**Grammar to Finite Automaton Conversion and String Generation**

**Course: Formal Languages & Finite Automata**



**Author: Doschinescu Dan**

**Group: FAF-222**

**Variant: 8**

**Chisinau 2024**

**Theory:**

A language, in a broad sense, is a system of communication that uses symbols (such as words, gestures, or symbols) and rules to convey meaning. Languages can be spoken, written, or signed, and they are essential for human communication and the expression of thoughts, ideas, and emotions.

Formal languages, on the other hand, have specific characteristics that distinguish them from natural languages (like English or Spanish). In the context of computer science, mathematics, and logic, a formal language is a well-defined set of symbols and rules for manipulating those symbols. Here are some key features that a language needs to have in order to be considered formal:

⦁ Alphabet:

⦁ A formal language has an alphabet, which is a finite set of symbols or characters that are used to construct expressions or statements in the language.

⦁ Syntax:

⦁ A formal language has a syntax that defines the rules for constructing valid expressions or statements. Syntax specifies the arrangement and combination of symbols to create meaningful structures.

⦁ Grammar:

⦁ The grammar of a formal language defines the set of rules governing the structure of expressions or statements. It describes how symbols can be combined to create valid constructs within the language.

⦁ Semantics:

⦁ Semantics deals with the meaning of expressions or statements in the formal language. It defines the relationship between the symbols and the real-world entities they represent or the operations they perform.

⦁ Precision:

⦁ Formal languages are characterized by precision and lack ambiguity. Every symbol and rule must have a clear and unambiguous interpretation.

⦁ Completeness:

⦁ The formal language should be complete, meaning that it should be able to express all possible statements or truths within its scope.

⦁ Rigor:

⦁ Rigor is a high level of accuracy and strict adherence to rules. Formal languages often require rigor in both syntax and semantics to ensure unambiguous interpretation.

⦁ Formalism:

⦁ Formal languages are often associated with formal systems, which are mathematical or logical structures that provide a foundation for reasoning about the language.

Examples of formal languages include programming languages (like Java, Python, or C++), mathematical notations (like set theory or propositional logic), and markup languages (like HTML or XML). These languages play a crucial role in various fields, providing a precise and structured way to express information and carry out computations.

**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)

According to your variant number, get the grammar definition and do the following:

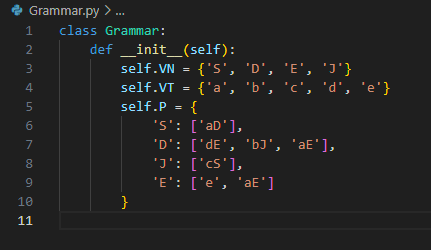
a. Implement a type/class for your grammar;

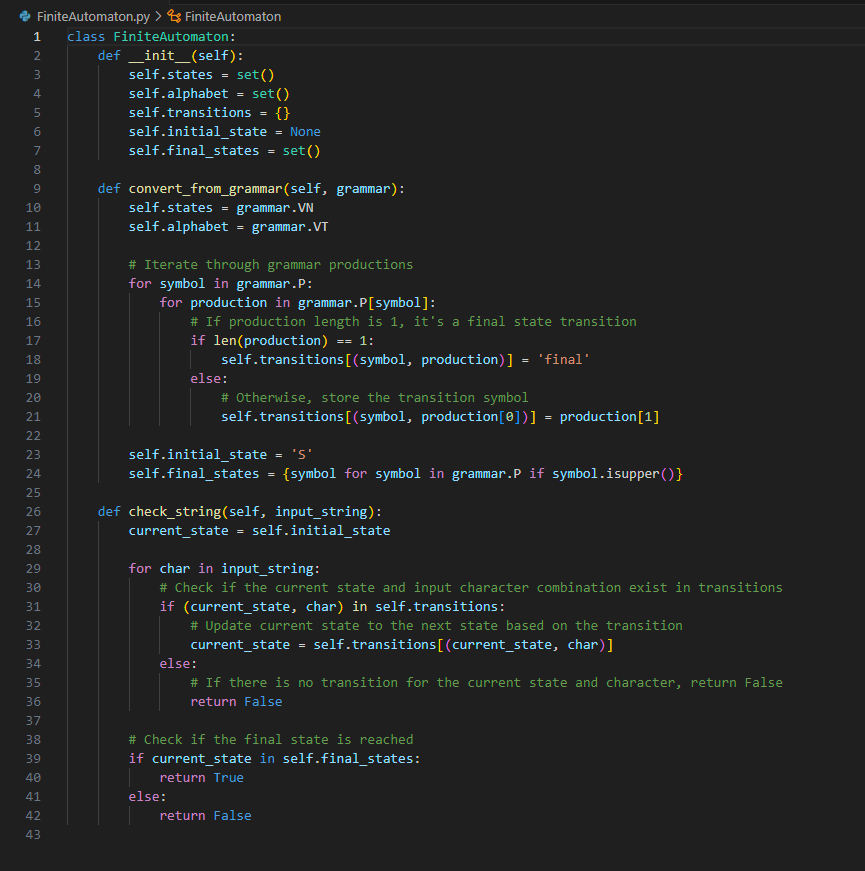
b. Add one function that would generate 5 valid strings from the language expressed by your given grammar;

