Laboratory work nr.2
Regular Grammars

Course: Formal Languages & Finite Automata
Student: Schipschi Daniil

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;

Implementation:

To carry out this laboratory assignment, the initial step involved the creation of the Grammar class using Python. This class serves as a blueprint for various Grammars, featuring attributes with clear and representative names. The important method within this class is `generateString()`.

```java
import java.util.*;

public class Grammar {
    // Set of non-terminal symbols
    private Set<Character> VN;
    // Set of terminal symbols
    private Set<Character> VT;
    // Map representing the productions
    private Map<Character, List<String>> P;
    // Start symbol
    private char S;
    // Random object for generating random strings
    private Random random;

    // Constructor to initialize the Grammar object
    public Grammar(Set<Character> VN, Set<Character> VT, Map<Character, List<String>> P, char S) {
        this.VN = VN;
        this.VT = VT;
        this.P = P;
        this.S = S;
        // Initialize Random object with current system time as seed
        this.random = new Random(System.currentTimeMillis());
    }

    // Method to generate a random string from the grammar
    public String generateString() {
        // StringBuilder to construct the generated string
        StringBuilder sb = new StringBuilder();
        // Call the helper method to generate the string starting from the start symbol
        generateStringHelper(S, sb);
        // Return the generated string
        return sb.toString();
    }

    // Helper method to recursively generate the string
    private void generateStringHelper(char symbol, StringBuilder sb) {
        // If the symbol is terminal, append it to the string builder
        if (VT.contains(symbol)) {
            sb.append(symbol);
        } else {
            // If the symbol is non-terminal, randomly choose a production and expand it
            List<String> productions = P.get(symbol);
            String chosenProduction = productions.get(random.nextInt(productions.size()));
            for (char c : chosenProduction.toCharArray()) {
                // Recursively call generateStringHelper for each symbol in the chosen production
                generateStringHelper(c, sb);
            }
        }
    }

    // Method to convert the grammar to a finite automaton
    public FiniteAutomaton toFiniteAutomaton() {
        // Set of states in the finite automaton
        Set<Character> Q = new HashSet<>(VN);
        Q.addAll(VT);
        // Alphabet of the finite automaton
        Set<Character> Sigma = new HashSet<>(VT);
        // Initial state of the finite automaton
        char q0 = S;
        // Set of final states in the finite automaton
        Set<Character> F = new HashSet<>();
        // Determine final states based on productions with ε
        for (char vn : VN) {
            if (P.get(vn).contains("ε")) {
                F.add(vn);
            }
        }
        // Transition function of the finite automaton
        Map<Character, Map<Character, Character>> delta = new HashMap<>();
        for (char q : Q) {
            delta.put(q, new HashMap<>());
            for (char c : Sigma) {
                // Initialize transition function with empty transitions
                delta.get(q).put(c, ' ');
            }
        }
        // Populate transition function based on grammar productions
        for (char vn : VN) {
            if (!P.containsKey(vn)) {
                P.put(vn, Collections.emptyList());
            }
            for (String production : P.get(vn)) {
                char nextState = production.charAt(0);
                char inputSymbol = production.length() > 1 ? production.charAt(1) : ' ';
                delta.get(vn).put(inputSymbol, nextState);
            }
        }
        // Create and return the finite automaton
        return new FiniteAutomaton(Q, Sigma, delta, q0, F);
    }
}
```

```java
import java.util.Map;
import java.util.Set;

class FiniteAutomaton {
    // Set of states
    private Set<Character> Q;
    // Alphabet (set of input symbols)
    private Set<Character> Sigma;
    // Transition function: maps a state and an input symbol to the next state
    private Map<Character, Map<Character, Character>> delta;
    // Initial state
    private char q0;
    // Set of accepting states
    private Set<Character> F;

    // Constructor to initialize the finite automaton
    public FiniteAutomaton(Set<Character> Q, Set<Character> Sigma, Map<Character, Map<Character, Character>> delta,
                           char q0, Set<Character> F) {
        this.Q = Q;
        this.Sigma = Sigma;
        this.delta = delta;
        this.q0 = q0;
        this.F = F;
    }

    // Method to check if a given string belongs to the language accepted by the automaton
    public boolean stringBelongToLanguage(final String inputString) {
        // Start from the initial state
        char currentState = q0;
        // Iterate through each character in the input string
        for (char c : inputString.toCharArray()) {
            // If the input symbol is not in the alphabet, the string cannot be accepted
            if (!Sigma.contains(c)) {
                return false;
            }
            // If there's no transition defined for the current state and input symbol, the string cannot be accepted
            if (!delta.get(currentState).containsKey(c)) {
                return false;
            }
            // Move to the next state based on the transition function
            currentState = delta.get(currentState).get(c);
        }
        // Check if the final state is one of the accepting states
        return F.contains(currentState);
    }

    // Method to generate a string representation of the finite automaton
    @Override
    public String toString() {
        StringBuilder sb = new StringBuilder();
        sb.append("States (Q): ").append(Q).append("\n");
        sb.append("Alphabet (Sigma): ").append(Sigma).append("\n");
        sb.append("Transition Function (delta): ").append(delta).append("\n");
        sb.append("Initial State (q0): ").append(q0).append("\n");
        sb.append("Accepting States (F): ").append(F);
        return sb.toString();
    }
}
```

```java
import java.util.*;

public class Main {
    public static void main(String[] args) {
        // Define the set of non-terminal symbols (VN), terminal symbols (VT), productions (P), and start symbol (S)
        Set<Character> VN = new HashSet<>(Arrays.asList('S', 'A', 'B'));
        Set<Character> VT = new HashSet<>(Arrays.asList('a', 'b', 'c', 'd'));
        Map<Character, List<String>> P = new HashMap<>();
        P.put('S', Arrays.asList("bS", "dA"));
        P.put('A', Arrays.asList("aA", "dB", "b"));
        P.put('B', Arrays.asList("cB", "a"));
        char S = 'S';

        // Create a Grammar object with the defined VN, VT, P, and S
        Grammar grammar = new Grammar(VN, VT, P, S);

        // Convert the grammar to a finite automaton
        FiniteAutomaton finiteAutomaton = grammar.toFiniteAutomaton();

        // Generate and print 5 random strings using the grammar
        System.out.println("Generated Strings: ");
        for (int i = 1; i <= 5; i++) {
            String generated = grammar.generateString();
            System.out.println(i + " " + generated);
            // Uncomment the following line if you want to check if the generated string is accepted by the automaton
            // System.out.println("Accepted by automaton? " + finiteAutomaton.stringBelongToLanguage(generated));
        }
    }
}
```

Conclusions:

This lab provided a valuable hands-on opportunity to grasp the foundational concepts of formal languages and their properties through practical examples. It enabled us to apply these concepts effectively to generate strings following predefined rules. Furthermore, it imparted the knowledge of transforming a grammar into a finite automaton and utilizing it to validate the correctness of given words. This experience serves as an excellent foundation for upcoming lab assignments, allowing us to approach them with confidence and understanding.