

**MINISTERUL EDUCAȚIEI, CULTURII ȘI CERCETĂRII AL REPUBLICII MOLDOVA**

**Universitatea Tehnică a Moldovei**

**Facultatea Calculatoare, Informatică şi Microelectronică**

**Departamentul Inginerie Software și Automatică**

**Tabanschi Nichita FAF-222**

**Report**

*Laboratory work n.2*

***of Limbaje Formale și Automate***

Checked by:

**Dumitru Cretu**, *university assistant*

DISA, FCIM, UTM

**Chișinău – 2024**

## Overview

    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.

* 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*):
     1. You can use external libraries, tools or APIs to generate the figures/diagrams.
     2. Your program needs to gather and send the data about the automaton and the lib/tool/API return the visual representation.

Please consider that all elements of the task 3 can be done manually, writing a detailed report about how you've done the conversion and what changes have you introduced. In case if you'll be able to write a complete program that will take some finite automata and then convert it to the regular grammar - this will be a good bonus point.

Variant 28

Q = {q0,q1,q2,q3},

∑ = {a,b,c},

F = {q3},

δ(q0,a) = q0,

δ(q0,a) = q1,

δ(q1,a) = q1,

δ(q1,c) = q2,

δ(q1,b) = q3,

δ(q0,b) = q2,

δ(q2,b) = q3.

Let's start by analyzing the given finite automaton (FA) and determining whether it is deterministic or non-deterministic. Then we'll proceed to implement the conversion to a regular grammar.

Given FA:

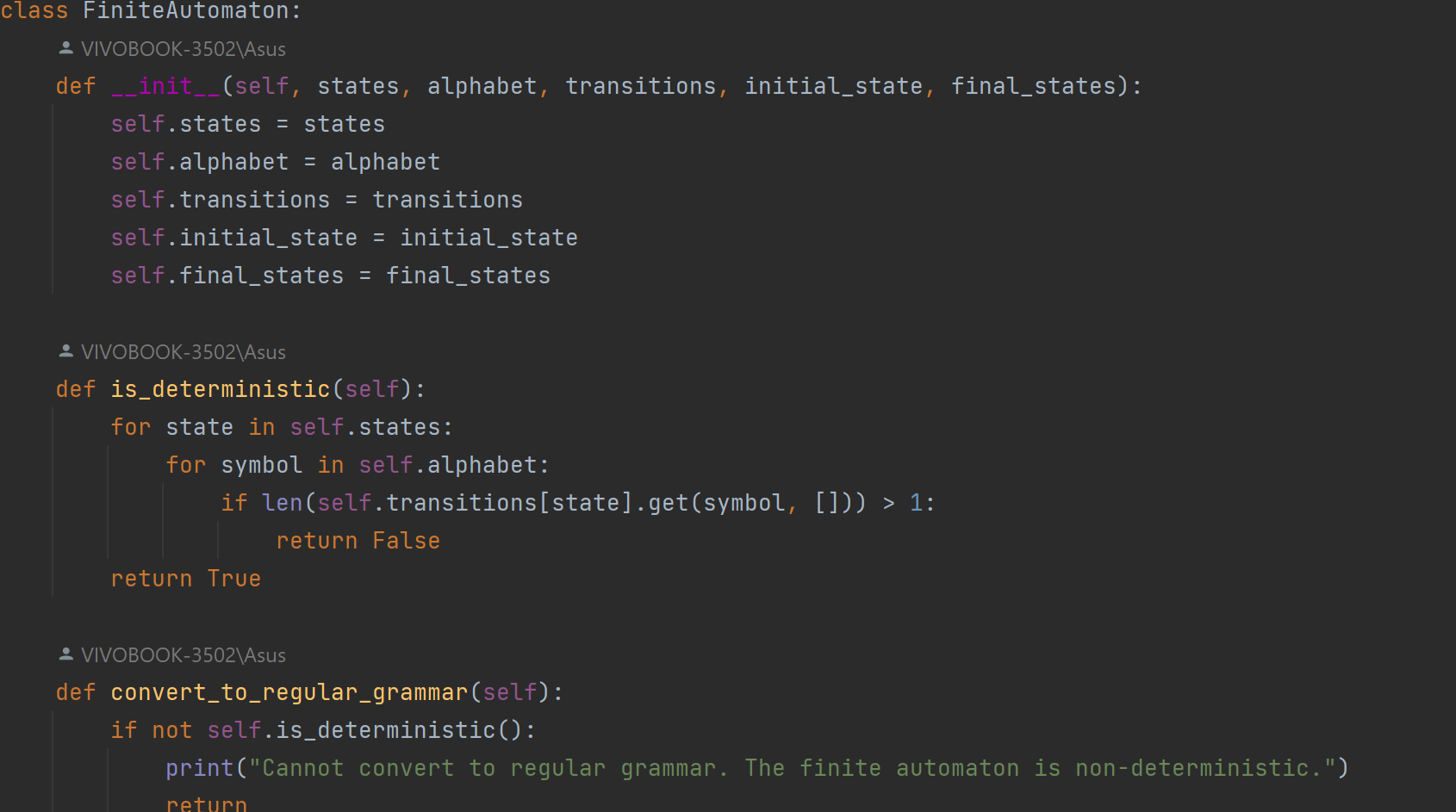
* States (Q): {q0, q1, q2, q3}
* Alphabet (∑): {a, b, c}
* Final States (F): {q3}
* Transitions (δ):
  + δ(q0, a) = q0, δ(q0, a) = q1, δ(q1, a) = q1
  + δ(q1, c) = q2, δ(q1, b) = q3
  + δ(q0, b) = q2, δ(q2, b) = q3

To determine if it's deterministic, we need to check if for every state and every symbol in the alphabet, there's at most one possible transition. If there's more than one possible transition for any state and symbol, it's non-deterministic.

