Laboratory Work Nr.1 Grammars and Finite Automata

Course: Formal Languages & Finite Automata

Author: Tofan Liviu

Group: FAF-223

Theory

A formal language serves as the framework for conveying information from a sender to a receiver. It comprises three key elements:

Alphabet: The set of valid characters.

Vocabulary: The set of valid words.

Grammar: The set of rules or constraints governing the language.

In the world of formal language theory, grammars and automata are closely connected. Think of a grammar as a set of rules that creates strings, like sentences in a language. On the other hand, an automaton is like a detective that checks if a string belongs to that language.

When we turn a grammar into a finite automaton, it helps us figure out if a specific string fits the language described by the grammar. It's like having a tool to confirm whether a sentence is correct according to the rules of the language.

Objectives:

According to the 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

Grammar Class:

The Grammar class is designed to represent a formal language with nonterminals, terminals, language rules, and a designated starting symbol. The generate_string() method generates strings from the language, replacing nonter-

minals with random productions until a valid terminal word is formed. The generate_words() function produces a list of 5 valid strings based on the grammar. Additionally, the class has the capability to convert itself into a finite automaton and check if a given string can be obtained through state transitions in the corresponding finite automaton.

```
import random

class Grammar:
    def __init__(self, _VN: list, _VT: list, _L: dict, _S: str) -> None:
        self.nonterminals = _VN
        self.terminals = _VT
        self.language = _L
        self.start = _S

    def generate_string(self, word=None) -> str:
        current_word = self.start if word is None else word
        while not self.terminal_word(current_word):
            for char in current_word:
                if not self.terminal_char(char):
                    production = self.__replace(self.language[char])
                    current_word = current_word.replace(char, production, 1)  # Replace only
        return current_word

    def terminal_word(self, word: str) -> bool:
        # Check if all char from word are terminals
        return all(char in self.terminals for char in word)

    def terminal_char(self, char: str) -> bool:
        # Check if char is terminal
        return char in self.terminals

    def __replace(self, value: list) -> str:
        # Choose random from values of dictionary of given char
        return random.choice(value)

    def generate_words(self, num_strings=5) -> list:
        ans = []
        while len(ans) < num_strings:
            word = self.generate_string()
            if word not in ans:
                ans.append(word)
        return ans
```

FiniteAutomaton Class:

The FiniteAutomaton class is designed to represent a finite automaton converted from a given grammar. It includes the set of states (Q), the alphabet (Alphabet), the start state (q0), state transitions (delta), and the set of accepting states (F). The class method grammar_to_DFA() converts a Grammar object into a corresponding deterministic finite automaton (DFA). The word_belongs_to_language() method checks if a given string belongs to the language recognized by the finite automaton, traversing state transitions based on the input string.

```
from Grammar import Grammar

class FiniteAutomaton:
    def __init__(self, grammar: Grammar, _delta: dict) -> None:
        self.Q = grammar.nonterminals + ["X"]
        self.Alphabet = grammar.terminals
        self.q0 = grammar.start
        self.delta = _delta
        self.F = ["X"]

    @classmethod
    def grammar_to_DFA(cls, grammar: Grammar) -> 'FiniteAutomaton':
        delta = {}

        for nonterminal in grammar.language:
            for production in grammar.language[nonterminal]:
                if len(production) > 1:
                    transition = production[0]
                    result_state = production[1]
                    delta.setdefault(nonterminal, {})[transition] = result_state
                else:
                    transition = production
                    result_state = "X"
                    delta.setdefault(nonterminal, {})[transition] = result_state

        return cls(grammar, delta)

    def word_belongs_to_language(self, string: str) -> bool:
        state = self.q0
        for char in string:
            if char not in self.Alphabet:
                return False
            if char in self.delta[state]:
                state = self.delta[state][char]
            else:
```

```
                return False
        return state in self.F
```

Main.py

In the provided main script:

Point A, B: It creates an instance of the Grammar class using the given language definition. Then, it generates 5 valid strings starting with the nonterminal 'S' using the generate_words method and prints them.

Point C: It converts the Grammar instance to a finite automaton (DFA) using the grammar_to_DFA class method of the FiniteAutomaton class.

Point D: It checks if the input string "FAF-223" belongs to the language recognized by the created DFA using the word_belongs_to_language method and prints the result.

Overall, the main script demonstrates the usage of the Grammar and FiniteAutomaton classes, including string generation, grammar-to-DFA conversion, and language recognition.

```
from Grammar import Grammar
from Automaton import FiniteAutomaton

language = {
    'S': ['bS', 'dA'],
    'A': ['aA', 'dB', 'b'],
    'B': ['cB', 'a']
}

# Create an object of the Grammar class
grammar_instance = Grammar(list(language), ['a', 'b', 'c', 'd'], language, "S")

# Point A, B: Generate 5 valid strings starting with S
generated_strings = grammar_instance.generate_words()

for word in range(len(generated_strings)):
    print("Word", word+1, ":", generated_strings[word])

# Point C: Convert Grammar to FiniteAutomaton
finite_automaton_instance = FiniteAutomaton.grammar_to_DFA(grammar_instance)

print("\nFinite Automaton Details:")
print("States:", finite_automaton_instance.Q)
print("Alphabet:", finite_automaton_instance.Alphabet)
print("Start State:", finite_automaton_instance.q0)
print("Transitions:")
```

```python
for state, transitions in finite_automaton_instance.delta.items():
    for symbol, result_state in transitions.items():
        print(f"({state}, {symbol}) -> {result_state}")

# Point D: check if the word can be obtained
input_string = "FAF-223"
result = finite_automaton_instance.word_belongs_to_language(input_string)

print(f"\nThe string '{input_string}' belongs to the language: {result}")
```

Conclusions

In this laboratory work, a Python project was developed to explore formal languages, grammars, and finite automata. The Grammar class was created to define formal languages, and the FiniteAutomaton class allowed for converting grammars to deterministic finite automata (DFA) and checking if a string belongs to the recognized language. The Main script demonstrated the project's functionality, including generating strings and checking language membership. The project was organized using a GitHub repository for version control.

Results

*Variant 25:

VN={S, A, B}, VT={a, b, c, d}, P={ S → bS
S → dA
A → aA
A → dB
B → cB
A → b
B → a }

And the results, output:

```
Word 1 : bdb
Word 2 : bbbbbbdb
Word 3 : daaada
Word 4 : db
Word 5 : ddca

Finite Automaton Details:
States: ['S', 'A', 'B', 'X']
Alphabet: ['a', 'b', 'c', 'd']
Start State: S
Transitions:
(S, b) -> S
(S, d) -> A
```

```
(A, a) -> A
(A, d) -> B
(A, b) -> X
(B, c) -> B
(B, a) -> X

The string 'FAF-223' belongs to the language: False
```

References

1) General Information: https://else.fcim.utm.md/pluginfile.php/110458/mod_resource/content/0/LFPC_
2) Finite Automata to Regular Grammar: http://www.cs.um.edu.mt/gordon.pace/Research/Software/Relic
3) Indian Youtube Videos: https://www.youtube.com/watch?v=Qa6csfkK7_I&list=PLBlnK6fEyqRgp46K