**Ministry of Education, Culture, and Research of the Republic of Moldova**

**Technical University of Moldova**

**The Faculty of Computers, Informatics, and Microelectronics**

**REPORT**

Laboratory work no.1

*Formal Languages & Finite Automata*

Executed by:

st. gr. FAF-222 Șarov Andrei

Verified by:

univ. assist. Crețu Dumitru

Chişinău - 2024

**Objectives**

1. Discover 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. You can deal with each laboratory work as a separate task or project to demonstrate your understanding of the given themes, but you also can deal with labs as stages of making your own big solution, your own project. Do the following:
   * a. Create GitHub repository to deal with storing and updating your project;
   * b. Choose a programming language. Pick one that will be easiest for dealing with your tasks, you need to learn how to solve the problem itself, not everything around the problem (like setting up the project, launching it correctly and etc.);
   * c. Store reports separately in a way to make verification of your work simpler (duh)
3. According to your variant number, get the grammar definition and do the following:
   * 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;

**Variant 26:**

VN={S, A, B, C},

VT={a, b, c, d},

P={

S → dA

A → aB

B → bC

C → cB

B → d

C → aA

A → b

}

**Code:**

import random

class Grammar:

def \_\_init\_\_(self):

# Define production rules for the grammar

self.P = {

'S': ['dA'],

'A': ['aB', 'b'],

'B': ['bC', 'd'],

'C': ['cB', 'aA']

}

def generate\_string(self, symbol, length):

# Recursive function to generate strings based on the grammar

if length == 0:

return ""

production = random.choice(self.P.get(symbol, ['']))

result = ""

for s in production:

result += self.generate\_string(s, length - 1)

return result

def generate\_valid\_strings(self, count, length):

# Generate a specified number of valid strings

valid\_strings = []

for \_ in range(count):

valid\_strings.append(self.generate\_string('S', length))

return valid\_strings

class FiniteAutomaton:

def \_\_init\_\_(self, grammar):

# Convert grammar into a Finite Automaton

self.transitions = {}

for state in grammar.P:

self.transitions[state] = {}

for prod in grammar.P[state]:

if len(prod) > 1:

self.transitions[state][prod[1]] = prod[0]

def can\_generate\_string(self, string):

# Check if a given string is valid based on the Finite Automaton

current\_state = 'S'

for symbol in string:

if symbol not in self.transitions[current\_state]:

return False

current\_state = self.transitions[current\_state].get(symbol, None)

if current\_state is None:

return False

return current\_state == 'S'

# Example usage

grammar = Grammar()

automaton = FiniteAutomaton(grammar)

print("Valid Strings:")

valid\_strings = grammar.generate\_valid\_strings(5, 10)

for s in valid\_strings:

print(s)

print("\nChecking if strings are valid via Finite Automaton:")

for s in valid\_strings:

print(f"'{s}' is valid:", automaton.can\_generate\_string(s))

**Conclusion:**

In this laboratory assignment, we successfully tackled Variant 26 by developing a Python solution that explores formal languages and automata theory. The code comprises a Grammar class that represents the given grammar, facilitating the generation of valid strings adhering to its rules. Additionally, there was implemented a Finite Automaton class, enabling the conversion of the grammar into an automaton for string validation. Through meticulous commenting, It was ensured the code's comprehensibility and ease of understanding. This exercise not only provided practical insight into formal language concepts but also laid a strong foundation for future endeavors in this field.