Laboratory work nr.2
Regular Grammars

Course: Formal Languages & Finite Automata
Student: Negai Marin

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;

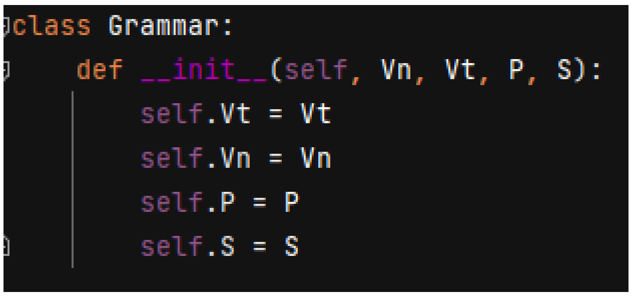
b. Add one function that would generate 5 valid strings from the language expressed by your given grammar;

c. Implement some functionality that would convert and object of type Grammar to one of type Finite Automaton;

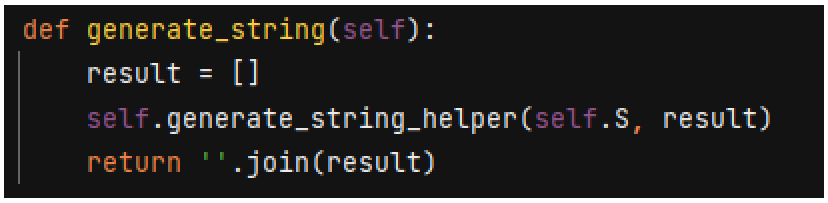
d. For the Finite Automaton, please add a method that checks if an input string can be obtained via the state transition from it;

Implementation:

To carry out this laboratory assignment, the initial step involved the creation of the Grammar class using Python. This class serves as a blueprint for various Grammars, featuring attributes with clear and representative names. The important method within this class is `generateString()`.

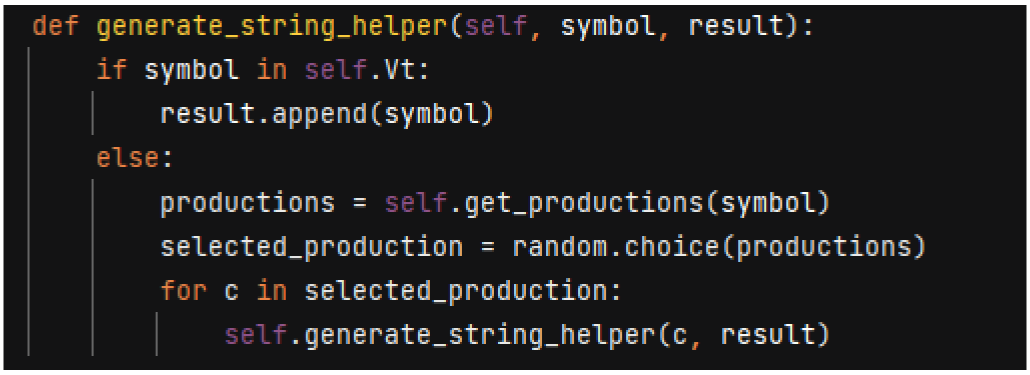


```python
class Grammar:
    def __init__(self, Vn, Vt, P, S):
        self.Vt = Vt
        self.Vn = Vn
        self.P = P
        self.S = S
```

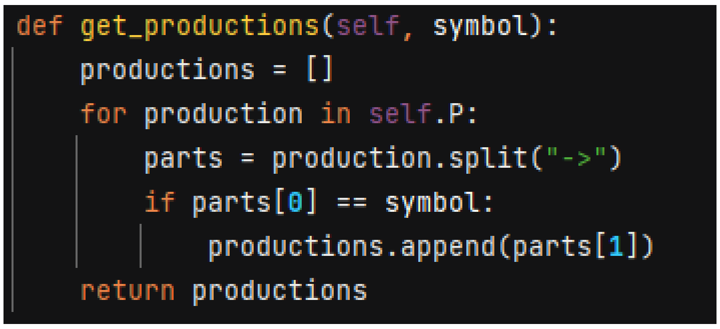


```python
def generate_string(self):
    result = []
    self.generate_string_helper(self.S, result)
    return ''.join(result)
```

This method uses another function named `generateStringHelper()`. If the symbol is a terminal, it is appended to the result; however, if it is a non-terminal symbol, the method identifies possible productions through `getProductions()`, randomly selects one, and proceeds recursively. The recursive process stops when all productions exclusively consist of terminals.

