MINISTERUL EDUCAȚIEI, CULTURII ȘI CERCETĂRII

AL REPUBLICII MOLDOVA

Universitatea Tehnică a Moldovei Facultatea Calculatoare, Informatică şi

Microelectronică Departamentul Inginerie Software și Automatică

GLIGA DANIELA, FAF-223

Report

Laboratory work nr.2

Determinism in Finite Automata.

Conversion from NDFA to DFA.

Chomsky Hierarchy.

of Formal Languages and Finite Automata

Checked by:

Dumitru Crețu, *university assistant*

FCIM, UTM

Chișinău – 2024

- **Theory:**

Finite Automata, a fundamental concept in theoretical computer science, provides a mathematical model for representing and analyzing computation. Determinism, the conversion from Non-Deterministic Finite Automata (NDFA) to Deterministic Finite Automata (DFA), and the Chomsky Hierarchy are interconnected topics that delve into the study of computational complexity and the expressive power of formal languages.

Determinism in Finite Automata: Deterministic Finite Automata (DFA) are a type of finite automaton where, given a current state and an input symbol, there is exactly one next state. This determinism simplifies the analysis of automata and ensures unique outcomes for any given input. Deterministic behavior aids in better understanding and predictable computation, making DFAs particularly suitable for certain types of applications such as lexical analysis in compilers.

Conversion from NDFA to DFA: Non-Deterministic Finite Automata (NDFA) allow multiple transitions from a state on the same input symbol, introducing non-deterministic choices. To analyze and compare the expressive power of NDFA and DFA, it is often necessary to convert NDFA to DFA. The subset construction algorithm is a commonly used method for this conversion. It involves creating a DFA whose states represent sets of states from the original NDFA. The resulting DFA preserves the language recognized by the NDFA while providing deterministic behavior, simplifying further analysis.

Chomsky Hierarchy: The Chomsky Hierarchy classifies formal languages into four types based on their generative power. These types are Type 3 (Regular Languages), Type 2 (Context-Free Languages), Type 1 (Context-Sensitive Languages), and Type 0 (Recursively Enumerable Languages). Each type is associated with a specific grammar type, with regular expressions corresponding to regular languages, context-free grammars corresponding to context-free languages, and so on.

The connection between NDFA to DFA conversion and the Chomsky Hierarchy lies in the equivalence between regular languages and languages recognized by finite automata. Regular languages, which can be recognized by both NDFA and DFA, fall within the first level of the Chomsky Hierarchy. The process of converting NDFA to DFA allows us to explore the limits and capabilities of regular languages and understand their place within the broader hierarchy.

- ## 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:

In order to implement checking the grammar according Chomsky Hierarchy , I have added to my grammar class the following methods, which are checking to which type my own variant belongs to.

```
public static String classifyGrammar() {
      if (isChomskyType3()) {
          return "Type 3";
      } else if (isChomskyType2()) {
          return "Type 2";
      } else if (isChomskyType1()) {
          return "Type 1";
      } else if (isChomskyType0()) {
          return "Type 0";
      } else {
          return "Unknown Type";
      }
  }

  private static boolean isChomskyType0() {
```

```
        // Specific checks for Type 0
        return true;
    }
```

Type 1 grammars, also referred to as Context-Sensitive grammars, represent a higher level of complexity than both Type 2 (Context-Free) and Type 3 (Regular) grammars. In Type 1 grammars, production rules take the form $\alpha \rightarrow \beta$, where α and β are strings of terminals and/or non-terminals, with the constraint that the length of α is less than or equal to the length of β, and β is not allowed to be an empty string.

```
    private static boolean isChomskyType1() {
        for (String production : P) {
            String[] parts = production.split("→");
            if (parts.length == 2) {
                String alpha = parts[0].trim();
                String beta = parts[1].trim();
                if (alpha.length() > beta.length() || beta.isEmpty()) {
                    return false;
                }
            }
        }
        return true;
    }
```

