https://github.com/the-coding-cloud/FLCD/tree/main/Lab4 - GitHub repository

Lab 4 – Documentation

Purpose:

The goal for this lab was to implement an algorithm that checks if a sequence is accepted by a deterministic finite automaton.

Implementation:

In my code, I chose to implement a class named FA, with several private fields that store:

- The states (python list)
- The alphabet (python list)
- The transitions (python dictionary)
- The initial state (simple value)
- The final states (python list)
- If the FA is a DFA or not (boolean value)

The idea is that during the instantiation, the file given as parameter to the object constructor is read line by line and the fields of the object are populated when the ***readFA*** method is called. Mainly, the split of the elements on each line is done by spaces, with the exception of transitions. Transition functions are initially split by ";", then each of these token is then split by ",". The first two tokens will be the tuple given as a (composed) key in the dictionary, while the third one will be the value.

Then, there are several functions available for printing information about the finite automaton (***printStates, printAlphabet, printTransitions, printInitialState, printFinalState***).

The most important method, however, is ***checkSequenceAcceptance***. The algorithm starts by initializing the *currentState* variable with the initial state of the FA. Then, while the sequence is not empty, the *currentState* is updated accordingly to the available transition functions from the FA and the sequence is shortened, leaving only the rest of the unread sequence.

Since the transitions are stored in a dictionary, the key consists of a tuple of the form *(q, a)*, where *q* is the current state and *a* is the first character from the unread sequence, and the value is the list of results of the transition function (the next state). If there is no available transition for a certain tuple, the sequence is rejected and the function returns *False*. When we reach the end of the sequence, we check to see whether the currentState has now reached a final state and return *True*, otherwise return *False*.

When *FA.py* runs, a menu will be printed in the console, showing several possible commands and a sequence can be also given as input in order to be verified by the DFA implemented.

In addition, in the *ScannerIntegration* folder of Lab4, the *FA* class was used in the *Scanner* class in order to check if the identifiers and integer constants follow the specifications of the mini language.

File format:

miniLangCharacter = letter | digit | sign

letter = "A" | "B" | "C" | "D" | ... | "Z" | "a" | "b" | "c" | "d" | ... | "z"

digit = "0" | nonZeroDigit

nonZeroDigit = "1" | "2" | "3" | ... | "9"

sign = + | -

state = letter

transition = state “,” miniLangCharacter “,” state

line1 = { state “ “ }

line2 = { alphanumeric “ “}

line3 = { transition “;” }

line4 = state

line5 = { state “ “}

file = line1 \n line2 \n line3 \n line4 \n line5 \n

Example: ***FA.in***

p q r

a b

p,a,q;q,a,q;q,b,r;p,b,r

p

r

Comments on the file structure:

- *States in my files in particular are represented only as a letter*
- *A transition δ(p,a) = q will be represented in the file as p,a,q*
- *Multiple transitions are separated through “;”*
- *Line 1 contains all the possible states of the FA*
- *Line 2 contains all the elements from the alphabet*
- *Line 3 contains the transitions*
- *Line 4 contains the initial state*
- *Line 5 contains the final states*

Interface:

- ***__readFA***
  - **Input**: filename – string
  - **Output**: -
  - **Preconditions**: filename represents the name of a file having the structure mentioned above
  - **Postconditions**: all the elements of the FA will be loaded into the fields of the instantiated object
- ***printStates***
  - **Input**: -
  - **Output**: all the FA states will be printed in the console
- ***printAlphabet***
  - **Input**: -
  - **Output**: the FA alphabet will be printed in the console
- ***printInitialState***
  - **Input**: -
  - **Output**: the initial state of the FA will be printed in the console
- ***printFinalStates***
  - **Input**: -
  - **Output**: all the FA final states will be printed in the console
- ***printTransitions***
  - **Input**: -
  - **Output**: all the FA transition functions will be printed in the console
- ***checkSequenceAcceptance***
  - **Input**: sequence – string
  - **Output**: True/False
  - **Preconditions**: the sequence is a string, the FA is a DFA
  - **Postconditions**: the functions returns *True* if the sequence is accepted by the DFA and *False* otherwise; returns *None* if the FA is not a DFA

UML Diagram for FA: