MINISTRY OF EDUCATION, CULTURE AND RESEARCH OF REPUBLIC OF MOLDOVA
TECHNICAL UNIVERSITY OF MOLDOVA
FACULTY OF COMPUTERS, INFORMATICS AND MICROELECTRONICS
DEPARTMENT OF SOFTWARE ENGINEERING AND AUTOMATICS

Formal Languages Finite Automata

Laboratory work 2 : Determinism in Finite Automata .Conversion from NDFA 2 DFA. Chomsky Hierarchy.

Elaborated:
st.gr. FAF-211 Andrei Ceban
Verified:
asist.univ. Vasile Drumea

Chișinău, 2023

Content

Introduction

A finite automaton is a mechanism used to represent processes of different kinds. It can be compared to a state machine as they both have similar structures and purpose as well. The word finite signifies the fact that an automaton comes with a starting and a set of final states. In other words, for process modeled by an automaton has a beginning and an ending.

Based on the structure of an automaton, there are cases in which with one transition multiple states can be reached which causes non determinism to appear. In general, when talking about systems theory the word determinism characterizes how predictable a system is. If there are random variables involved, the system becomes stochastic or non deterministic.

That being said, the automata can be classified as non-/deterministic, and there is in fact a possibility to reach determinism by following algorithms which modify the structure of the automaton.

The process of converting a non-deterministic finite automaton (NFA) to a deterministic finite automaton (DFA) is known as the subset construction method. The idea is to construct a DFA whose states represent sets of states of the NFA, where the DFA's transitions are determined by the transitions of the NFA.

Determinism is an important concept in the theory of finite automata. A deterministic finite automaton (DFA) is a mathematical model used to recognize regular languages. It consists of a finite set of states, a finite set of input symbols (the alphabet), a transition function that maps each state and input symbol to a unique next state, a start state, and a set of accepting states.

The key property of a DFA is that for each input string, there is exactly one possible sequence of state transitions that the DFA can make. This means that the behavior of a DFA is completely determined by its current state and the input symbol it receives.

In contrast, a non-deterministic finite automaton (NFA) allows for multiple possible transitions from a given state on a given input symbol. This means that the behavior of an NFA is not uniquely determined by its current state and input symbol. NFAs are more powerful than DFAs in the sense that they can recognize some languages that DFAs cannot, but they are also more difficult to analyze and implement.

Determinism has important practical implications for the use of finite automata in programming and language recognition. Deterministic automata are easier to implement and reason about, and they are also more efficient in terms of memory and processing time. For these reasons, most practical applications of finite automata use deterministic models.

1 Objectives

1. Understand what an automaton is and what it can be used for.

2. Continuing the work in the same repository and the same project, the following need to be added:

    a. Provide a function in your grammar type/class that could classify the grammar based on Chomsky hierarchy.

    b. For this you can use the variant from the previous lab.

3. According to your variant number (by universal convention it is register ID), get the finite automaton definition and do the following tasks:

    a. Implement conversion of a finite automaton to a regular grammar.

    b. Determine whether your FA is deterministic or non-deterministic.

    c. Implement some functionality that would convert an NDFA to a DFA.

    d. Represent the finite automaton graphically (Optional, and can be considered as a bonus point).

2 Implementation

2.1 Code:

```
Main ------------------------------------------------------------------------
import Grammar
import ChomskyHierarchy

# Varianta 8

# Set of terminal symbols
VT = {'a', 'b', 'c', 'd', 'e'}

# Set of non-terminal symbols
VN = {'S', 'D', 'E', 'J'}

# Set of productions, rules or transitions
P = {"S": ["aD"],"D": ["dE", "bJ", "aE"],"J": ["cS"],"E": ["eX", "aE"],"X": " " }

# Initiate the grammar calling the constructor with all 3 params
grammar = Grammar.Grammar(VN, VT, P)

# Initiate the finite automaton
# finite_automaton = grammar.from_grammar_to_finite_automaton()

# Generate 5 string
word_1 = grammar.generate_string()
word_2 = grammar.generate_string()
word_3 = grammar.generate_string()
word_4 = grammar.generate_string()
word_5 = grammar.generate_string()

# Print the strings including the  validation
```

```
# print("word_1 = {x} ".format(x = word_1) + " | " + str(finite_automaton.check_stri
# print("word_2 = {x} ".format(x = word_2) + " | " + str(finite_automaton.check_stri
# print("word_3 = {x} ".format(x = word_3) + " | " + str(finite_automaton.check_stri
# print("word_4 = {x} ".format(x = word_4) + " | " + str(finite_automaton.check_stri
# print("word_5 = {x} ".format(x = word_5) + " | " + str(finite_automaton.check_stri

#Lab 2 ------------------------------------------------------------------------
#Clasify grammar for Lab1
#grammar = Grammar.Grammar(VN, VT, P)
# classify the grammar based on Chomsky hierarchy
get_classify = grammar.check_Grammar(P)

print(f"Grammar clasification LAB1 :  {get_classify}")



states = ['q0', 'q1', 'q2', 'q3', 'q4']
alphabet = ['a', 'b']
F = ['q3']
start_state = 'q0'
transitions =  {('q0', 'a'): {'q1'},
                ('q1', 'b'): {'q1','q2'},
                ('q2', 'b'): {'q0'},
                ('q3', 'a'): {'q4'},
                ('q4', 'a'): {'q0'},
                ('q2', 'a'): {'q3'},}


# Get the constructor with farams to  check the automation
Automation = ChomskyHierarchy.ChomskyHierarchy(states, alphabet, F, start_state, tra

# Check if automaton is deterministic
is_deterministic = Automation.check_deterministic()
print(f"Is automaton deterministic? {is_deterministic}")
```

```
# Convert NDFA to DFA
dfa = Automation.NDFA_to_a_DFA()
print(f" The states: {dfa.states}")
print(f" The transition function: {dfa.transitions}")
print(f" The initial state: {dfa.q0}")
print(f" The final states: {dfa.F}")

# Convert automaton to regular grammar
grammar = Automation.finite_to_grammar()
print(f"Regular grammar productions: {grammar}")
```

