Regular Grammars & Finite Automata

Course: Formal Languages & Finite Automata

Author: Gavriliuc Tudor

Theory

The study of abstract computing devices, or abstract state machines, is known as automata theory. An automaton is a digital computer's abstract model. Every automaton therefore has a few basic characteristics. It is equipped with a mechanism to read input. The input will be taken to be a string across a specified alphabet. One symbol at a time, the input mechanism may scan the input string from left to right and determine when the string is finished. The automaton contains a control unit that can be in any one of a finite number of internal states and can change state in a predetermined way based on transition functions. It can also produce output in some form.

The finite automata (FA) is characterized by the finite number of states and it is known the following types of the FA:

- Deterministic finite automata (DFA).
- Nondeterministic finite automata (NFA).
- ε -Nondeterministic finite automata (ε –NFA).

Regular grammars, also known as Type-3 grammars in the Chomsky hierarchy, can be represented by finite automata. Regular grammars have productions of the form $A \rightarrow aB$ or $A \rightarrow a$, where A and B are non-terminal symbols, and a is a terminal symbol.

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:
    - Create GitHub repository to deal with storing and updating your project;
    - 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.);
    - Store reports separately in a way to make verification of your work simpler
3. According to your variant number, get the grammar definition and do the following:
    - 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;

Implementation description

1. Grammar Class:

The Grammar class represents a context-free grammar (CFG).

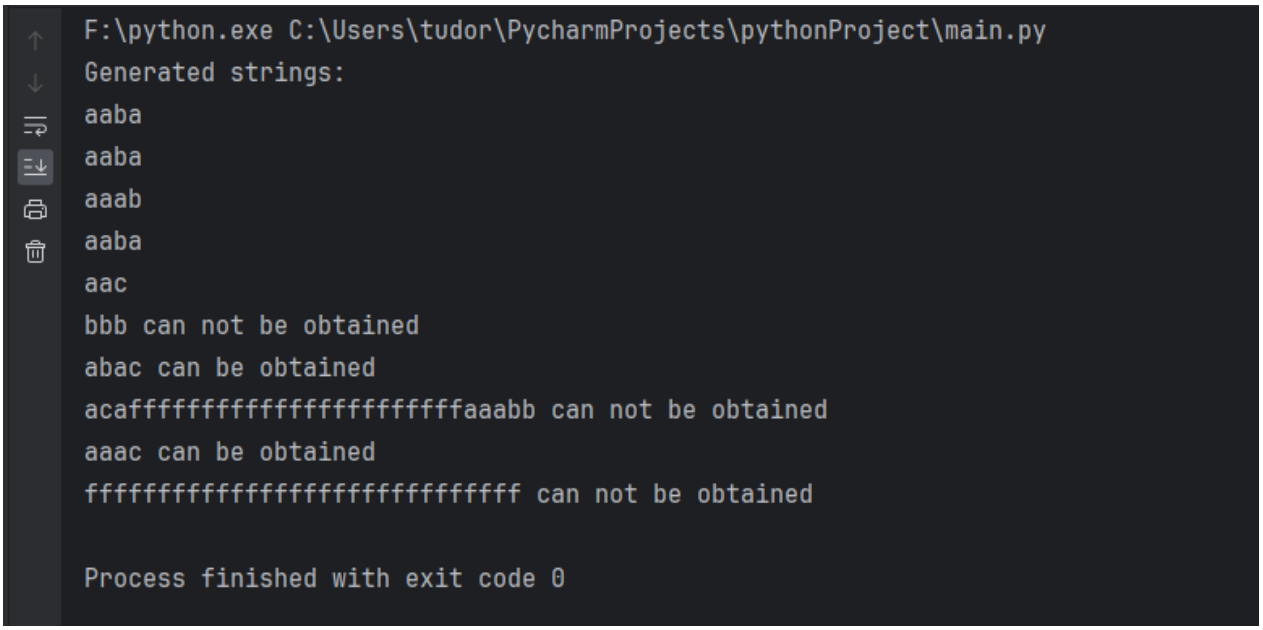
- Attributes:
  - VN: Set of non-terminal symbols.
  - VT: Set of terminal symbols.
  - P: Production rules, where each non-terminal symbol maps to a list of production rules.
  - S: Start symbol.
- Methods:
  - generate_valid_string(max_depth = default 5): Generates valid strings based on the grammar rules. It recursively generates strings by replacing non-terminals with their corresponding productions until terminal symbols are reached.
  - generate_5_valid_strings(): Calls the generate_valid_string method and stores the output in a list.
  - toFiniteAutomaton(): Converts the grammar to an equivalent finite automaton (FA). It constructs the states, alphabet, transition function, initial state, and set of final states of the FA based on the grammar rules.

2. FiniteAutomaton Class:

The FiniteAutomaton class represents a finite automaton (FA).

- Attributes:
  - Q: Set of states.
  - Sigma: Alphabet (set of symbols).
  - Delta: Transition function, which maps a state and a symbol to a set of states.
  - q0: Initial state.
  - F: Set of final states.
- Methods:
  - follow_rules (w): Determines whether a given string w belongs to the language accepted by the automaton. It simulates the state transitions based on the input string and returns True if the final state after processing the string is one of the final states, otherwise returns False.

Results



Conclusions

To sum up, this study has offered an investigation into the domain of finite automata and formal languages. We have studied the basic ideas of automata theory and gained a knowledge of automata, which are abstract computational devices that are used as models for digital computers. We have learned more about ε-NFA and deterministic and nondeterministic finite automata, each with unique properties and uses, thanks to this research.

In addition, we have built two classes: FiniteAutomaton to represent finite automata and Grammar to express context-free grammars. We have been able to determine language acceptability through state transitions and create acceptable strings from grammatical rules thanks to these implementations, which have helped us close the gap between theoretical notions and real-world applications.