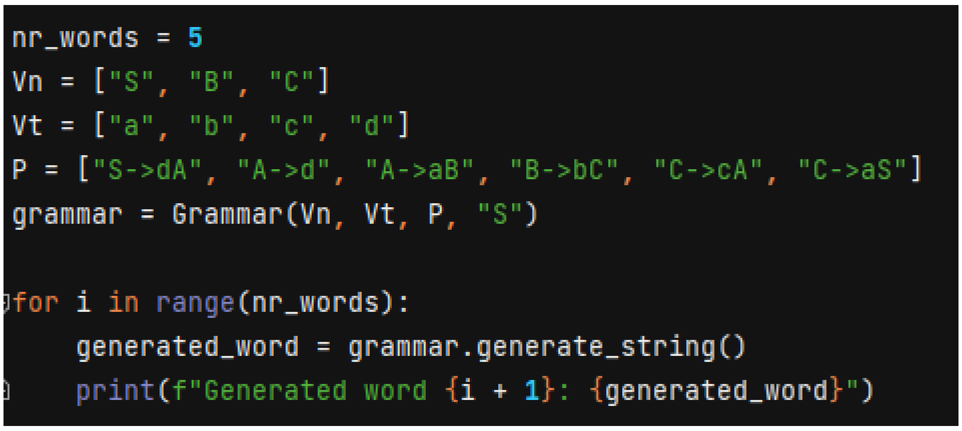


```python
def generate_string_helper(self, symbol, result):
    if symbol in self.Vt:
        result.append(symbol)
    else:
        productions = self.get_productions(symbol)
        selected_production = random.choice(productions)
        for c in selected_production:
            self.generate_string_helper(c, result)
```



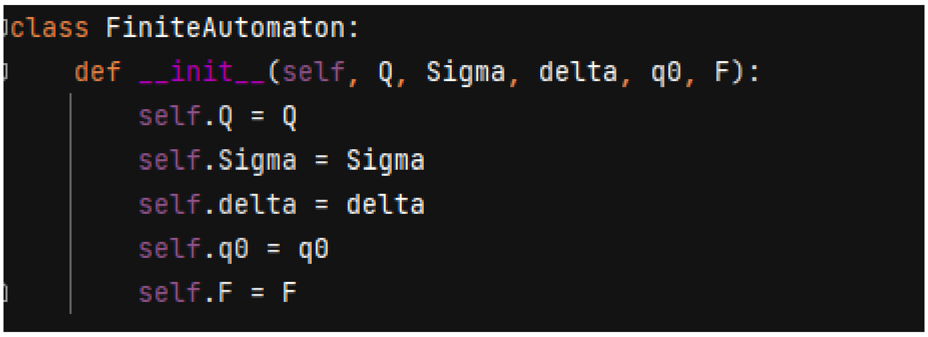
```python
def get_productions(self, symbol):
    productions = []
    for production in self.P:
        parts = production.split("->")
        if parts[0] == symbol:
            productions.append(parts[1])
    return productions
```

Subsequently, armed with the Grammar class, I could generate five strings in the main class using a for loop and a Grammar object.