Let's analyze the transitions:

* δ(q0, a) has two possible destinations (q0, q1) -> Non-deterministic.
* δ(q0, b) has one destination (q2)
* δ(q1, a) has one destination (q1)
* δ(q1, c) has one destination (q2)
* δ(q1, b) has one destination (q3)
* δ(q2, b) has one destination (q3)

Since there is at least one state with a transition for a symbol that leads to more than one state, the FA is non-deterministic.

****

To convert the non-deterministic finite automaton (NFA) to a deterministic finite automaton (DFA), we'll apply the subset construction algorithm. Once we have the equivalent DFA, we can then convert it into a regular grammar.

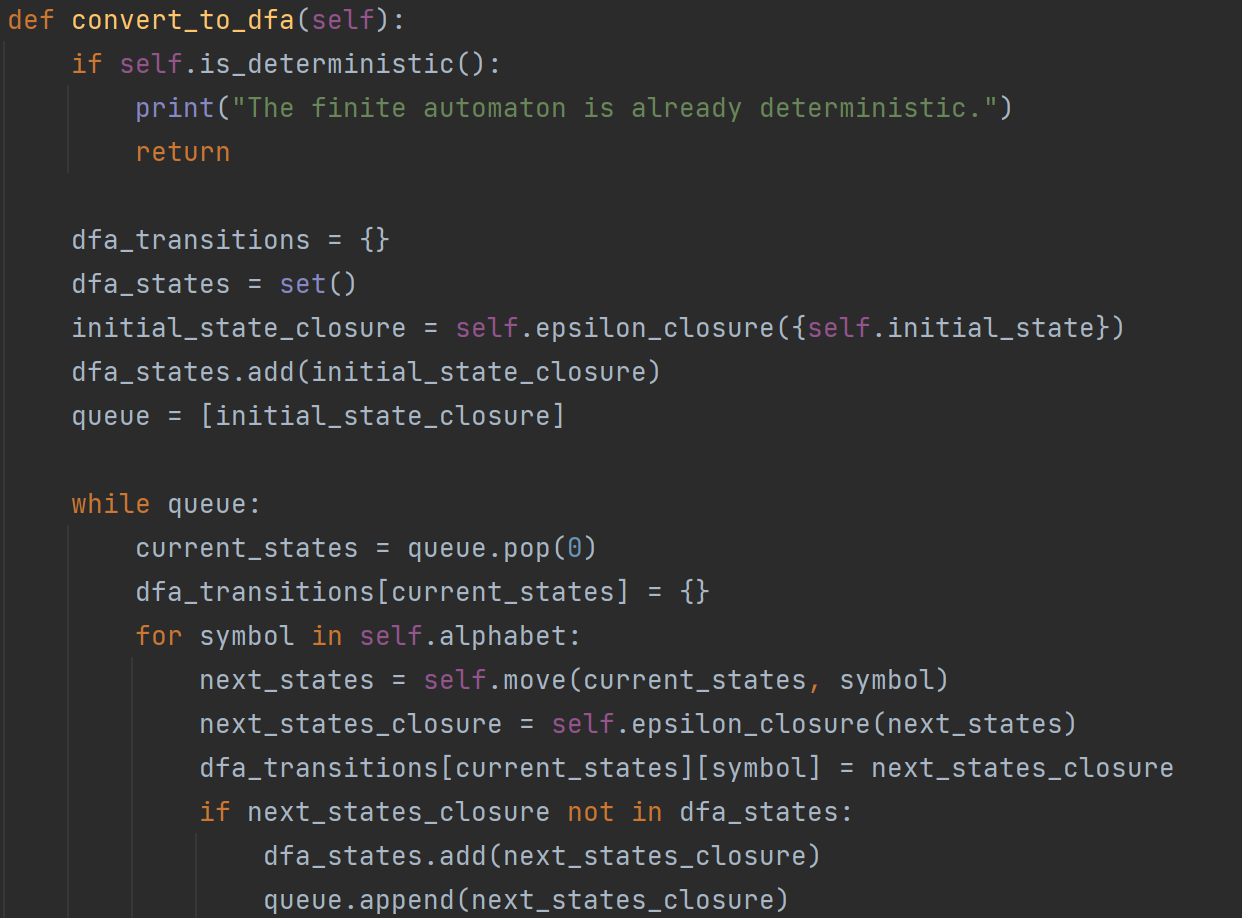
Let's go through the steps:

### Step 1: Subset Construction (Convert NFA to DFA)

* Start with the initial state of the NFA as the initial state of the DFA.
* For each state set in the DFA, determine the possible transitions for each symbol in the alphabet by following the transitions of the NFA.
* If a state set has transitions to new states, add those states to the DFA.
* Repeat the process until no new states are added.

### Step 2: Convert DFA to Regular Grammar

Once we have the DFA, we can convert it into a regular grammar. Each state in the DFA will correspond to a non-terminal symbol in the regular grammar, and the transitions will be converted into production rules.



**Conclusion:**

* finite\_automaton.py: This file contains the definition of the FiniteAutomaton class, which represents a finite automaton. It includes methods for determining if the automaton is deterministic, converting it to a regular grammar, and converting a non-deterministic automaton to a deterministic one.
* graph\_drawer.py: This file defines the GraphDrawer class, which is responsible for drawing the finite automaton graphically using the NetworkX library. It includes a static method draw that takes states and transitions as input and visualizes the automaton graphically.
* main.py: This file serves as the main program entry point. It creates an instance of the FiniteAutomaton class, determines if the automaton is deterministic or non-deterministic, converts it to a regular grammar if possible, converts a non-deterministic automaton to a deterministic one, and finally, draws the automaton graphically using the GraphDrawer class.