Grammar ----------

```
import FiniteAutomaton
import random

class Grammar:
    # Constructor of Grammar class
    def __init__(self, VN, VT, P):
        self.VT = VT
        self.VN = VN
        self.P = P

    # Generate the strings
    def generate_string(self):
        # Start of the string
        string = "S"
        # Set of final states
        final_state = " "

        # Create a random string
```

```
    while string[-1] not in final_state:
        arr = []
        #go trough items and get the keys and values
        for key, value in self.P.items():
            if key == string[-1]:
                #move the value in array
                arr += value
        if not arr:
            return None
        production = random.choice(arr)
        #Reverse string, count backwards
        string = string[:-1] + production
    return string

# Convert from grammar to finite automaton, sending all the params with self
def from_grammar_to_finite_automaton(self):
    # Initiate finite set of states
    Q = set(self.VN)
    # Initiate the alphabet with all the characters
    Sigma = set(self.VT)
    # Initiate the initial state from the list
    q0 = "S"
    # Initiate the set of final states from the list
    F = "X"
    # Initiate the transition function
    arr = {}
    for key, value in self.P.items():
        for symbol in value:
            if (key, symbol[0]) in arr :
                arr[(key, symbol[0])].append(symbol[1:])
            else:
                arr[(key, symbol[0])] = [symbol[1:]]
    print("Automation :", arr)
```

```
    # Call the constructor of Finite Automaton with all the params
    return FiniteAutomaton.FiniteAutomaton(Q, Sigma, arr, q0, F)

def check_Grammar(self, P):
    if self.check_Context_free(P):
        return "Type context free"
    if self.check_Context_sensitive(P):
        return "Type context sensitive"

    if self.check_Regular(P):
        return "Type regular"

    if self.check_Unrestricted(P):
        return "Type unrestricted"

    return "Not in Chomsky Hierarchy"

def check_Context_sensitive(self, P):

    for symbol, string in P.items():
        for rhs in string:
            if len(rhs) < len(symbol):
                return False
            for i, symbol in enumerate(rhs):
                if symbol in P and i != len(rhs) - 1:
                    if len(rhs) <= len(symbol):
                        return False
    return True

def check_Context_free(self, P):

    for symbol, string in P.items():
        if len(symbol) != 1 or not symbol.isupper():
            return False
```