Type 2 grammars, known as Context-Free grammars, are more expressive than Type 3 grammars. They have production rules of the form $A \rightarrow \gamma$, where A is a non-terminal and γ is a string of terminals and/or non-terminals. Context-Free grammars are associated with context-free languages and are widely used in the representation of programming languages, natural language syntax, and various hierarchical structures. Parsing techniques such as LL and LR parsing are commonly employed for processing languages defined by Context-Free grammars.

```
    private static boolean isChomskyType2() {
        for (String production : P) {
            String[] parts = production.split("→");
            if (parts.length == 2) {
                String nonTerminal = parts[0].trim();
                String gamma = parts[1].trim();
                if (nonTerminal.length() == 1 && VNContains(nonTerminal) &&
isChomskyType2String(gamma)) {
                    continue;
                } else {
                    return false;
                }
            }
        }
        return true;
    }

    private static boolean isChomskyType2String(String str) {
        for (char c : str.toCharArray()) {
            if (!isTerminal(c) && !VNContains(String.valueOf(c))) {
                return false;
            }
        }
```

```
        return true;
    }
```

Type 3 grammars, also known as Regular grammars, are the simplest class in the Chomsky hierarchy. They have production rules of the form A → aB or A → a, where A and B are non-terminals, and a is a terminal. These grammars can be recognized and parsed efficiently by finite automata and are associated with regular languages, making them suitable for representing simple patterns and regular expressions.

```
    private static boolean isChomskyType3() {
        for (String production : P) {
            String[] parts = production.split("→");
            if (parts.length == 2) {
                String left = parts[0].trim();
                String right = parts[1].trim();
                if (left.length() == 1 && VNContains(left) && right.length() <= 2) {
                    if (right.length() == 1 && isTerminal(right.charAt(0))) {
                        continue;
                    } else if (right.length() == 2 && isTerminal(right.charAt(0)) &&
VNContains(String.valueOf(right.charAt(1)))) {
                        continue;
                    }
                }
                return false;
            }
        }
        return true;
    }

    private static boolean VNContains(String symbol) {
        return symbolTypes.containsKey(symbol.charAt(0)) &&
symbolTypes.get(symbol.charAt(0)).equals("Non-Terminal");
    }
}
```

Each method verifies the rules for a certain type and in the end the grammar is checked beginning with type 3. If the rules of type 3 grammar are present in my grammar then the type 3 is printed. In the same manner it checks the type 2, type 1and in the end type 0 , since it almost has no rules.

In the second part , I had to transform the given FA into Regular Grammar. This is how I have performed the transformation. Also I have done the illustration since it is much simpler for me to do the transormation in this way.