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;

**My code:**

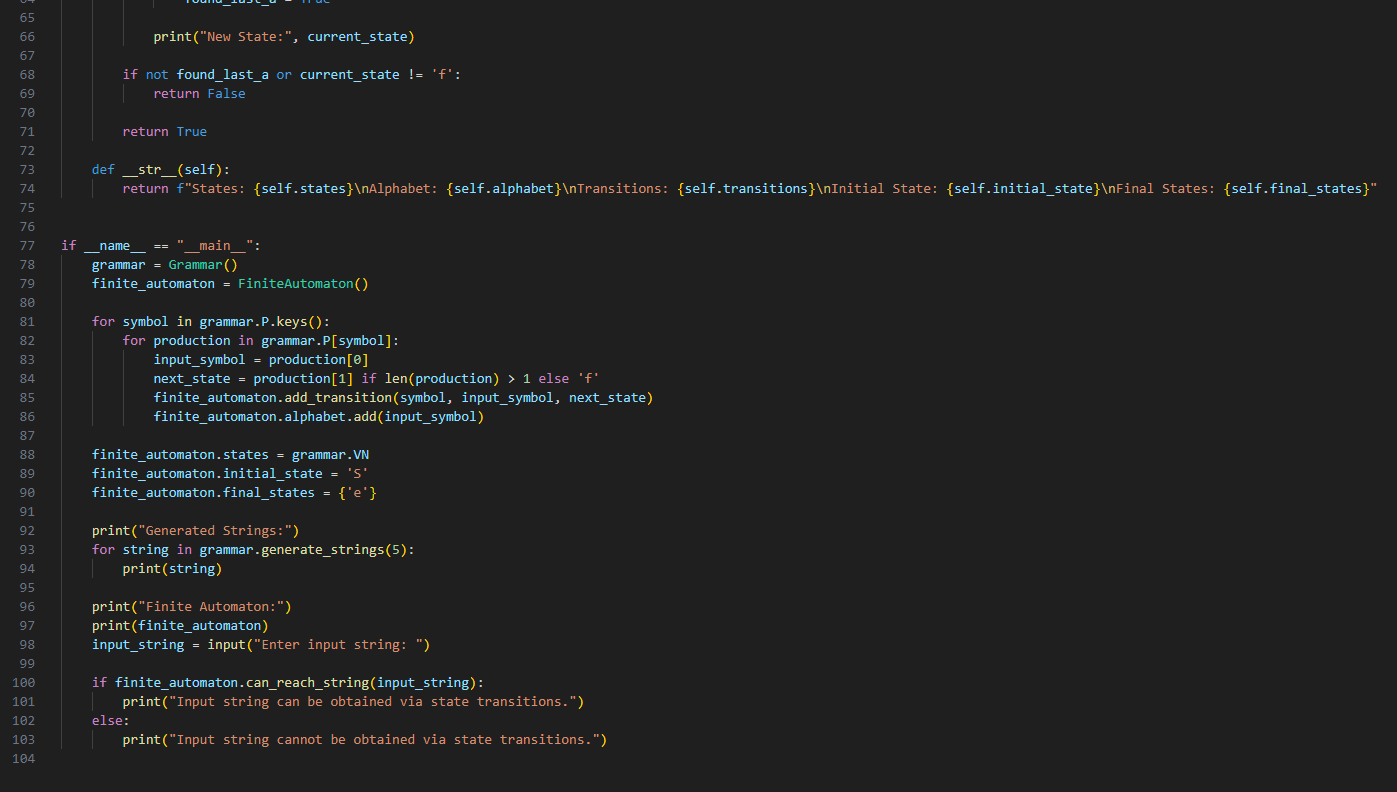




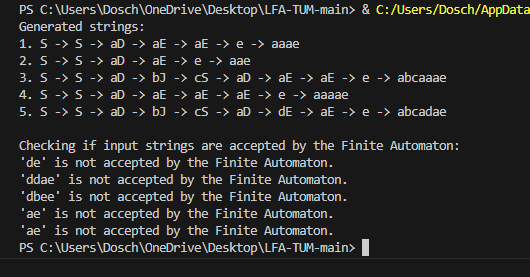


**Second variant of solving my code:**





**Results:**



**Second variant of emplimentation:**

