Ministerul Educaţiei și Cercetării al Republicii Moldova Universitatea Tehnică a Moldovei

Facultatea Calculatoare, Informatică și Microelectronică

Laboratory work 1:

Intro to formal languages. Regular grammars. Finite Automata.

Elaborated:

st. gr. FAF-223 Bujilov Dmitrii

Verified:

asist. univ. Dumitru Cretu

Chişinău - 2024

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 description

Firstly, I created Grammar class with all the necessary attributes and implemented a method to generate strings in it randomly (according to the grammar). You can also convert Grammar to FiniteAutomaton easily, using a method from the Grammar class:

```
public class Grammar {


    private final String startingSymbol;

    private final Set<String> nonTerminalSymbols;

    private final Set<String> terminalSymbols;

    private final Map<String, List<String>> productions;


    public Grammar(String startingSymbol, Set<String> nonTerminals, Set<String>
terminals, Map<String, List<String>> productions) {

        this.startingSymbol = startingSymbol;
```

```
        this.nonTerminalSymbols = nonTerminals;
        this.terminalSymbols = terminals;
        this.productions = productions;
    }

    public String generateString() {
        String result = getRandomProduction(startingSymbol);

        boolean containsNonTerminal = true;

        while(containsNonTerminal) {
            containsNonTerminal = false;
            for(char entry : result.toCharArray()) {
                String entryString = String.valueOf(entry);
                if(nonTerminalSymbols.contains(entryString)) {
                    result = result.replaceFirst(entryString,
getRandomProduction(entryString));
                    containsNonTerminal = true;
                }
            }
        }

        return result;
    }

    private String getRandomProduction(String nonTerminal) {
        return
productions.get(nonTerminal).get(ThreadLocalRandom.current().nextInt(productions
.get(nonTerminal).size()));
    }

    public FiniteAutomaton toFiniteAutomaton() {
```

```
        return new FiniteAutomaton(nonTerminalSymbols, terminalSymbols,
buildTransitions(productions), startingSymbol);

    }


    private Map<String, Map<String, String>> buildTransitions(Map<String,
List<String>> productions) {

        Map<String, Map<String, String>> transitions = new HashMap<>();


        for (Map.Entry<String, List<String>> entry : productions.entrySet()) {

            String state = entry.getKey();

            List<String> productionList = entry.getValue();


            Map<String, String> stateTransitions = new HashMap<>();


            for (String production : productionList) {

                String symbol = production.substring(0, 1); // Get the first
character as the symbol


                String nextState = production.length() > 1 ?
production.substring(1) : "accept";

                stateTransitions.put(symbol, nextState);

            }


            transitions.put(state, stateTransitions);

        }


        return transitions;

    }


}
```

Then I created FiniteAutomaton class with the necessary attributes and a method that would check whether a string belongs to the language or not.

```
public class FiniteAutomaton {


    private final Set<String> states;

    private final Set<String> alphabet;

    private final Map<String, Map<String, String>> transitions;

    private final String startState;


    public FiniteAutomaton(Set<String> states, Set<String> alphabet, Map<String,
Map<String, String>> transitions, String startState) {

        this.states = states;

        this.alphabet = alphabet;

        this.transitions = transitions;

        this.startState = startState;

    }


    public boolean stringBelongsToLanguage(String input) {

        String currentState = startState;


        for (char symbol : input.toCharArray()) {

            String symbolStr = String.valueOf(symbol);


            if (!alphabet.contains(symbolStr)) {

//                System.out.println("Invalid symbol: " + symbolStr);

                return false;

            }


            if (transitions.containsKey(currentState) &&
transitions.get(currentState).containsKey(symbolStr)) {

                currentState = transitions.get(currentState).get(symbolStr);

            } else {
```

```
//                System.out.println("No transition for state " + currentState +
" and symbol " + symbolStr);

                return false;

            }

        }


        return currentState.equals("accept");

    }


}
```

All the functionalities can be checked by providing all the needed data, like I did in the Main class.

```
public class Main {

   public static void main(String[] args) {

      String startingSymbol = "S";

      Set<String> nonTerminalSymbols = Set.of("S", "P", "Q");

      Set<String> terminalSymbols = Set.of("a", "b", "c", "d", "e", "f");

      Map<String, List<String>> productions = Map.of(

            "S", List.of("aP", "bQ"),

            "P", List.of("bP", "cP", "dQ", "e"),

            "Q", List.of("eQ", "fQ", "a")

      );


      Grammar grammar = new Grammar(startingSymbol, nonTerminalSymbols,
terminalSymbols, productions);

      FiniteAutomaton finiteAutomaton = grammar.toFiniteAutomaton();

      Scanner scanner = new Scanner(System.in);


      System.out.println("Generating 5 strings using the provided grammar:");

      for(int i = 0; i < 5; i++) System.out.println((i + 1) + ". " + grammar.generateString());
```

```
        System.out.print("Enter a string to check if it belongs to the language: ");
        String input = scanner.nextLine();
        System.out.println("The string \"" + input + "\"" +
(finiteAutomaton.stringBelongsToLanguage(input) ? " belongs " : " doesn't belong ") + "to the
language.");
    }
}
```

Conclusions and Results

In the course of my work in the laboratory, I successfully created a Grammar class to represent formal grammar, engaging in a range of activities related to grammar analysis and finite automaton conversion. As an overview I'd say that I did:

• Grammar Implementation:

I formulated a Grammar class that encapsulates the fundamental components of formal grammar, encompassing non-terminals, terminals, a start symbol, and production rules. This class facilitates efficient manipulation of these elements.

• String Generation:

I devised a function capable of generating five valid strings from the language specified by the given grammar. The string generation process adapts dynamically to the defined grammar rules.

• Finite Automaton Conversion:

I engineered functionality to convert a Grammar object into a Finite Automaton object. This entailed creating a FiniteAutomaton class, complete with states, an alphabet, transitions, and a method for validating whether an input string is accepted.

• String Acceptance by Finite Automaton:

I implemented a method within the Finite Automaton class to determine if an input string can be derived through state transitions. This implementation carefully adhered to the transition rules outlined by the grammar, ensuring accurate traversal of the automaton.