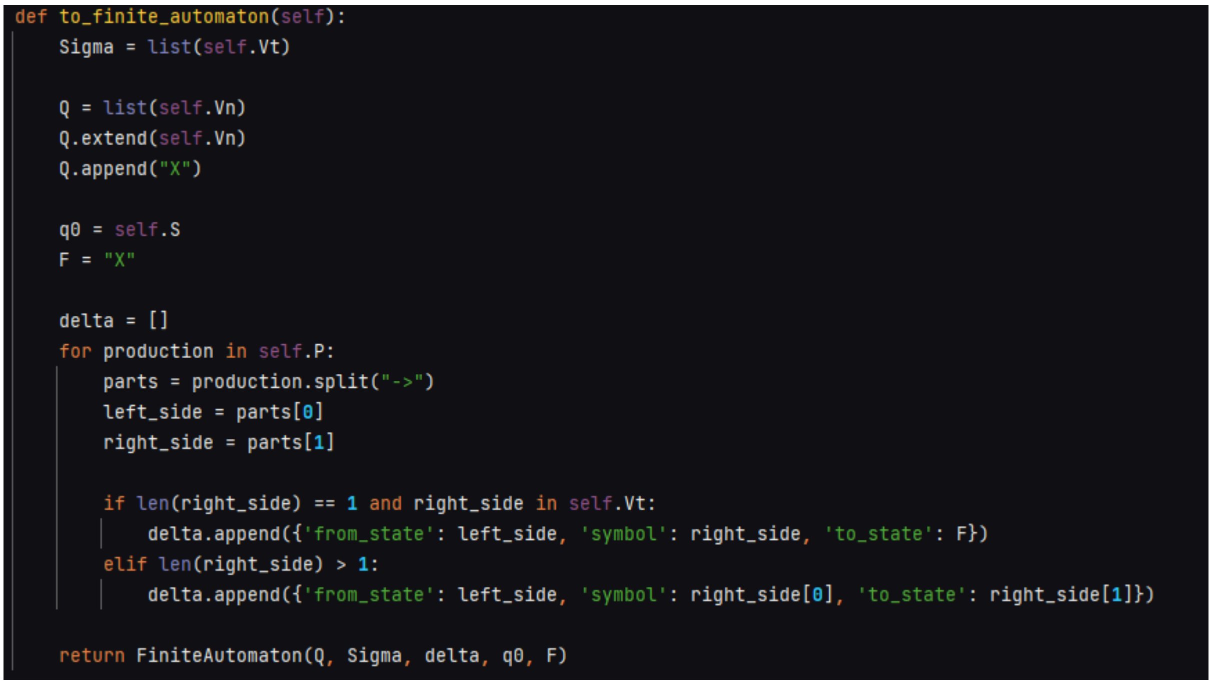


```python
nr_words = 5
Vn = ["S", "B", "C"]
Vt = ["a", "b", "c", "d"]
P = ["S->dA", "A->d", "A->aB", "B->bC", "C->cA", "C->aS"]
grammar = Grammar(Vn, Vt, P, "S")

for i in range(nr_words):
    generated_word = grammar.generate_string()
    print(f"Generated word {i + 1}: {generated_word}")
```

The next step was the creation of a FiniteAutomaton class, featuring specific attributes. A method was added to transform a Grammar object into a FiniteAutomaton. Delta is represented by a dictionary.

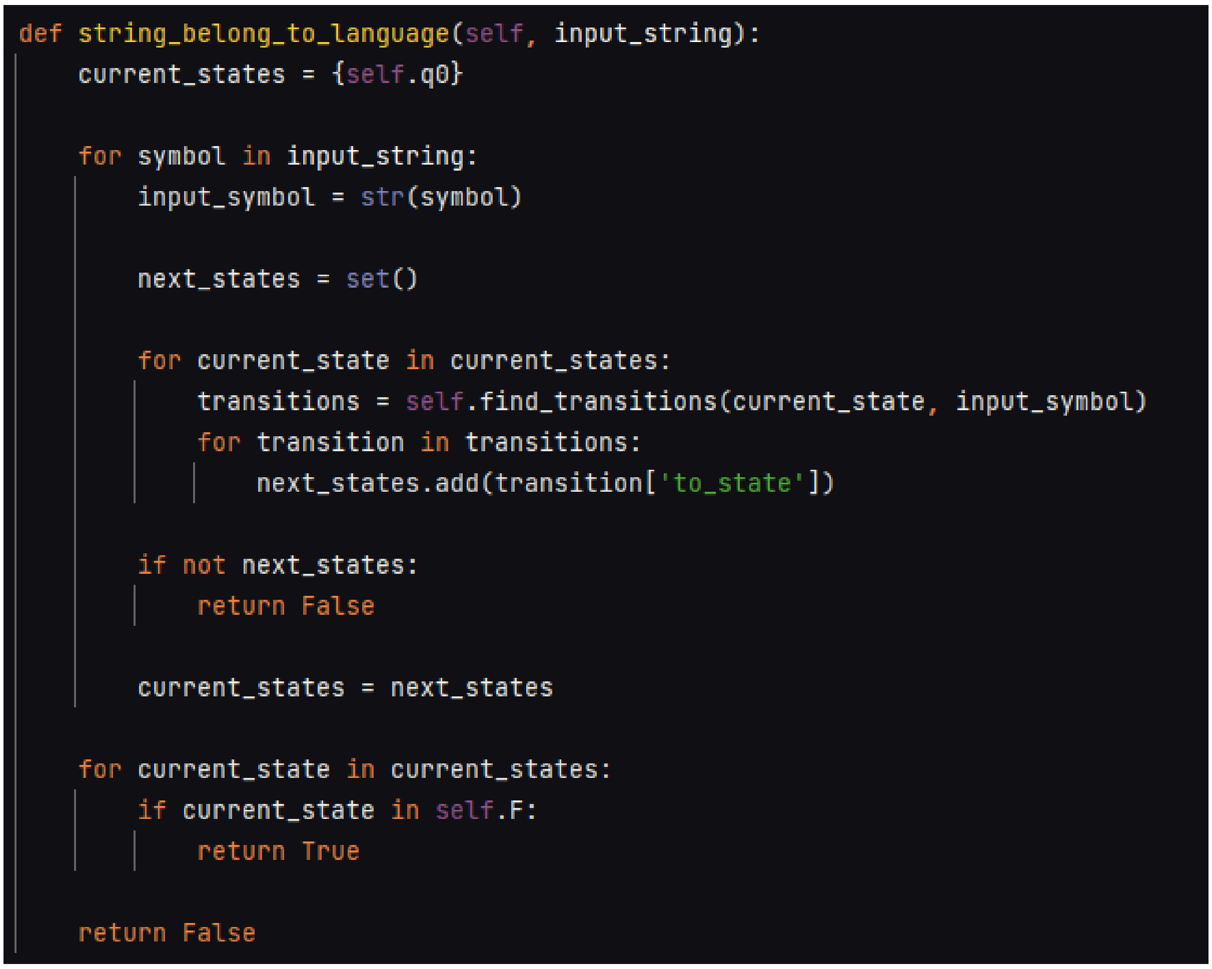


```python
class FiniteAutomaton:
    def __init__(self, Q, Sigma, delta, q0, F):
        self.Q = Q
        self.Sigma = Sigma
        self.delta = delta
        self.q0 = q0
        self.F = F
```

