Ministerul Educaţiei și Cercetării al Republicii Moldova Universitatea Tehnică a Moldovei Facultatea Calculatoare, Informatică și Microelectronică

Laboratory work nr. 1

Course: Formal languages and finite automata

Topic: Intro to formal languages. Regular grammars. Finite automata

Elaborated: st. gr. FAF-221 Cusnir Grigorii

Verified: asist. univ.Cretu Dumitru

Chişinău - 2024

1 Introduction

The provided Java code includes classes for handling grammars and converting them into finite automata. This report aims to analyze the functionality of the Grammar class, the process of converting grammars to finite automata, and the effectiveness of the code in validating strings against the generated automata.

2 Code Explanation

2.1 Grammar Class

The Grammar class represents a context-free grammar (CFG). It consists of variables `S`, `V_n`, `V_t`, and `P`, representing the start symbol, non-terminal symbols, terminal symbols, and production rules, respectively. The `generateString()` method generates strings based on the grammar rules, while the `generateFromSymbol()` method recursively expands symbols into strings following the production rules.

```
public class Grammar {
```

```
    private final char S = 'S';
    private final List<Character> V_n = List.of('S', 'D', 'R');
    private final List<Character> V_t = List.of('a', 'b', 'c', 'd', 'f');
    private final Map<Character, List<String>> P = Map.of(
            'S', List.of("aS", "bD", "fR"),
            'D', List.of("cD", "dR", "d"),
            'R', List.of("bR", "f")
    );
}
```

2.2 FiniteAutomaton Class

The FiniteAutomaton class represents a finite automaton (FA). It comprises states (`Q`), input alphabet (`Sigma`), transition function (`Delta`), initial state (`q0`), and accepting states (`F`). The class provides functionality to convert a grammar into an FA and validate whether a given string belongs to the language accepted by the FA.

```
class FiniteAutomaton {
    private final List<Character> Q;
    private final List<Character> Sigma;
    private final Map<Pair, Character> Delta;
    private final char q0;
    private final List<Character> F;
}
```

2.3 Pair Class

The Pair class defines a pair of characters, used as keys in the transition function of the FiniteAutomaton class. It overrides the `equals()` and `hashCode()` methods to ensure correct behavior when used in collections.

```
class Pair {
    private final char first;
    private final char second;

    public Pair(char first, char second) {
        this.first = first;
        this.second = second;
    }
}
```

3 Functionality

3.1 Grammar Generation

The Grammar class generates strings by recursively expanding symbols according to the production rules until only terminal symbols remain. It uses a random selection of production rules to introduce variability in generated strings.

3.2 Finite Automaton Conversion

The FiniteAutomaton class provides a method `toFiniteAutomaton()` to convert a given grammar into a corresponding finite automaton. This conversion involves mapping non-terminals to states, terminals to the input alphabet, and production rules to transitions in the automaton.

3.3 Language Validation

The FiniteAutomaton class offers a method `stringBelongToLanguage()` to determine whether a given string belongs to the language accepted by the automaton. It simulates the automaton's transitions on the input string and checks if the final state is one of the accepting states.

4 Analysis

The provided code effectively demonstrates the generation of strings from a context-free grammar and the conversion of grammars into finite automata. However, it lacks error handling and assumes well-formed input grammars. Additionally, the randomness in string generation may lead to non-deterministic outputs. Improvements could include better encapsulation, error handling, and optimization of string generation algorithms.

5 Conclusion

The Grammar and FiniteAutomaton classes offer functionality for handling grammars and finite automata in Java. While the code showcases the conversion process and language validation, there is room for enhancement in terms of error handling and efficiency. Overall, the code serves as a useful foundation for understanding grammar manipulation and automata theory concepts in Java programming.