Regular Grammars. Finite Automata

**Course: Formal Languages & Finite Automata**

**Author: Popescu Nichifor FAF-212**

**Theory**

Def:

Language - a means of communicating information, by using visual or audio interpretations of words.

Formal language - a set of strings based on an alphabet that are generated with the help of a grammar.

String - a combination of symbols generated with the help of rules from the production set.

Grammar - an entity defined by four elements: the set of non-terminal symbols, the set of terminal symbols, the start symbol, and the set of production rules.

Automation - an abstract computational device. It contains the states, an alphabet, transition functions for each state, the initial and final states.

Finite automaton - an automaton with finite amounts of states and transitions.

**Objectives:**

Understand what a language is and what it needs to have in order to be considered a formal one.

Provide the initial setup for the evolving project that you will work on during this semester. I said project because usually at lab works, I encourage/impose students to treat all the labs like stages of development of a whole project. Basically you need to do the following:

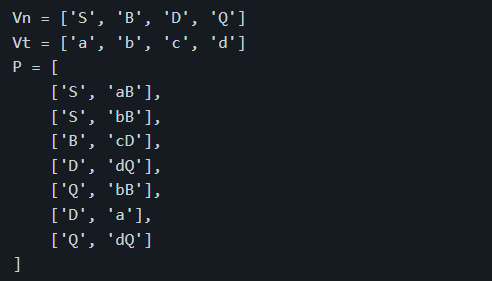
* + Create a local && remote repository of a VCS hosting service (let us all use Github to avoid unnecessary headaches);
  + Choose a programming language, and my suggestion would be to choose one that supports all the main paradigms;
  + Create a separate folder where you will be keeping the report. This semester I wish I won't see reports alongside source code files, fingers crossed;

According to your variant number (by universal convention it is register ID), get the grammar definition and do the following tasks:

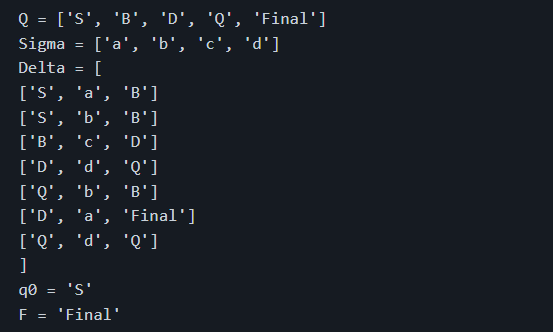
* + Implement a type/class for your grammar;
  + Add one function that would generate 5 valid strings from the language expressed by your given grammar;
  + Implement some functionality that would convert and object of type Grammar to one of type Finite Automaton;
  + For the Finite Automaton, please add a method that checks if an input string can be obtained via the state transition from it;

**Implementations Description:**

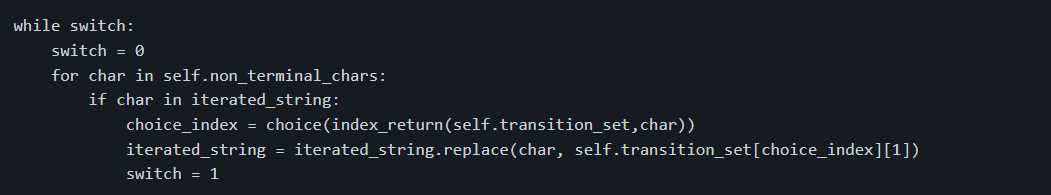
The grammar was implemented under this format. Vn and Vt represent the non-terminal and terminal set of characters respectively. The transition set P contains a list of pairs. A pair consists of two strings where the first string shows what combination of terminal and non-terminal characters should convert into the second string. The first non-terminal character in the set is used as the starting character, whereas by convention, the starting character is 'S'.



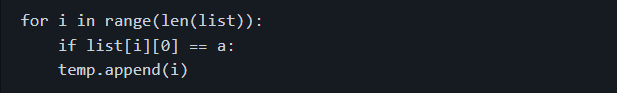
The FA was implemented under this format. Q is the set of all states, Sigma is the alphabet, q0 is the initial state and F is the set of final states possible. Delta is the transition set, it contains lists of 3 strings. The first string is the state the FA is in, the second string is the character the FA is checking, and the third string is the state the FA will transition to.



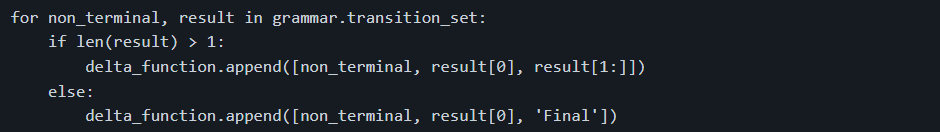
In the method below, using a while loop, I'm replacing the non-terminals with their respective strings until there are no more non-terminal characters. The replaced strings are chosen randomly given there are multiple ways for a non-terminal to transition.



