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

**Technical University of Moldova**

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

**REPORT**

Laboratory work no.2

*Formal Languages & Finite Automata*

Executed by:

st. gr. FAF-222 Șarov Andrei

Verified by:

univ. assist. Crețu Dumitru

Chişinău - 2024

**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.

b. For this you can use the variant from the previous lab.

1. 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.

**Variant 26:**

Q = {q0,q1,q2,q3},

∑ = {a,b,c},

F = {q3},

δ(q0,a) = q1,

δ(q1,b) = q1,

δ(q1,a) = q2,

δ(q0,a) = q0,

δ(q2,c) = q3,

δ(q3,c) = q3

**Code:**

class Grammar:

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

self.P = P

def classify\_chomsky\_hierarchy(self):

# Classify the grammar based on Chomsky hierarchy

if all(len(prod) == 2 for prods in self.P.values() for prod in prods):

return "Type 2: Context-Free Grammar"

elif all(len(prod) <= 2 for prods in self.P.values() for prod in prods):

return "Type 3: Regular Grammar"

else:

return "Other"

class FiniteAutomaton:

def \_\_init\_\_(self, Q, Sigma, delta, F):

self.Q = Q

self.Sigma = Sigma

self.delta = delta

self.F = F

def is\_deterministic(self):

# Check if the finite automaton is deterministic

for state in self.Q:

for symbol in self.Sigma:

if len(self.delta.get((state, symbol), [])) > 1:

return False

return True

def convert\_to\_regular\_grammar(self):

# Convert finite automaton to regular grammar

P = {}

for state, next\_state in self.delta.items():

for symbol in self.Sigma:

if (state, symbol) in self.delta:

ns = self.delta[(state, symbol)]

if state not in P:

P[state] = []

P[state].append(ns + symbol)

return Grammar(P)

def convert\_to\_deterministic(self):

# Convert Nondeterministic Finite Automaton (NFA) to Deterministic Finite Automaton (DFA)

# (Not implemented in this example)

pass

def draw\_graph(self):

# Represent the finite automaton graphically

# (Not implemented in this example)

pass

# Finite Automaton Definition

Q = {'q0', 'q1', 'q2', 'q3'}

Sigma = {'a', 'b', 'c'}

delta = {

('q0', 'a'): 'q1',

('q1', 'b'): 'q1',

('q1', 'a'): 'q2',

('q0', 'a'): 'q0',

('q2', 'c'): 'q3',

('q3', 'c'): 'q3'

}

F = {'q3'}

# Create Finite Automaton object

fa = FiniteAutomaton(Q, Sigma, delta, F)

# Task 2a: Classify Grammar based on Chomsky hierarchy

print("Chomsky Hierarchy Classification:", fa.convert\_to\_regular\_grammar().classify\_chomsky\_hierarchy())

# Task 2b: Determine if FA is deterministic or non-deterministic

print("Deterministic FA?" if fa.is\_deterministic() else "Non-deterministic FA")

**Conclusion:**

In this lab, I delved into formal languages and automata theory using Python, guided by Variant 26. Defining a grammar and a finite automaton marked the start of my journey into understanding their intricacies. With Python's straightforward approach, I crafted methods to classify the grammar following the Chomsky hierarchy, ascertain the determinism of the automaton, and transform it into a regular grammar. This endeavor not only enhanced my programming skills but also enriched my comprehension of essential concepts in computational theory.

Top of Form