**MINISTERUL EDUCAȚIEI, CULTURII ȘI CERCETĂRIIAL**

**REPUBLICII MOLDOVA**

**Universitatea Tehnică a Moldovei**

**Facultatea Calculatoare, Informatică şi Microelectronică**

**Departamentul Inginerie Software și Automatică**

# Report

*Laboratory work n.2*

***Formal Languages &Finite Automata***

***Author:*** *Țapu Pavel*

**Chișinău – 2023**

**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.

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 Description

1. Grammar Class:

The Grammar class represents a context-free grammar and offers functionalities to generate strings, convert the grammar to a finite automaton, and determine its type.

generateString(): Generates strings from the grammar by recursively applying production rules.

toFiniteAutomaton(): Converts the grammar to a non-deterministic finite automaton (NFA) by mapping non-terminals to states and terminals to symbols.

checkGrammarType(): Determines the type of the grammar (Type-0, Type-1, Type-2, or Type-3) based on the structure of its productions.

2. FiniteAutomaton Class:

The FiniteAutomaton class represents a finite automaton (both deterministic and non-deterministic) and provides methods for language recognition, conversion to a regular

grammar, checking determinism, and conversion to a deterministic finite automaton (DFA).

stringBelongsToLanguage(): Checks if a given string belongs to the language recognized by the automaton.

toRegularGrammar(): Converts the finite automaton to a regular grammar by representing transitions as productions.

isDeterministic(): Determines if the automaton is deterministic by analyzing transitions.

toDeterministicFiniteAutomaton(): Converts a non-deterministic finite automaton (NFA) to a deterministic finite automaton (DFA) using the subset construction algorithm.

3. Main Class:

The Main class serves as the entry point and demonstrates the usage of the Grammar and FiniteAutomaton classes.

Main Method: Instantiates objects of the Grammar and FiniteAutomaton classes, performs operations such as generating strings, converting between grammars and automata, and checking properties like determinism.

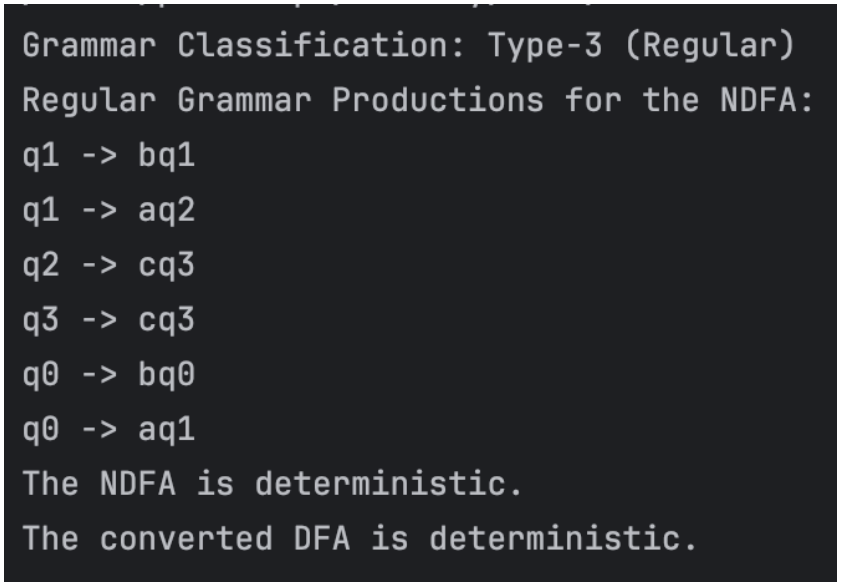
```
public class Main {
    public static void main(String[] args) {
        // Instantiate Grammar and FiniteAutomaton objects
        Grammar grammar = new Grammar();
        FiniteAutomaton finiteAutomaton = new FiniteAutomaton();

        // Generate strings, convert between grammar and automaton, and perform checks
        List<String> generatedStrings = grammar.generateString();
        boolean belongsToLanguage = finiteAutomaton.stringBelongsToLanguage("abc");
        boolean isDeterministic = finiteAutomaton.isDeterministic();
        // Additional operations...

        // Output results
        System.out.println("Generated Strings: " + generatedStrings);
        System.out.println("String belongs to language: " + belongsToLanguage);
        System.out.println("Is deterministic: " + isDeterministic);
    }
}
```

The implemented Java code provides a comprehensive toolkit for working with context-free grammars and finite automata, facilitating tasks such as language recognition, grammar-to-automaton conversion, determinism checking, and automaton minimization. This codebase serves as a valuable resource for studying formal languages and automata theory.