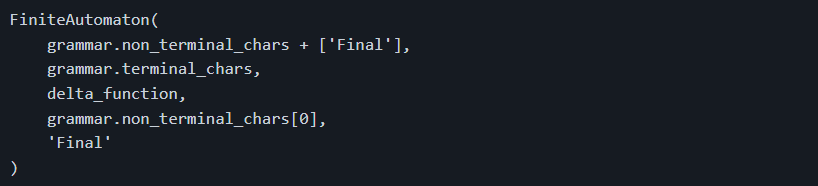
The index\_return function iterates through the transition set and adds all the indices where the passed non-terminal is present.

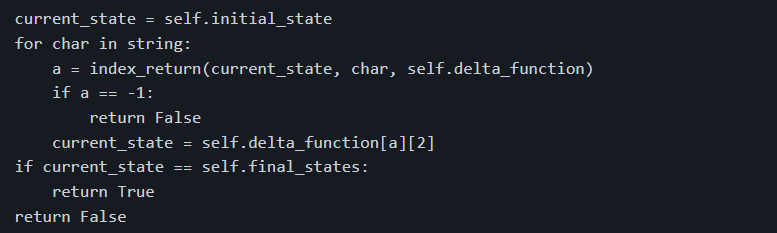


In order to convert the grammar to a finite automaton, I implemented a Convertor class with the respective method. By using a for loop, I iterated through the transition set of the grammar, extracting the necessary non-terminal and terminal characters for the finite automaton. However, there is a special case where the result string from the transition set is only a character long, that is when there are no non-terminal characters in the string, meaning we should use an unconventionally chosen 'Final' state for the FA.

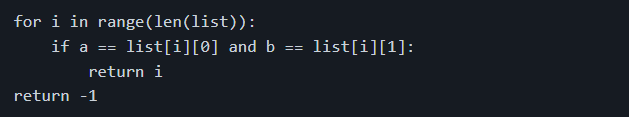


For the construction of the FA, we use the non-terminal characters and the 'Final' state as the set of all states in the FA. The alphabet of the grammar is used for the set of terminal characters. The initial state is chosen as the first non-terminal character of the Grammar.

The method below, by using a for loop, I iterated through each character of the passed string. I then identified the index of each state/character pair in the delta function, if the pair is not contained in the delta function, then the function returns -1, which by an if statement, will return False. The current state is changed to the respective state from the delta function. The process is repeated until the end of the string, where it will be checked whether the current state is a final state. Note that this method won't work in case the FA is non-deterministic.

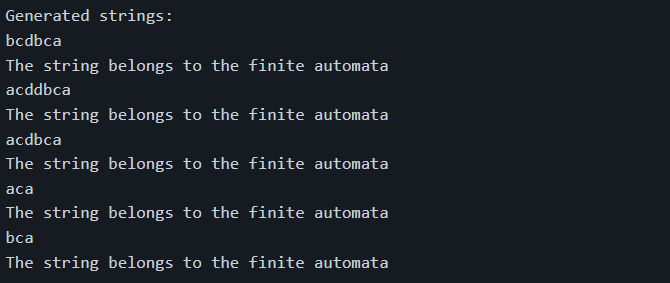


The index\_return function here is different from the previous one, though it functions in a similar way. It looks for the matching state/character pair in the passed delta function.

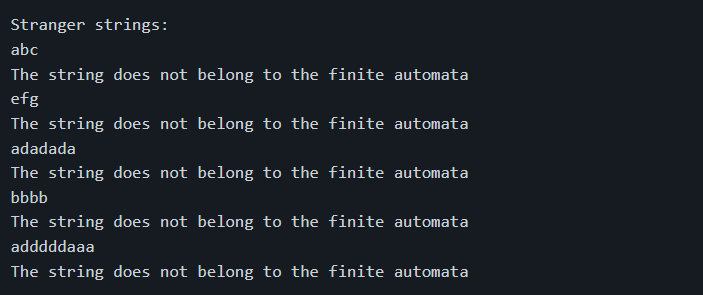


**Results**

The following strings are generated by my grammar given the variant. The FA immediately checks if they belong to the language.



If instead I input "stranger" strings into the finite automaton, it will return the following message.



**Conclusion**

In conclusion, we have implemented a Python program that generates valid strings for a given context-free grammar and converts the grammar to a finite automaton. The program includes several classes, methods, and unit tests to ensure correct functionality. Overall, this program provides a useful tool for generating valid strings for a given context-free grammar and checking if they are accepted by a finite automaton, which has a wide range of applications in various fields, such as natural language processing, programming languages, and computer science.