MINISTERUL EDUCAȚIEI, CULTURII ȘI CERCETĂRII

AL REPUBLICII MOLDOVA Universitatea Tehnică a Moldovei Facultatea Calculatoare, Informatică şi Microelectronică Departamentul Inginerie Software și Automatică

Copta Adrian | FAF-223

Report

Laboratory work n.1

of Formal Languages & Finite Automata

Checked by:

Cretu Dumitru, *university assistant*

DISA, FCIM, UTM

Chișinău – 2023

1. Theory:

Formal Language: A formal language is a precisely defined set of strings (sequences of symbols) constructed from a specific alphabet according to a set of well-defined rules. These rules dictate how the symbols can be combined to form valid words or sentences in the language.

Formal Grammar: Formal grammar is a set of rules that precisely defines the structure and formation of valid strings in a formal language. It consists of production rules that specify how symbols from the alphabet can be combined to generate valid sentences or words in the language. Formal grammars are used to describe the syntax of programming languages, natural languages, and other formal systems.

Finite Automaton: A finite automaton is a computational model characterized by a finite set of states, an alphabet of input symbols, a set of transition rules that define how the automaton transitions between states based on input symbols, and a set of designated initial and accepting states. It processes input symbols sequentially, transitioning between states according to the specified rules, and accepts or rejects an input based on whether the final state is an accepting state. Finite automata are fundamental in the study of formal languages and are classified into different types, such as deterministic and non-deterministic automata.

2. Purpose of the task work:

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;

3. Implementation description:

Grammar class:

The provided class "Grammar" represents a context-free grammar and is initialized with non-terminal symbols (V_n), terminal symbols (V_t), and production rules (P). The class includes a method, generate_string, which generates a string by iteratively replacing non-terminals in the input word using randomly selected production rules until a string with only terminal symbols is formed, adhering to the given grammar rules.

```python
import random

class Grammar:
    def __init__(self, V_n, V_t, P) :
        self.V_n = V_n
        self.V_t = V_t
        self.P = P

    def generate_string(self, word="S"):
        while (not self.check_word(word)):
            for char in word:
                if not self.check_symbol(char):
                    production = self.replace(
                        self.P[char])
                    word = word.replace(char, production)
        return word

    def check_word(self, word):
        for char in word:
            if char in self.V_n:
                return False
        return True

    def check_symbol(self, char):
        if char in self.V_n:
            return False
        return True

    def replace(self, P):
        return random.choice(P)
```

Deterministic Finite Automaton Class:

The class "DFA" represents a deterministic finite automaton (DFA) and is initialized with the set of states (Q), input alphabet (Sigma), transition function (delta) represented as a dictionary, initial state (q0), and set of final states (F). The class includes a check method that takes an input string (w) and determines whether the DFA accepts the input by simulating the transitions based on the input symbols, returning True if the final state is in the set of final states (F), and False otherwise.

```python
class DFA :

    def __init__(self,Q,Sigma,delta,q0,F) :
        self.Q = Q # set of states
        self.Sigma = Sigma # set of symbols
        self.delta = delta # transition function as a dictionary
        self.q0 = q0 # initial state
        self.F = F # set of final states

    def __repr__(self) :
        return f"DFA({self.Q},\n\t{self.Sigma},\n\t{self.delta},\n\t{self.q0},\n\t{self.F})"

    def check (self,w) :
        q = self.q0
        while w!="" :
            q = self.delta[(q,w[0])]
            w = w[1:]
        return q in self.F
```

Command Class: (optionally created)

The class "Command" contains static methods to display a set of commands, prompt the user to input the number of words or a word, and returns the respective inputs for generating strings, checking membership, or other interactions within a command-line interface.

```python
class Command:

    @staticmethod
    def commands():
        print("")
        print("h - Help")
        print("g - Generate strings")
        print("f - Check appartanence")
        print("exit - End program")
        print("")

    @staticmethod
    def command_1():
        print("")
        number_of_words = input("Insert number of words: ")
        print("")
        return int(number_of_words)

    @staticmethod
    def command_2():
        print("")
        word = str(input("Insert word: "))
        print("")
        return word
```

The main:

This code snippet creates instances D0 and D1 representing a grammar and a DFA, respectively, with specific sets of states, symbols, transitions, and final states.

```
from DFA import DFA
from Grammar import Grammar
from Command import Command

D0 = Grammar(["S", "I", "J", 'K'],
            ["b", "c", "e", "n", "m"],
            {"S":["cI"], "I":["bJ", "eK", "e"], "J":["nJ", "cS"], "K":["nK", "m"]})

D1 = DFA({0,1,2,3,4},
         {"c","b","n","e","m"},
         {(0,"c"):1,
          (1,"b"):2,(1,"e"):3,
          (2,"n"):2,(2,"c"):0,
          (3,"n"):4,(3,"e"):3,(3,"m"):3,
          (4,"n"):4,(4,"m"):3},
          0,
          {0,3})
```

This while loop creates a command-line interface that continuously prompts the user for input commands. Depending on the entered command ('h', 'g', 'f', or 'exit'), it executes corresponding actions: displaying help information for commands, generating strings according to the grammar defined by D0, checking the membership of a word in the language recognized by the DFA D1, or ending the program if the command is 'exit'. The loop continues until the user enters 'exit', providing an interactive interface for grammar and automaton interactions.

```python
Command.commands()

while True:

    command = input("Insert command: ")
    if command == 'h':
        Command.commands()

    elif command == 'g':
        number_of_words = Command.command_1()
        words = []
        while (len(words) < number_of_words):
            word = D0.generate_string()
            if word not in words:
                words.append(word)
        print(*map(str, words), sep='\n')
        print("")

    elif command == 'f':
        word = Command.command_2()
        print(D1.check(word))
        print("")

    elif command == "exit":
        print("Program finished.")
        break
```

4. Program execution:

```
h - Help
g - Generate strings
f - Check appartanence
exit - End program

Insert command: g

Insert number of words: 7

cennnm
cem
cennm
cbccenm
cbnnccem
cbccbnnccem
cenm

Insert command: f

Insert word: cennnm

True

Insert command: f

Insert word: cen

False

Insert command: h

h - Help
g - Generate strings
f - Check appartanence
exit - End program

Insert command: exit
Program finished.
PS D:\FAF\DSL>
```

5. Conclusion:

In this laboratory work, we explored fundamental concepts in formal language theory and automata. We began by defining and implementing a context-free grammar using the Grammar class, followed by the creation of a deterministic finite automaton (DFA) using the DFA class.

Throughout the laboratory work, we gained practical experience in working with formal grammars, finite automata, and command interfaces. The implementation showcased the application of these concepts in generating strings adhering to grammar rules and determining whether a given word is accepted by a DFA. This hands-on exploration provided valuable insights into the foundational principles of formal language theory and its practical implications.