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

Course: Formal Languages & Finite Automata

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.

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.

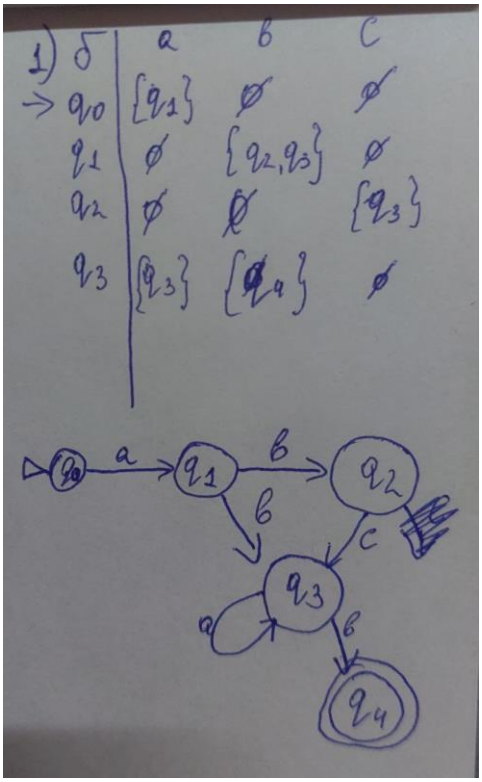
d. Represent the finite automaton graphically (Optional, and can be considered as a bonus point):

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

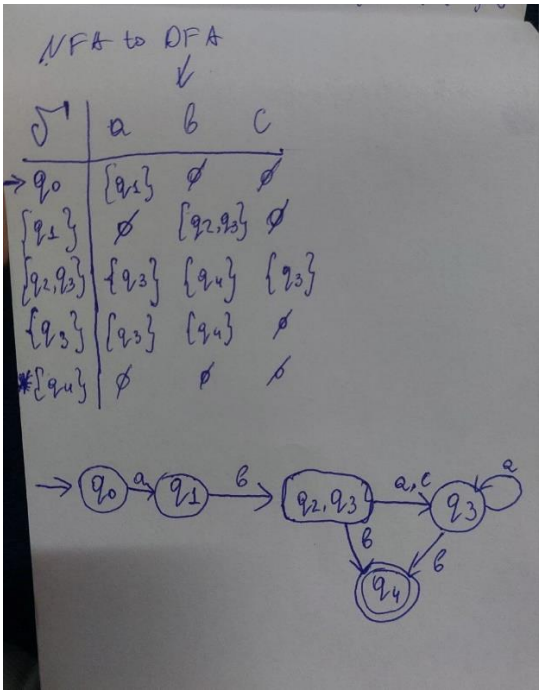
Your program needs to gather and send the data about the automaton and the lib/tool/API return the visual representation.

Implementation:

In the initial step of converting the NFA to a DFA, we defined the transition function of the given NFA. This involves specifying the transitions for each state and input symbol. The NFA transition function serves as the basis for the the conversion process.



With this table and automaton, we can start the conversion to DFA and build the automaton. We take care of specifying which is initial and final state. The process is writing the initial state and its transitions, then the new formed state we will stake into account and write its transitions to create new states. Then we continue this till we have no new states.



Code Implementation:

The convertNDFAtoDFA method takes an NDFA represented as a map of state transitions and an initial state. Within this method, the epsilon closure of states is calculated, ensuring all reachable states, including those reachable through epsilon transitions, are included. The method iterates over the alphabet of the NDFA, constructing the corresponding DFA transitions for each state. Once the conversion is complete, the resulting DFA transition table is returned.

The main method serves as the entry point, defining the NDFA transition function, invoking the conversion method, and printing the resulting DFA transition table.

Conclusion:

Regular grammars and finite automata are fundamental models for describing regular languages. By converting between these two formalisms, researchers and practitioners gain valuable insights into the structure and computational properties of regular languages, enabling them to develop efficient algorithms for language recognition, compiler construction, and pattern matching.