Ministerul Educaţiei și Cercetării al Republicii Moldova
Universitatea Tehnică a Moldovei
Facultatea Calculatoare, Informatică și Microelectronică

Laboratory work nr. 2
Course: Formal languages and finite automata
Topic: Determinism in Finite Automata. Conversion from NDFA to DFA. Chomsky Hierarchy.

Elaborated:
st. gr. FAF-221 Călugăreanu Ana

Verified:
asist. univ. Cretu Dumitru

Chişinău - 2024

Theory

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.

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):

    You can use external libraries, tools or APIs to generate the figures/diagrams.

    Your program needs to gather and send the data about the automaton and the lib/tool/API return the visual representation.

Implementation Description

This lab report explores the theory and application of finite automata (FA), their conversion into regular grammars (RG), determination of determinism, transformation into deterministic finite automata (DFA), and graphical representation. The provided Python code serves as a practical implementation, demonstrating the manipulation and analysis of a finite automaton within a computational context.

Task a:

Conversion of FA to Regular Grammar The convert_FA_to_RG function converts a finite automaton into a regular grammar. The conversion process iterates over the transition rules of the FA, mapping each state and symbol pair to its corresponding next states. For each transition, it constructs grammar rules where the left-hand side is the current state and the right-hand side is the concatenation of the symbol and the next state. If the next state is a final state, a rule ending in just the symbol is also added. This allows for the representation of the FA's accept states within the regular grammar format.

```python
# Task a: Convert FA to Regular Grammar
def convert_FA_to_RG(FA):
    RG = {}
    for (state, symbol), next_states in FA['transitions'].items():
        for next_state in next_states:
            rule_key = state
            rule_value = symbol + next_state
            if next_state in FA['final_states']:
                RG.setdefault(rule_key, []).append(symbol)
            RG.setdefault(rule_key, []).append(rule_value)
    return RG
```

Task b: Determinism Check

The `is_deterministic` function checks whether the provided FA is deterministic or non-deterministic. It inspects each transition in the FA; if any state-symbol pair maps to more than one subsequent state, the FA is considered non-deterministic, otherwise, it is deterministic. This check is crucial for understanding the type of finite automaton we are dealing with and its implications on the complexity and capabilities of language recognition.

```python
# Task b: Determine if the FA is deterministic or non-deterministic
def is_deterministic(FA):
    for (state, symbol), next_states in FA['transitions'].items():
        if len(next_states) > 1:
            return False
    return True
```

Task c: Conversion from NDFA to DFA (Subset Construction)

The `convert_NDFA_to_DFA` function applies the subset construction algorithm to convert a non-deterministic finite automaton (NDFA) into a deterministic finite automaton (DFA). It initializes new DFA states, transitions, and final states. The function iterates through each combination of states and symbols to create new transitions based on the union of next states in the NDFA. This process continues until all new states are processed, resulting in a DFA that recognizes the same language as the original NDFA.

```python
def convert_NDFA_to_DFA(FA):
    new_states = set(['q0'])
    new_final_states = set()
    new_transitions = {}
    unprocessed_states = [set(['q0'])]

    while unprocessed_states:
        current_new_state = unprocessed_states.pop()
        for char in FA['alphabet']:
            next_new_state = set()
            for state in current_new_state:
                if (state, char) in FA['transitions']:
                    next_new_state.update(FA['transitions'][(state, char)])
            if next_new_state:

                for next_state in next_new_state:
                    transition_key = (''.join(sorted(current_new_state)), char)
                    new_transitions[transition_key] = next_state
                    if next_state not in new_states:
                        new_states.add(next_state)
                        unprocessed_states.append(set([next_state]))
                    if next_state in FA['final_states']:
                        new_final_states.add(next_state)
```

```
    return {
        'states': new_states,
        'alphabet': FA['alphabet'],
        'final_states': new_final_states,
        'transitions': new_transitions
    }
```

