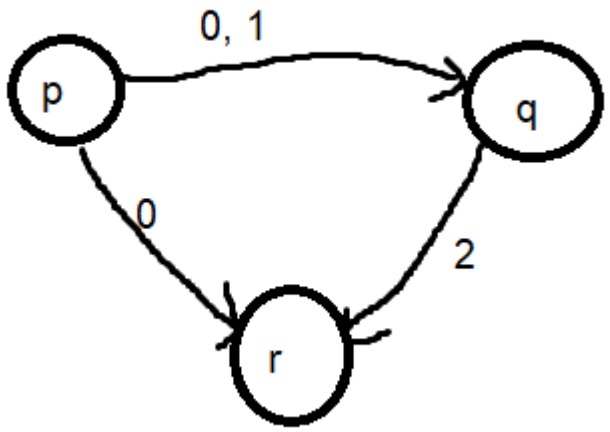
Rusu Horia-Radu
https://github.com/HoriaRaduRusu/Formal-Languages-And-Compiler-Design

Vertex:

Vertex is a data class used to represent a node in the graph described below. It has a label attribute, representing the identifier of the vertex.

GraphWithLabeledAdjacency:

This class describes an oriented graph with the property that each edge is labeled with at least one String. It has a Map attribute, which describes the graph in the following way: the keys are the starting vertices of the edges, and the values are Maps, holding as keys the labels of the edges that start from the vertex, and as values, a list of the vertices that can be reached from the starting vertex by going through edges labeled with a given label. For example, if we take the following graph:



the map would look like this:
p -> (0 -> [q, r], 1->[q])
q -> (2 -> [r])

It has the following methods:

- List<Vertex> getNeighborsThroughElement(Vertex starter, String element):
  Returns the vertices that neighbor the “starter” vertex through edges labeled with “element”.

- Map<Vertex, Map<String, List<Vertex>>> getAdjMap():
  Returns the adjacency map.

FiniteAutomaton:

This class describes a finite automaton described in a given file. It holds a set of strings for the states and one for the alphabet (since the order of elements in the states and alphabet don’t matter and they can’t repeat), a string representing the starting state, a set of strings representing the end states (with the same reasoning as above), and a GraphWithLabeledAdjacency for the transitions. When creating a FiniteAutomaton, if any state or alphabet element used in the file is not present in the set of states or alphabet elements, an error is thrown. It exposes the following methods:

- Set<String> getStates():
  Returns the set of states.

- Set<String> getAlphabet():
  Returns the set of alphabet elements.

- String getStartingState():
  Returns the starting state.

- Set<String> getEndStates():
  Returns the set of end states.

- String getTransitionsForPrinting():
  Returns the transitions graph in a form that can be printed.

- boolean isDFA():
  Returns true if the FiniteAutomaton represents a deterministic finite automaton, otherwise false.

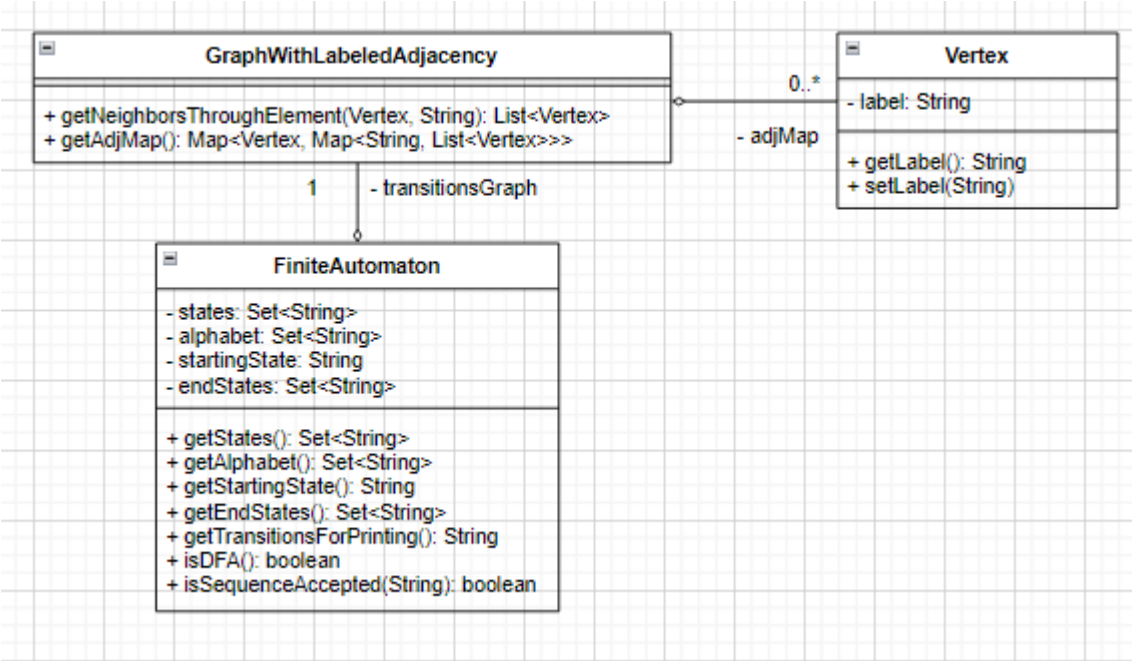
- boolean isSequenceAccepted(String sequence):
  Returns true if the finite automaton described accepts the given sequence, otherwise false.

The file that specifies the finite automaton should be written in the format described in the following way (in EBNF form, and with “\n” representing a new line):

nZDigit = “1” | … | “9”
digit = “0” | nZDigit
state = “s” nZDigit {digit}
alphabetElement = digit | “a” | … | “z” | “A” | … | “Z” | “_”
stateList = state {“ ” state}
alphabet = alphabetElement {“ ” alphabetElement}
transition = state “ ” alphabetElement “ ” stateList
transitionList = transition {“\n” transition}

file = stateList ”\n” alphabet “\n” state “\n” stateList “\n” transitionList

Class diagram:


GraphWithLabeledAdjacency
+ getNeighborsThroughElement(Vertex, String): List<Vertex>
+ getAdjMap(): Map<Vertex, Map<String, List<Vertex>>>
0..*
- adjMap
Vertex
- label: String
+ getLabel(): String
+ setLabel(String)
1
- transitionsGraph
FiniteAutomaton
- states: Set<String>
- alphabet: Set<String>
- startingState: String
- endStates: Set<String>
+ getStates(): Set<String>
+ getAlphabet(): Set<String>
+ getStartingState(): String
+ getEndStates(): Set<String>
+ getTransitionsForPrinting(): String
+ isDFA(): boolean
+ isSequenceAccepted(String): boolean