Finite Automata

FLCD Lab 4

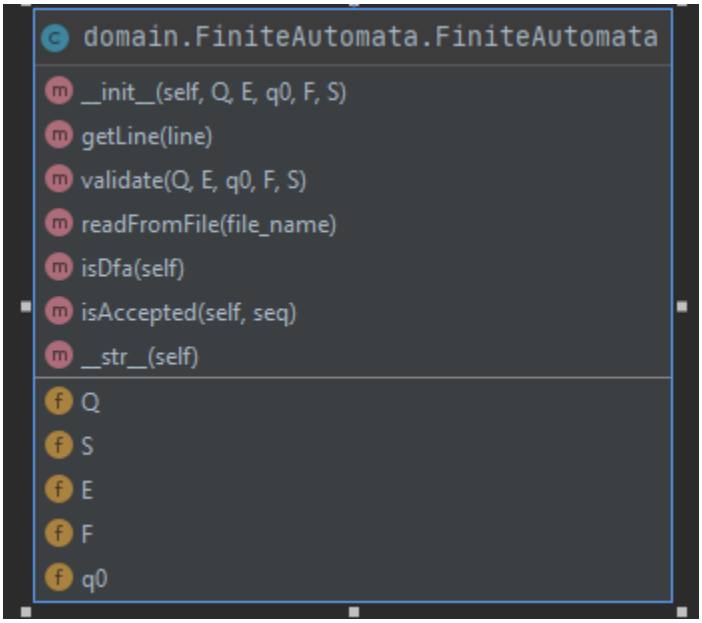
Popa Alex Ovidiu, 936/1

Github Link

The Finite Automaton is structured as a class with 5 fields: Q, E (Σ), q0, F, S, where each field is equivalent to its theoretical definition.

The transitions S are kept in a Python Dictionary, I.e. HashMap, where each pair (q, a) is mapped to a list of destination states, for example: (q, 1)->[p], meaning q goes to p with value 1.

FA class structure + methods



Checking that the FA is a DFA is done by going through all the dictionary keys, and looking if there's any list with a length greater than 1, as shown in the pseudocode below:

```
def isDfa(self):
 for k in self.S.keys():
  if len(self.S[k])>1:
   return False
 return True
```

Checking that a sequence is accepted by the FA is done by going through each symbol from the given sequence and checking that the respective point can be reached by following the FA transitions, as shown below:

```
def isAccepted(self, seq):
 if self.isDfa():
  crt = self.q0
  for symbol in seq:
   if (crt, symbol) in self.S.keys():
    crt = self.S[(crt,symbol)][0]
   else:
    return False
  return crt in self.F
 return False
```

FA.in example for NFA + output

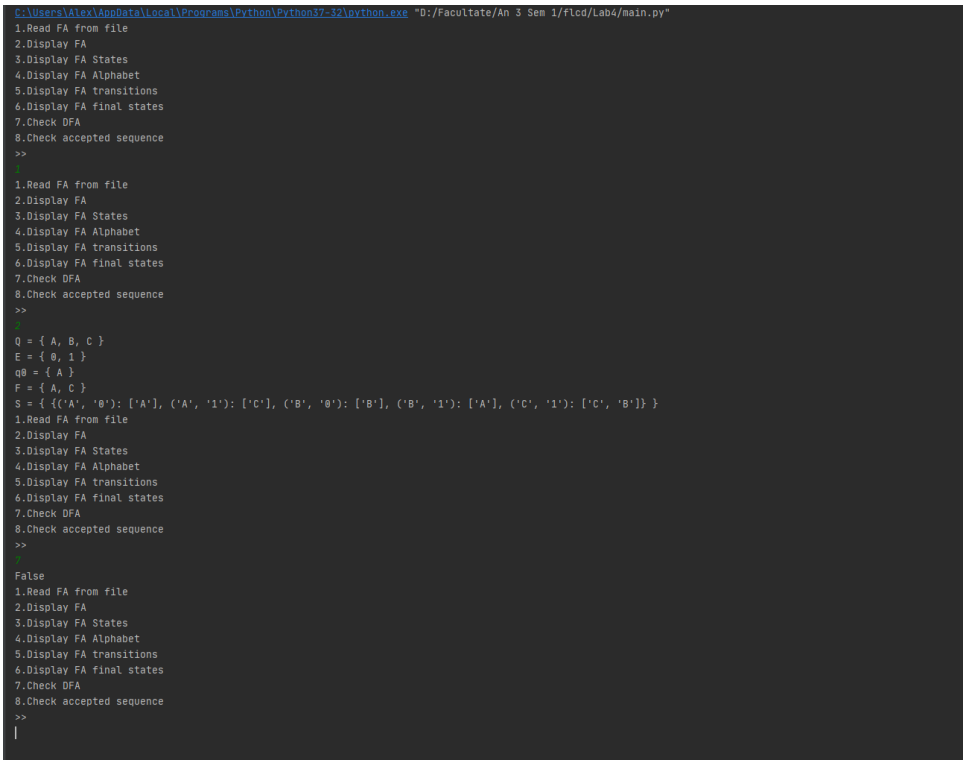
Q = A B C
E = 0 1
q0 = A
F = A C
S =
  (A,0) -> A
  (A,1) -> C

(B,0) -> B

(B,1) -> A

(C,1) -> C

(C,1) -> B



FA.in example for DFA + output

Q = A B C

E = 0 1

q0 = A

F = A C

S =

(A,0) -> A

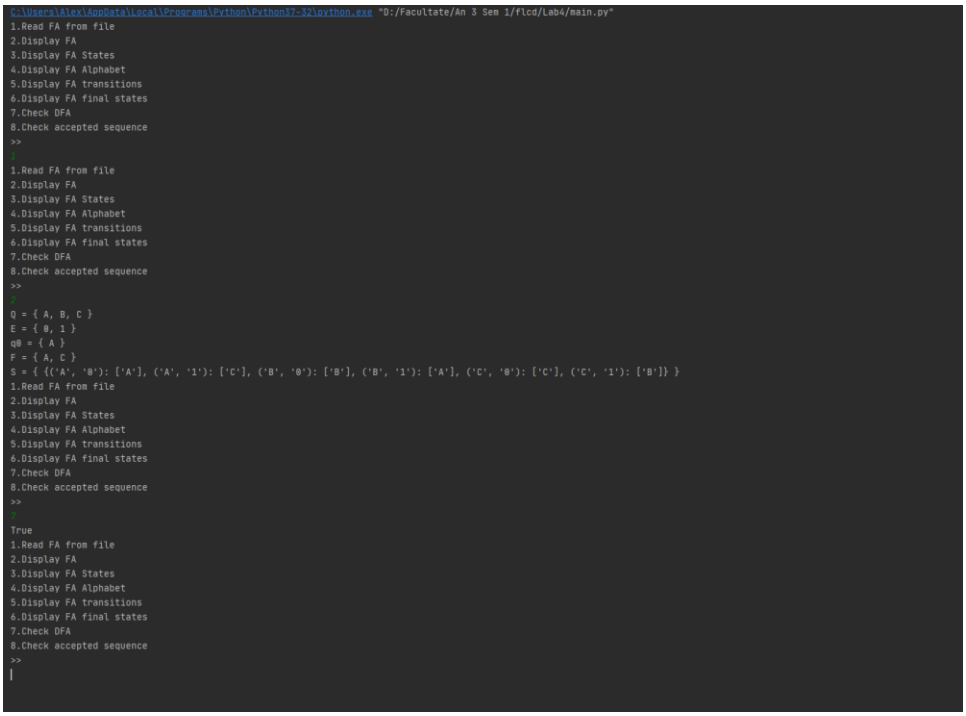
(A,1) -> C

(B,0) -> B

(B,1) -> A

(C,0) -> C

(C,1) -> B



Integration with Scanner lab

Link

The idea is that regex matching for constants and identifiers is replaced with checking that the FA accepts a given sequence, namely one which represents an identifier/a constant, like 12, abc, -100 a.s.o.

The FAs are kept in the fa-identifier.in and fa-constant.in files, and read on start up. They are equivalent to the identifier/constant definitions in EBNF from the second lab, see link here.