Ministerul Educaţiei și 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-223 Nedealcova Irina

Verified:

asist. univ. Cretu Dumitru

Chişinău - 2023

Theory

A formal language serves as a conduit for transmitting information from one party to another. For a language to operate effectively, it requires several essential components:

1. Alphabet: This is a collection of permissible characters.

2. Vocabulary: It comprises allowable words formed from characters in the alphabet.

3. Grammar: This encompasses a set of regulations dictating the structure of the language and how words can be amalgamated to form sentences or expressions.

These elements can be arranged in myriad ways, making language creation a process of judiciously selecting these constituents to best suit its intended purpose. Often, these choices are influenced by preferences, leading to the proliferation of various natural, programming, and markup languages that fulfill similar functions.

Regular grammars, the most basic type of grammars for describing formal languages, delineate rules for constructing strings within the language using concatenation (sequential connection), choice (alternation between characters or strings), and iteration (repetition of characters or strings).

Finite automata, mathematical models employed for analyzing and designing systems with a finite number of states, are particularly well-suited for implementing and scrutinizing regular grammars. They offer precise representations of the processes inherent in these grammars. Finite automata can be deterministic (DFA), where each state has a uniquely defined transition for every 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

This class defines a context-free grammar. It has non-terminal symbols (`VN`), terminal symbols (`VT`), production rules (`P`), and a start symbol (`S`). The `generateString` method recursively expands the grammar based on the production rules to generate strings.

```
class Grammar:
    def __init__(self):
        self.VN = {'S', 'A', 'B', 'C'}  # Non-terminal symbols
        self.VT = {'a', 'b'}  # Terminal symbols
        self.P = {  # Production rules
            'S': ['aA', 'aB'],
            'A': ['bS'],
            'B': ['aC'],
            'C': ['a', 'bS']
        }
        self.S = 'S'  # Start symbol

    def generateString(self):
        # Function to recursively expand and generate strings based on
production rules
        def expand(symbol):
            if symbol in self.VN:
                productions = self.P[symbol]
                chosen_production = random.choice(productions)
                return ''.join(expand(sym) for sym in chosen_production)
            else:
                return symbol
        return expand(self.S)
```

This class represents a deterministic finite automaton (DFA). It has states (`Q`), an alphabet (`Sigma`), a transition function (`delta`), a start state (`q0`), and accepting states (`F`). The `stringBelongsToLanguage` method checks if a given string belongs to the language recognized by the DFA.

```
def grammar_to_FA(grammar):
    # Define finite automaton states, alphabet, transitions, start and
accepting states
    states = {'q0', 'q1', 'q2', 'q3', 'q4', 'qF'}
    alphabet = {'a', 'b'}
    transition_function = {
        ('q0', 'a'): 'q1',
        ('q1', 'b'): 'q2',
        ('q2', 'a'): 'q3',
        ('q3', 'a'): 'q4',
    }
    start_state = 'q0'
    accepting_states = {'q4'}
    return FiniteAutomaton(states, alphabet, transition_function,
start_state, accepting_states)
```

This `main` function demonstrates the usage of the grammar and DFA classes. It generates strings based on the grammar and then checks if a specific test string belongs to the language recognized by the DFA.

```
def main():
    grammar = Grammar()
    print("Generated strings from the grammar:")
    for _ in range(5):
        print(grammar.generateString())

    fa = grammar_to_FA(grammar)
    test_string = "abaababaababaababaababababaabaababababaaa"
    if fa.stringBelongsToLanguage(test_string):
        print(f"String {test_string} belongs to the language.")
    else:
        print(f"String {test_string} does not belong to the language.")

if __name__ == "__main__":
    main()
```

Conclusions

Throughout this laboratory session, participants delved into the fundamental aspects of formal languages, focusing particularly on regular grammars and finite automata. The overarching objective was to comprehensively grasp the theoretical foundations and practical applications of these concepts. Initially, students familiarized themselves with the essential components of formal languages, including alphabets, vocabularies, and grammars, understanding their roles in defining languages and conveying information. Subsequently, practical exercises were

conducted, beginning with the setup of project environments via GitHub repositories and the selection of appropriate programming languages to facilitate implementation tasks.

Students then proceeded to implement a grammar class, which involved defining rules and constraints for language construction. This class was equipped with functionalities to generate valid strings representative of the language defined by the grammar. Moreover, students were tasked with developing conversion mechanisms to translate grammar objects into finite automata, enabling them to explore the relationship between grammars and automata in language representation. Additionally, the implementation of methods within the finite automata class allowed for the validation of input strings against defined state transitions, reinforcing the practical understanding of automata theory.

Overall, this laboratory exercise provided participants with a holistic learning experience, bridging theoretical knowledge with practical application. By engaging in hands-on tasks, students gained invaluable insights into formal language theory, enhancing their problem-solving skills and laying a solid foundation for further exploration in this field.