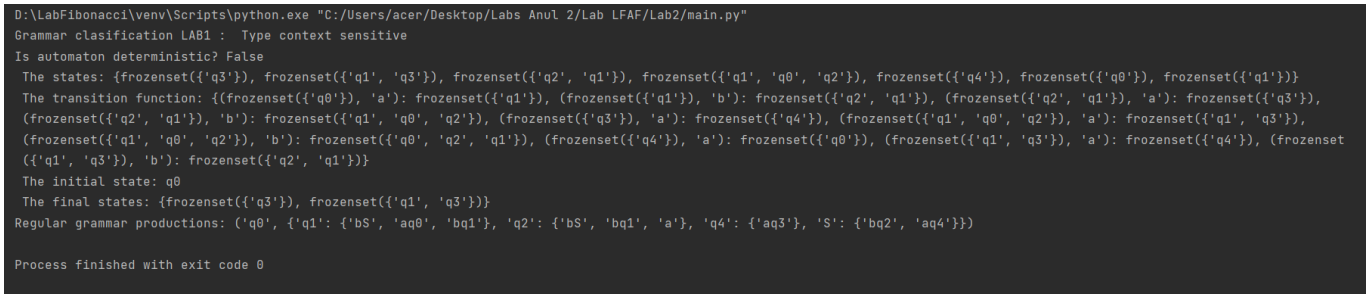
```
            for rhs in string:
                for symbol in rhs:
                    if symbol not in P and not symbol.islower():
                        return False
        return True

    def check_Regular(self, P):

        for symbol, string in P.items():
            if not symbol.isupper():
                return False
            for rhs in string:
                if len(rhs) == 1 and rhs.islower():
                    continue
                elif len(rhs) == 2 and rhs[0].islower() and rhs[1].isupper():
                    continue
                else:
                    return False
        return True

    def check_Unrestricted(self, P):
        return True


ChomskyHierarchy ----------------------------------------------------------------

class ChomskyHierarchy:
    #initiate the constructor to set the parameters
    def __init__(self, states, alphabet, F, q0, transitions):
        self.states = states
        self.alphabet = alphabet
        self.F = F
        self.q0 = q0
        self.transitions = transitions
```

```
#Determine whether your FA is deterministic or non-deterministic
def check_deterministic(self):
    for state in self.states:
        for symbol in self.alphabet:
            next_states = self.transitions.get((state, symbol), set())
            if len(next_states) != 1:
                return False
    return True

#Implement some functionality that would convert an NDFA to a DFA
def NDFA_to_a_DFA(self):
    if self.check_deterministic():
        return self

    dfa_states = set()
    dfa_F = set()
    dfa_transitions = dict()
    state_queue = [frozenset([self.q0])]
    while state_queue:
        current_states = state_queue.pop(0)
        dfa_states.add(current_states)
        if any(state in self.F for state in current_states):
            dfa_F.add(current_states)
        for symbol in self.alphabet:
            next_states = set()
            for state in current_states:
                next_states |= set(self.transitions.get((state, symbol), set()))
            if next_states:
                next_states = frozenset(next_states)
                dfa_transitions[(current_states, symbol)] = next_states
                if next_states not in dfa_states:
                    state_queue.append(next_states)
```

```
    dfa = ChomskyHierarchy(self, self.alphabet, self.F, self.q0, self.transition
    dfa.states = dfa_states
    dfa.F = dfa_F
    dfa.transitions = dfa_transitions

    return dfa

# Implement conversion of a finite automaton to a regular grammar.
def finite_to_grammar(self):
    product = dict()
    for state in self.states:
        for symbol in self.alphabet:
            next_states = self.transitions.get((state, symbol), set())
            for next_state in next_states:
                if next_state in self.F:
                    if state not in product:
                        product[state] = set()
                    product[state].add(symbol)
                else:
                    if next_state not in product:
                        product[next_state] = set()
                    product[next_state].add(symbol + state)
    start_symbol = self.q0
    if start_symbol in product:
        product['S'] = product[start_symbol]
        del product[start_symbol]
        for state in self.states:
            for symbol in self.alphabet:
                next_states = self.transitions.get((state, symbol), set())
                for next_state in next_states:
                    if state in product and symbol + state in product[state]:
                        if next_state not in product:
                            product[next_state] = set()
                        product[next_state].add(symbol + 'S')
```

```
else:
    start_symbol = 'S'
    product[start_symbol] = set()
    for FF in self.F:
        product[start_symbol].add('eps' + FF)

return start_symbol, product
```

2.2 Screenshot:



Conclusions

After doing this laboratory work,i understand that determinism in finite automata is an important concept in computer science that refers to the ability of a machine to uniquely determine its next state based on the current input and state. Deterministic finite automata (DFA) are particularly useful for pattern recognition and language processing tasks, as they can efficiently recognize regular languages.

The process of converting a non-deterministic finite automaton (NDFA) to a DFA involves transforming the NDFA into a deterministic version of itself, where each input symbol leads to exactly one next state. This conversion is important for simplifying the machine and making it easier to analyze and understand.

Also, using the Chomsky hierarchy which is a classification of formal languages into four categories based on their complexity and the types of grammars that can generate them. These categories include regular languages, context-free languages, context-sensitive languages, and recursively enumerable languages. Each category has its own set of rules and limitations, and understanding these categories can help in the design and analysis of computational systems.

In Conclusion, all the functions that i made during this lab are NDFATOADFA which convert a NDFA to a DFA, checkDeterministic that verify if the Finite Automata is deterministic or non-deterministic and finiteTogrammar which takes the finite automation and convers it to to grammar. These functions helped me to understand better the basics concepts of determinism, NDFA , DFA and Chomsky Hierarchy.