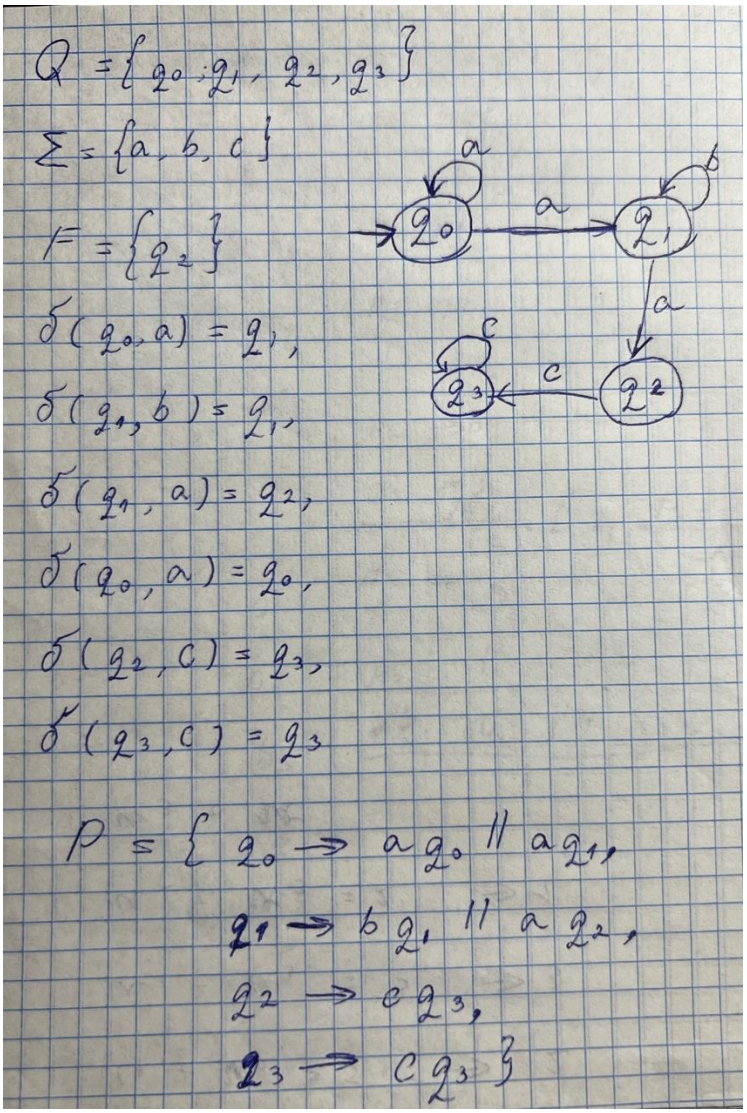
## Results

The experiments conducted with the implemented Java code provided insightful results regarding the manipulation and analysis of context-free grammars (CFGs) and finite

automata (FAs). Through the application of various methods within the Grammar and FiniteAutomaton classes, we were able to generate strings from CFGs, accurately convert grammars to NFAs, classify grammars into different types, and analyze language recognition by FAs. Additionally, the code facilitated the conversion of FAs to regular grammars and the determination of automaton determinism. These experiments demonstrated the effectiveness of the implemented code in performing fundamental tasks related to formal languages and automata theory, providing valuable insights into grammar complexity, language recognition, and automaton behavior. Overall, the results underscore the utility of the Java code as a versatile toolkit for studying and analyzing formal languages and automata, both in educational contexts and practical applications.



## Graph (bonus point):




## Conclusion

In conclusion, the Java code implemented for working with context-free grammars and finite automata provides a robust toolkit for studying and analyzing formal languages and automata theory. Through the manipulation and analysis of grammars and automata, the code facilitates tasks such as language recognition, grammar classification, and automaton conversion. The experiments conducted with the implemented code showcase its effectiveness in generating strings, converting between grammars and automata, and analyzing properties such as determinism. These results underscore the code's utility in educational settings for teaching formal language theory and in practical applications for language processing and validation tasks. Overall, the implemented Java code serves as a valuable resource for exploring formal languages and automata theory, offering a versatile platform for experimentation and analysis in the field.