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

Course: Formal Languages & Finite Automata

Intro to formal languages.
Regular grammars. Finite Automata.

Elaborated:
st. gr. FAF-222 Covalenco Alexandr

Verified: asist. univ. Cretu Dumitru

Chişinău - 2023

Theory

Formal language is commonly used in work-related correspondence and other formal communications. Formal language serves as a form of official correspondence, used in the business world and academic circles

Alphabet - a set of allowed characters.

Vocabulary - a set of allowed words composed of characters from the alphabet.

Grammar - a set of rules that define the structure of the language and how words can be combined into sentences or expressions.

Regular grammars are the simplest type of grammars used to describe formal languages. They define the rules for constructing strings in the language using operations of concatenation (sequential connection), choice (alternation between characters or strings), and iteration (repetition of characters or strings).

Finite automata are mathematical models used for analyzing and designing systems with a finite number of states. They are ideally suited for implementing and analyzing regular grammars, as they can precisely represent the processes occurring within these grammars. Finite automata can be deterministic (DFA), where each state has a uniquely defined transition for each alphabet symbol, or nondeterministic (NFA), where transitions can be ambiguous.

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 description

```python
import random

# Step 1: Define the Grammar class
class Grammar:
    def __init__(self, VN, VT, P, S):
        # Initialize non-terminals (VN), terminals (VT), production rules (P), and start symbol (S)
        self.VN, self.VT, self.P, self.S = VN, VT, P, S
```

```python
    # Step 2: Add a function to generate 5 valid strings
    def generate_valid_strings(self):
        # Generate 5 valid strings using list comprehension
        return [self.generate_string() for _ in range(5)]

    # Helper function to generate a string based on the grammar rules
    def generate_string(self):
        result, changed = [self.S], True

        # Keep generating until no more changes occur
        while changed:
            changed, temp = False, []

            # Iterate through each symbol in the result
            for ch in result:
                # If the symbol is a terminal, add it directly to the result
                if ch in self.VT:
                    temp.append(ch)
                # If the symbol is a non-terminal, replace it with its production rule(s)
                else:
                    # If the symbol has multiple production rules, choose one randomly
                    temp.extend(self.P.get(ch, [ch]))

            # Check if any changes occurred in this iteration
            changed = len(temp) != len(result)
            result = temp

        # Convert the list of symbols into a string
        return ''.join(result)
```

```python
# Step 3: Implement functionality to convert Grammar to Finite Automaton
def to_finite_automaton(self):
    # Define states (Q), start state (q0), final states (F), and transition function (delta)
    Q = set(self.VN)
    q0 = self.S
    F = {symbol for symbol, productions in self.P.items() if not productions}
    delta = {state: {p[0]: p[1] for p in (production[:2] for production in productions if len(production) > 1)} for state, productions in self.P.items()}
    return FiniteAutomaton(Q, self.VT, delta, q0, F)
```

```python
# Step 4: Define the FiniteAutomaton class
class FiniteAutomaton:
    def __init__(self, Q, Sigma, delta, q0, F):
        # Initialize states (Q), input alphabet (Sigma), transition function (delta), start state (q0), and final states (F)
        self.Q, self.Sigma, self.delta, self.q0, self.F = Q, Sigma, delta, q0, F
```

```python
    # Step 5: Add a method to check if an input string can be obtained via state transition
    def string_belongs_to_language(self, input_string):
        current_state = self.q0
        # Simulate the transition of the input string through the automaton
        for symbol in input_string:
            # Check if the current state has a transition for the symbol
            if current_state not in self.delta or symbol not in self.delta[current_state]:
                return False
            # Move to the next state based on the transition
            current_state = self.delta[current_state][symbol]
        # Check if the final state is in the set of final states
        return current_state in self.F

# Step 6: Define the vocabulary
VN, VT = {'S', 'F', 'L'}, {'a', 'b', 'c', 'd'}
P, S = {'S': ['bS', 'aF', 'd'], 'F': ['cF', 'dF', 'aL', 'b'], 'L': ['aL', 'c']}, 'S'
```

```python
# Step 7: Create a Grammar object
grammar = Grammar(VN, VT, P, S)

# Step 8: Generate 5 valid strings
valid_strings = grammar.generate_valid_strings()
print("5 Valid Strings:", *valid_strings, sep='\n')

# Step 9: Convert Grammar to Finite Automaton
automaton = grammar.to_finite_automaton()

# Step 10: Test if an input string belongs to the language of the Finite Automaton
input_string = "ab"
print(f"Does '{input_string}' belong to the language? {automaton.string_belongs_to_language(input_string)}")
```

Conalusions

This laboratory work on "Intro to Formal Languages, Regular Grammars, and Finite Automata" provides a huge understanding of how formal languages are formed and used in computer science. It underlines the importance of defining a language through its alphabet, vocabulary, and grammar, setting the stage for more complex computational theories and applications. By implementing a grammar and generating valid strings, students gain hands-on experience with the mechanisms behind language formation and the practical aspects of programming languages.