regular languages and automata, computer scientists can design efficient algorithms for tasks such as pattern matching, text processing, and lexical analysis. These algorithms are used in many real-world applications, such as search engines, spam filters, and data compression. Furthermore, the ability to recognize and generate regular languages is crucial for computer systems to communicate with each other in a standardized way, which is a crucial aspect of network communication.

The study of regular grammars and finite automata also has theoretical implications for computer science. By understanding the limitations of these models, computer scientists can develop new models of computation that are more powerful and can solve more complex problems. For example, context-free grammars and pushdown automata extend the capabilities of regular grammars and finite automata by allowing more complex patterns to be recognized. These models, in turn, provide the basis for more advanced models, such as Turing machines, which can solve any problem that can be solved by a computer.

In addition to their applications in computer science, regular grammars and finite automata have connections to other areas of mathematics, including group theory, topology, and algebra. These connections provide insight into the nature of computation and its relationship to other mathematical concepts.

In conclusion, regular grammars and finite automata are foundational concepts in computer science and have numerous practical applications in many areas, including natural language processing, compilers, and artificial intelligence. They are also essential tools for understanding the capabilities and limitations of computers and have connections to other areas of mathematics. Their study continues to be an active area of research, and they will undoubtedly play a crucial role in shaping the future of computer science.