Task d: Graphical Representation of the FA

The `draw_FA` function uses the Graphviz library to create a visual representation of the finite automaton. It sets up nodes for each state, distinguishing final states with a double circle. It also creates directed edges for each transition, labeled with the appropriate symbol. This graphical representation aids in understanding the structure and function of the FA.

```
def draw_FA(FA, filename='FA'):
    dot = graphviz.Digraph(comment='Finite Automaton')
    dot.attr(rankdir='LR', size='8')

    dot.node('start', shape='none')
    for state in FA['states']:
        if state in FA['final_states']:
            dot.node(state, shape='doublecircle')
        else:
            dot.node(state)
        if 'q0' in state:
            dot.edge('start', state)

    for (state, symbol), next_states in FA['transitions'].items():
        for next_state in next_states:
            dot.edge(state, next_state, label=symbol)

    dot.render(filename, view=True)
```

Results

The script processes a predefined finite automaton, executing tasks to convert it to regular grammar, determine its determinism, convert it from NDFA to DFA if necessary, and graphically represent the FA. The output includes the corresponding regular grammar, the determinism status, the Chomsky classification of the RG, the states and transitions of the constructed DFA, and a visual diagram of the original FA.

```
Regular Grammar: {'q0': ['aq2', 'aq1'], 'q1': ['bq1', 'aq2'], 'q2': ['aq1', 'b', 'bq3']}
The FA is Non-deterministic
The RG is Type 2 (Context-Free Grammar)
DFA States: {'q1', 'q0', 'q3', 'q2'}
DFA Transitions: {('q0', 'a'): 'q1', ('q1', 'a'): 'q2', ('q1', 'b'): 'q1', ('q2', 'a'): 'q1', ('q2', 'b'): 'q3'}
```

Screenshots

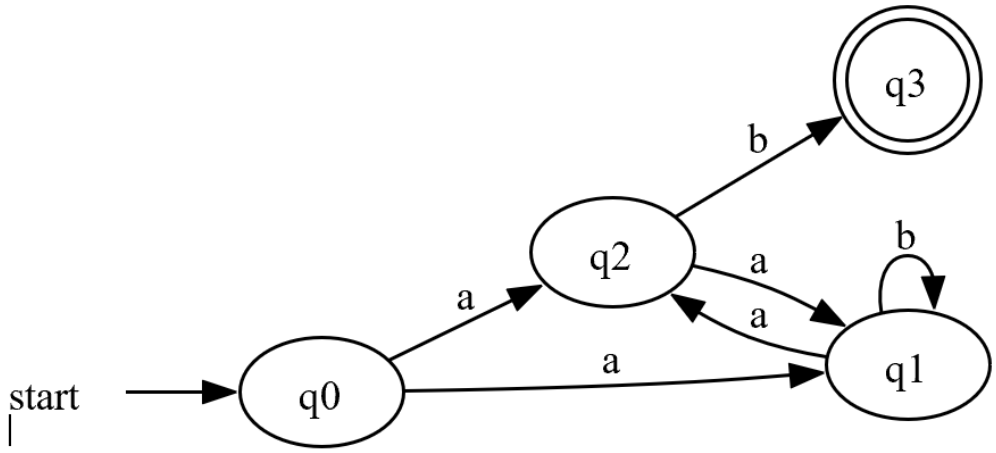

Figure 1: Results of testing

Conclusions

In conclusion, this lab has provided a comprehensive exploration of finite automata, grammars, and their interrelationships. We started by defining a finite automaton class and demonstrated how to convert it into a grammar representation. Through this conversion process, we gained insight into the structure and behavior of grammars corresponding to finite automata.

Furthermore, we explored deterministic and non-deterministic finite automata, as well as their transformations. By implementing algorithms for converting non-deterministic finite automata to deterministic finite automata, we illustrated the concept of determinism and its importance in automata theory.

Overall, this lab deepened our understanding of formal language theory and its practical applications. By analyzing and manipulating finite automata and grammars, we gained valuable insights into their theoretical underpinnings and practical implications. These concepts are fundamental in the study of computer science and play a crucial role in various areas such as compiler design, natural language processing, and algorithm development.