MINISTERUL EDUCAȚIEI, CULTURII ȘI CERCETĂRII

AL REPUBLICII MOLDOVA

Universitatea Tehnică a Moldovei

Facultatea Calculatoare, Informatică şi Microelectronică

Departamentul Inginerie Software și Automatică

Grebennicova Ecaterina FAF-223

Report

Intro to formal languages. Regular grammar. Finite Automata.

of Formal Languages & Finite Automata

Checked by:

Cretu Dumitru, *university assistant*

Chișinău – 2023

1. **Theory**

A formal language is a structured system of communication used to represent ideas or instructions in a precise, unambiguous manner.

It consists of:

- Alphabet, which is the set of symbols or characters the language uses
- Vocabulary, which comprises the set of valid words or strings formed from the alphabet
- Grammar, which is the set of rules that dictates how symbols and words can be combined to produce valid sentences or expressions within the language.

The grammar ensures that the language can convey information effectively and without misunderstanding. For a language to be considered "formal," it must have a well-defined alphabet, vocabulary, and grammar, allowing it to systematically represent complex ideas and processes. Grammar, in this context, serves as the backbone of the language, establishing the syntax and structure necessary for coherent and meaningful communication. This precise structuring makes formal languages essential in fields like mathematics, computer science, and logic, where clarity and precision are paramount.

2. **Objectives**

- Discover what a language is and what it needs to have in order to be considered a formal one
- 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
- Implement a type/class for your grammar
- Add one function that would generate 5 valid strings from the language expressed by your given grammar
- Implement some functionality that would convert and object of type Grammar to one of type Finite Automaton
- For the Finite Automaton, please add a method that checks if an input string can be obtained via the state transition from it

3. **Implementation Description**

Point a, b - implemented

In this code was implenented a simple grammar-based string generator using recursive methods. It starts with a basic symbol and uses a set of rules to replace symbols with other symbols or characters until it ends up with a complete string. For this it uses a map to associate non-terminal symbols with their production rules and generates strings recursively expanding these symbols according to randomly selected production rules. It does this 5 times to show different possible outcomes.

The code:

```
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Random;

class Grammar {
    private final Map<Character, List<String>> rules;
    private final Random random;

    public Grammar() {
        this.rules = new HashMap<>();
        this.random = new Random();
        initializeRules();
    }

    private void initializeRules() {
        // Each non-terminal symbol ('S', 'B', 'D') is associated with its production
        // rules.
        rules.put('S', List.of("aB"));
        rules.put('B', List.of("aD", "bB", "cS"));
        rules.put('D', List.of("aD", "bS", "c"));
    }

    public String generateString() {
        // Start symbol - S
        return expand('S');
    }

    private String expand(char symbol) {
        StringBuilder result = new StringBuilder();
        // Check if the symbol is a non-terminal symbol
        if (rules.containsKey(symbol)) {
            // Randomly select one of the production rules for the symbol
            List<String> possibleProductions = rules.get(symbol);
            String production =
```

```
possibleProductions.get(random.nextInt(possibleProductions.size()));
            // Recursively expand each symbol in the selected production
            for (int i = 0; i < production.length(); i++) {
                char nextSymbol = production.charAt(i);
                result.append(expand(nextSymbol));
            }
        } else {
            // If it's a terminal symbol, simply append it to the result
            result.append(symbol);
        }
        return result.toString();
    }

    public static void main(String[] args) {
        Grammar grammar = new Grammar();
        // Generate 5 valid strings
        for (int i = 0; i < 5; i++) {
            System.out.println(grammar.generateString());
        }
    }
}
```

Point d - implemented

In this program was generated a finite automaton, that processes strings of characters to determine if they follow a specific pattern defined by the automaton's rules. It uses a set of steps (states) and changes these steps based on the letters it sees, then tells you if the string is okay or not based on where it ends up.

```
import java.util.*;

public class FiniteAutomaton {
    private Set<String> states;
    private Set<Character> alphabet;
    private Map<Map.Entry<String, Character>, String> transitions;
    private String current_state;
    private Set<String> accepting_states;

    public FiniteAutomaton() {
        states = new HashSet<>();
        states.add("S");
        states.add("B");
        states.add("D");

        alphabet = new HashSet<>();
        alphabet.add('a');
```

```
        alphabet.add('b');
        alphabet.add('c');

        transitions = new HashMap<>();
        // Define transitions: state + input symbol -> new state
        transitions.put(new AbstractMap.SimpleEntry<>("S", 'a'), "B");
        transitions.put(new AbstractMap.SimpleEntry<>("B", 'a'), "D");
        transitions.put(new AbstractMap.SimpleEntry<>("B", 'b'), "B");
        transitions.put(new AbstractMap.SimpleEntry<>("B", 'c'), "S");
        transitions.put(new AbstractMap.SimpleEntry<>("D", 'a'), "D");
        transitions.put(new AbstractMap.SimpleEntry<>("D", 'b'), "S");
        transitions.put(new AbstractMap.SimpleEntry<>("D", 'c'), "c");

        // Start at S
        current_state = "S";

        accepting_states = new HashSet<>();
        // Define accepting state(s)
        accepting_states.add("c");
    }

    // Attempt a transition based on the current state and input symbol. Returns
    // true if successful.
    public boolean transition(char symbol) {
        Map.Entry<String, Character> key = new AbstractMap.SimpleEntry<>(current_state,
symbol);
        if (transitions.containsKey(key)) {
            // Update current state based on transition
            current_state = transitions.get(key);
            return true;
        } else {
            // Transition not possible
            return false;
        }
    }

    // Checks if the given input string is accepted by the automaton.
    public boolean isStringAccepted(String inputString) {
        for (int i = 0; i < inputString.length(); i++) {
            char symbol = inputString.charAt(i);
            // If symbol not in alphabet or transition fails, string is not accepted.
            if (!alphabet.contains(symbol) || !transition(symbol)) {
                return false;
            }
        }
        // Check if ending state is an accepting state.
        return accepting_states.contains(current_state);
    }
```

```
    public static void main(String[] args) {
        FiniteAutomaton fa = new FiniteAutomaton();
        try (Scanner scanner = new Scanner(System.in)) {
            System.out.println("Enter a string to check:");
            String inputString = scanner.nextLine();

            boolean isValid = fa.isStringAccepted(inputString);
            if (isValid) {
                System.out.println("The string is accepted by the automaton.");
            } else {
                System.out.println("The string is not accepted by the automaton.");
            }
        }
    }
}
```

4. Conclusion

As a student, I got the chance to work again with Java, learn a new topic and work on its implementation. Was interesting to write the code, to understand how to work with grammar to generate valid strings and with finite automaton, which is a delicate topic based on my opinion. Got little difficulties at the beginning as I was not fully well informed with the themes, as well I got a little help to better understand all details of implementation of the programs. The first program represents grammar-based string generation, where I've successfully implemented a system to create random strings based on predefined rules, using recursive programming and rule-based generation. The second program was about finite automaton, which processed input strings and determined their acceptance based on a set of transition rules.