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 Cravcenco 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:

```
public class Grammar {

    private Set<Character> nonTerminals;
    private Set<Character> terminals;
    private Character startSymbol;
    private Map<String, List<String>> rules;
}
```

Then I implemented a method to generate strings in the same Grammar class. Basically it takes a list of current non-terminals, selects randomly one, then randomly chooses a rule for the chosen non-terminal and replaces chosen non-terminal according to the rule.

```
public String generateString() {

        List<Character> currentNonTerminals = new
java.util.ArrayList<>(List.of(startSymbol));
        StringBuilder result = new StringBuilder();
        result.append(startSymbol);
        Random rand = new Random();
        while (!currentNonTerminals.isEmpty()) {

            int randIndex = rand.nextInt(currentNonTerminals.size());

            int placeToReplace =
result.indexOf(currentNonTerminals.get(randIndex).toString());

            int randIndexToReplace =
rand.nextInt(rules.get(currentNonTerminals.get(randIndex).toString()).size());
            String valueToReplace =
rules.get(currentNonTerminals.get(randIndex).toString()).get(randIndexToReplace)
;

            result.replace(placeToReplace, placeToReplace + 1, valueToReplace);

            currentNonTerminals.remove(randIndex);
            for (char c: valueToReplace.toCharArray()) {
                if (nonTerminals.contains(c)) {
                    currentNonTerminals.add(c);
                }
            }
        }
        return result.toString();
    }
```

Then I created FiniteAutomaton class with the necessary attributes and also Transition class.

```
public class FiniteAutomaton {

    private Set<Character> statesQ; // non-terminals + null
    private Set<Character> alphabetSigma; // terminals
    private Set<Transition> transitions; // A -> aB, (A,a)=B, A->a, (A,a)=empty
string
    private char startStateQ0; // start symbol
    private final Character finalStateF = null; // empty string
}
public class Transition {
        private Character fromState;
        private Character withSymbol;
        private Character toState;

        public Transition(Character fromState, Character withSymbol, Character
toState) {
            this.fromState = fromState;
            this.withSymbol = withSymbol;
            this.toState = toState;
        }
}
```

I add conversion Grammar->FiniteAutomaton, that is implemented according to the following rules:

- Set of states (Q) = Vn + {X}, X - final additional state
- Alphabet (sigma) = Vt
- Starting symbol (q0) = {S}
- Final symbol (F) = {X}
- Transitions: A -> aB, then: from state: A, withSymbol: a, toState: B
- Transition: A -> a, then: from state: A, withSymbol: a, toState: X

```
public FiniteAutomaton(Grammar grammar) {

        statesQ = grammar.getNonTerminals();
        statesQ.add(null);
        alphabetSigma = grammar.getTerminals();
        startStateQ0 = grammar.getStartSymbol();
        transitions = new HashSet<>();

        Set<String> rules = grammar.getRules().keySet();
        for (String rule : rules) {
            List<String> transitionTo = grammar.getRules().get(rule);

            for (String transition : transitionTo) {
                // A -> B
                if (transition.length() == 1 &&
grammar.getNonTerminals().contains(transition.charAt(0))) {
                    transitions.add(new Transition(rule.charAt(0), null,
transition.charAt(0)));
                }
                // A -> a
                if (transition.length() == 1 &&
grammar.getTerminals().contains(transition.charAt(0))) {
                    transitions.add(new Transition(rule.charAt(0),
transition.charAt(0), null));
                }

                // A -> aB
                if (transition.length() == 2 &&
grammar.getTerminals().contains(transition.charAt(0)) &&
grammar.getNonTerminals().contains(transition.charAt(1))) {
                    transitions.add(new Transition(rule.charAt(0),
transition.charAt(0), transition.charAt(1)));
                }
            }
        }
    }
```

To check if input string can be obtained via the state transition, according to current Grammar I wrote the following method. Its essence is that by using recursion I need to check if there is a transaction that can go from currentState using currentNonTerminal and in the end, by changing currentState and making the input string shorter by deleting the first symbol I can or can't reach an empty string, with currentState null, which represents that I reached final state.

```
public boolean checkString(String string, Character currentState) {

        if (string.isEmpty() && currentState == null) {
            return true;
        }

        boolean found = false;
        for (Transition transition : transitions) {
            if (transition.getFromState().equals(currentState) &&
transition.getWithSymbol() == string.charAt(0)) {
                if (transition.getToState() == null && string.length() > 1) {
                    continue;
                }
                currentState = transition.getToState();
                found = checkString(string.substring(1), currentState);
            }

            if (found) return true;
        }

        return false;
    }
```

Conclusions and Results

In this laboratory, I successfully implemented a Grammar class to represent formal grammar and performed various tasks related to grammar analysis and finite automaton conversion. Here's a summary of my key accomplishments:

- Grammar Implementation

I designed a Grammar class that encapsulates the essential elements of formal grammar, including non-terminals, terminals, a start symbol, and production rules. This class provides a convenient way to manipulate these components.

- String Generation

I implemented a function to generate 5 valid strings from the language expressed by the given grammar. The string generation logic is adaptable based on the rules defined in the grammar.

- Finite Automaton Conversion

I developed functionality to convert an object of type Grammar to an object of type Finite Automaton. This involved creating a FiniteAutomaton class with states, an alphabet, transitions, and a method to check if an input string is accepted.

- String Acceptance by Finite Automaton

I implemented a method in the Finite Automaton class to check if an input string can be obtained via state transitions. The implementation considered the transition rules defined by the grammar and traversed the automaton accordingly.