Laboratory Work Nr.1 Grammars and Finite Automata

Cretu Cristian

February 20, 2024

Theory

In formal language theory, there is a close relationship between grammars and automata. A grammar defines a language by generating strings, while an automaton recognizes or accepts strings from a language. The conversion from a grammar to a finite automaton enables us to verify whether a given string belongs to the language defined by the grammar.

Objectives

1. Create a Grammar class that represents the grammar of your language.
2. Make the Grammar produce strings that belong to your language.
3. Implement an algorithm that transforms your Grammar into a Finite Automaton.
4. Implement an algorithm that checks if a string belongs to the language defined by the Grammar using the Finite Automaton.

Implementation description

Variant Class

A simple utility class that extracts my variant from the file.

```
class Variant:
    def __init__(self, _variantpath):
        self.variantpath = _variantpath
```

Grammar Class

This class is responsible for storing the data about the symbols that compose the grammar and use them to generate strings or transform to Finite Automaton.

```
class Grammar:
    def __init__(self, _VN: list, _VT: list, _P: dict) ->
    None:
        self.nonterminals = _VN
        self.terminals = _VT
        self.productions = _P

    '''
    This generates a string that belongs to the language of
    the grammar. If
    a terminal has more than one production, it will choose
    one at random.
    '''

    def generate_string(self, word="S") -> str:
        while (not self.word_is_terminal(word)):
            for char in word:
                if not self.char_is_terminal(char):
                    production = self.__pick_replacement(
                        self.productions[char])
                    word = word.replace(char, production)
        return word

    def word_is_terminal(self, word: str) -> bool:
        for char in word:
            if char in self.nonterminals:
                return False
        return True

    def char_is_terminal(self, char: str) -> bool:
        if char in self.nonterminals:
            return False
        return True

    def __pick_replacement(self, productions: list) -> str:
        return random.choice(productions)

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

```

```
    '''This part transforms the Grammar into a Finite Automation object,
    which can be used to check if a string belongs to the language of the grammar.
    It follows the algorithm presented by Mrs. Cojuhari at the course.'''

    def to_finite_automation(self):
        Q = self.nonterminals
        Q.append("X")
        Sigma = self.terminals
        q0 = "S"
        F = ["X"]
        delta = {}
        '''The delta function is a dictionary of dictionaries,
        where the first key is the terminal'''
        for terminal in self.productions:
            for production in self.productions[terminal]:
                if len(production) > 1:
                    transition = production[0]
                    result_state = production[1]
                    if terminal not in delta:
                        delta[terminal] = {}
                    delta[terminal][transition] = result_state
                else:
                    transition = production
                    result_state = "X"
                    if terminal not in delta:
                        delta[terminal] = {}
                    delta[terminal][transition] = result_state
        return FiniteAutomation(Q, Sigma, q0, F, delta)
```

Finite Automaton Class

This class represents the Finite Automaton and allows checking if a string belongs to a language.

```
class FiniteAutomation:
    def __init__(self) -> None:
        self.Q = []
        self.Sigma = []
        self.delta = {}
        self.q0 = ""
        self.F = []

    def __init__(self, _Q:list, _Sigma:list, _q0:str, _F:list,_delta:dict) -> None:
```

```
        self.Q = _Q
        self.Sigma = _Sigma
        self.delta = _delta
        self.q0 = _q0
        self.F = _F

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

Main File

```
from Variant import Variant
from Grammar import Grammar

'''Get the variant from the restructured to json file'''
variant = Variant('/home/cristi/Documents/GitHub/LabsLFA/
    Lab1/variant.json')
VN = variant.getVN()
VT = variant.getVT()
P = variant.getP()
print("My variant is: ")
print(VN)
print(VT)
print(P)
print()

'''Create the grammar and the finite automation'''
g = Grammar(VN, VT, P)
'''Press Ctrl+Left Click on Grammar to see the
    implementation'''
fa = g.to_finite_automation()
print(g.generate_strings())

'''Test the grammar by generating 5 strings and checking if
    they belong to the language'''
for i in range(5):
    test = g.generate_strings()
    for string in test:
        if not fa.string_belongs_to_language(string):
            print(f'{string} did not pass the test')
    print("Test passed!")
```

```
28
29 '''Test if some random strings belong to the language'''
30 print(fa.string_belongs_to_language("a"))
31 print(fa.string_belongs_to_language("ab"))
32 print(fa.string_belongs_to_language("ba"))
33 print(fa.string_belongs_to_language("FAFTOPCIK"))
```

Conclusions / Screenshots / Results

The program successfully converts the grammar to a finite automaton and demonstrates the recognition of valid strings by the automaton.

```
1 My variant is:
2 ['S', 'D', 'E', 'J']
3 ['a', 'b', 'c', 'd', 'e']
4 {'S': ['aD'], 'D': ['dE', 'bJ', 'aE'], 'J': ['cS'], 'E': ['e
    ', 'aE']}
5
6 ['abcaaaaae', 'aae', 'abcabcaaaae', 'abcaaae', 'aaae']
7 Test passed!
8 Test passed!
9 Test passed!
10 Test passed!
11 Test passed!
12 False
13 False
14 False
15 False
```