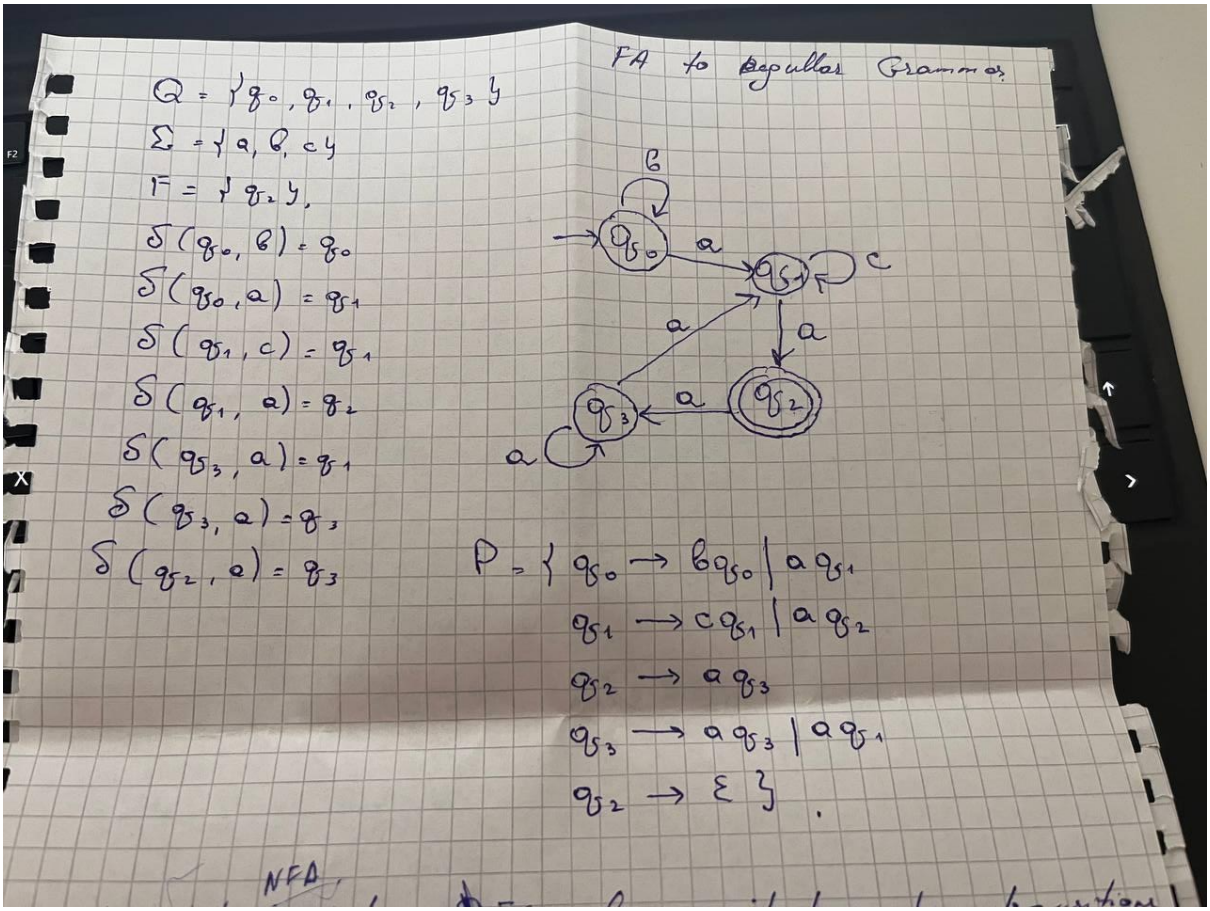
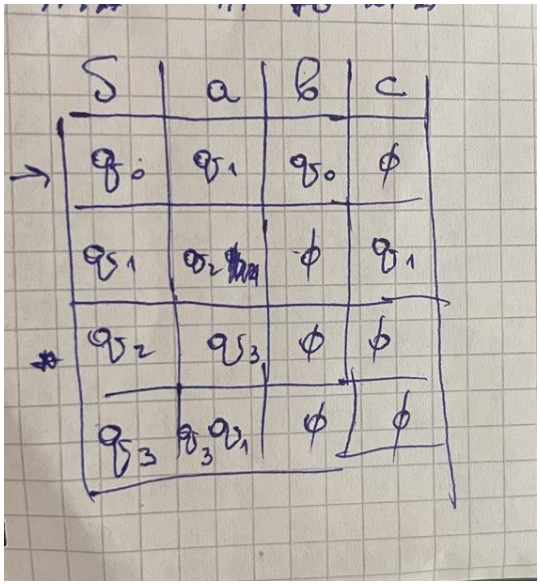


Figure 1. FA to Regular Grammar

This FA is a NFA because it has two transitions with the same variable from one state which leads to multiple states.