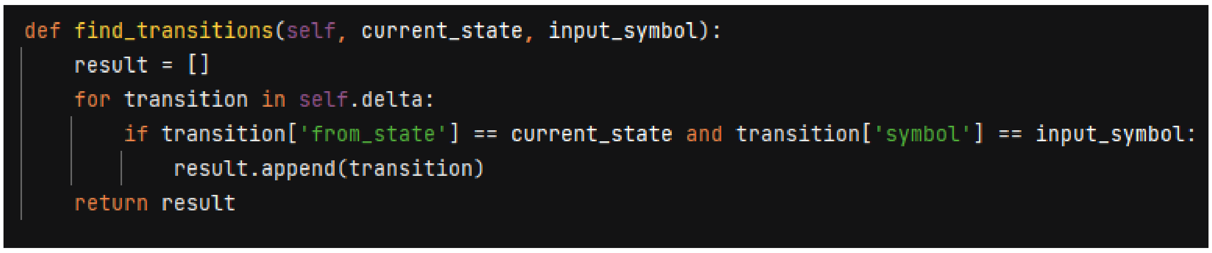


```python
def to_finite_automaton(self):
    Sigma = list(self.Vt)

    Q = list(self.Vn)
    Q.extend(self.Vn)
    Q.append("X")

    q0 = self.S
    F = "X"

    delta = []
    for production in self.P:
        parts = production.split("->")
        left_side = parts[0]
        right_side = parts[1]

        if len(right_side) == 1 and right_side in self.Vt:
            delta.append({'from_state': left_side, 'symbol': right_side, 'to_state': F})
        elif len(right_side) > 1:
            delta.append({'from_state': left_side, 'symbol': right_side[0], 'to_state': right_side[1]})

    return FiniteAutomaton(Q, Sigma, delta, q0, F)
```

The final step involved the creation of a method to check whether a string belongs to this FiniteAutomaton. The method iterates through each symbol, checking for appropriate transitions using an additional method named `findTransitions()`.



```python
def string_belong_to_language(self, input_string):
    current_states = {self.q0}

    for symbol in input_string:
        input_symbol = str(symbol)

        next_states = set()

        for current_state in current_states:
            transitions = self.find_transitions(current_state, input_symbol)
            for transition in transitions:
                next_states.add(transition['to_state'])

        if not next_states:
            return False

        current_states = next_states

    for current_state in current_states:
        if current_state in self.F:
            return True

    return False
```



```python
def find_transitions(self, current_state, input_symbol):
    result = []
    for transition in self.delta:
        if transition['from_state'] == current_state and transition['symbol'] == input_symbol:
            result.append(transition)
    return result
```

Conclusions:

This lab work was a good hands-on experience to understand the fundamental concepts of formal languages and their characteristics via concrete examples. We understood how to apply these notions to generate new strings adhering to specific rules. Additionally, it taught me how to convert a Grammar into a Finite Automaton and employ it to verify the accuracy of certain words. It is a good experience for the next laboratory assignments.