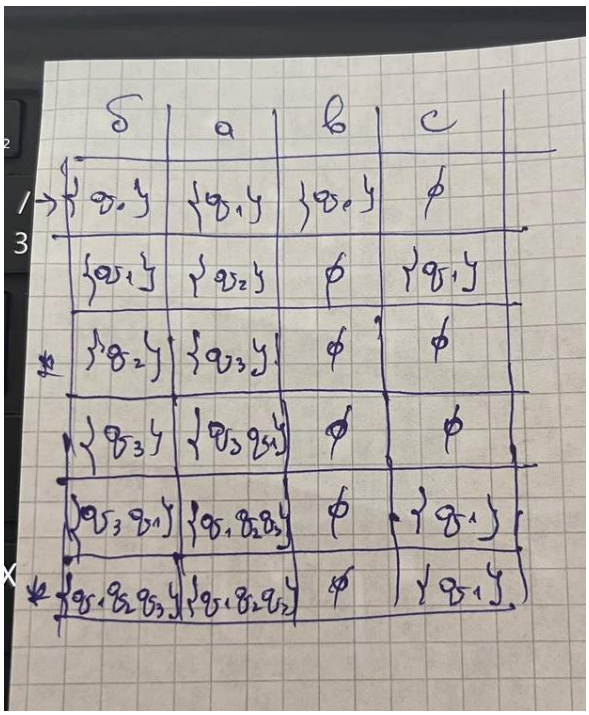
After determining it is a NFA I have done the transformation from NFA to DFA in the following way: Firstly, I transfered the FA into a basic table:



δ	a	b	c
→ q_0	q_1	q_0	ϕ
q_1	q_2	ϕ	q_1
* q_2	q_3	ϕ	ϕ
q_3	$q_3 q_1$	ϕ	ϕ

Figure 2. Table for NFA

Then, I made the proper transformation into another table, by adding the appearing new states and determining their paths.



δ	a	b	c
→ $\{q_0\}$	$\{q_1\}$	$\{q_0\}$	ϕ
$\{q_1\}$	$\{q_2\}$	ϕ	$\{q_1\}$
* $\{q_2\}$	$\{q_3\}$	ϕ	ϕ
$\{q_3\}$	$\{q_3 q_1\}$	ϕ	ϕ
$\{q_3 q_1\}$	$\{q_1 q_2 q_3\}$	ϕ	$\{q_1\}$
* $\{q_1 q_2 q_3\}$	$\{q_1 q_2 q_3\}$	ϕ	$\{q_1\}$

Figure 3. Table for obtained DFA

In the end, I have illustrated the grapth for obtained DFA:

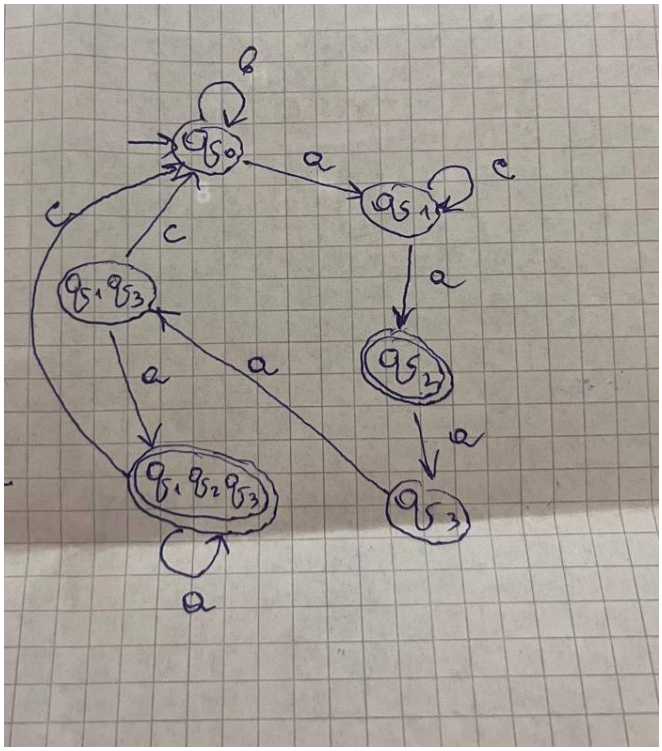


Figure 4. DFA grapth

- ## Conclusions:

In conclusion, after doing this laboratory work, I have trained my knowledge in the chomsky hirarchy, also I have a better understanding on how it works internally. Regarding finite automata and the transformation to regular grammar then to a DFA, it was interesting to understand through table transformation and graph illustartion how it works and to analyse the final result comparing it to the initial NFA. It offered a learning experience, blending theoretical knowledge with practical application