Finite Automaton

Github link:

https://github.com/CimpeanAndreea/FLCD/tree/master/Lab4

The definition of the finite automaton is given in a file that has the following structure:

FA.in

identifier::=letter(letter|digit|symbol)

character::=letter|digit|symbol

letter::= "a"|"b"|…|"z"|"A"|"B"|...|"Z"

digit::="0"|"1"|"2"|...|"9"

symbol::= "_"|"+"|"-"|"!"|"?"|":"|"."|"/"|"*"|"="|"&"|"# "|"< "|">"|"="

fa::=list_states "new_line" list_alphabet "new_line" initial_state "new_line" list_final_states "new_line" list_transitions

list_states::=identifier| identifier "," list_states

list_alphabet::=character|character "," list_alphabet

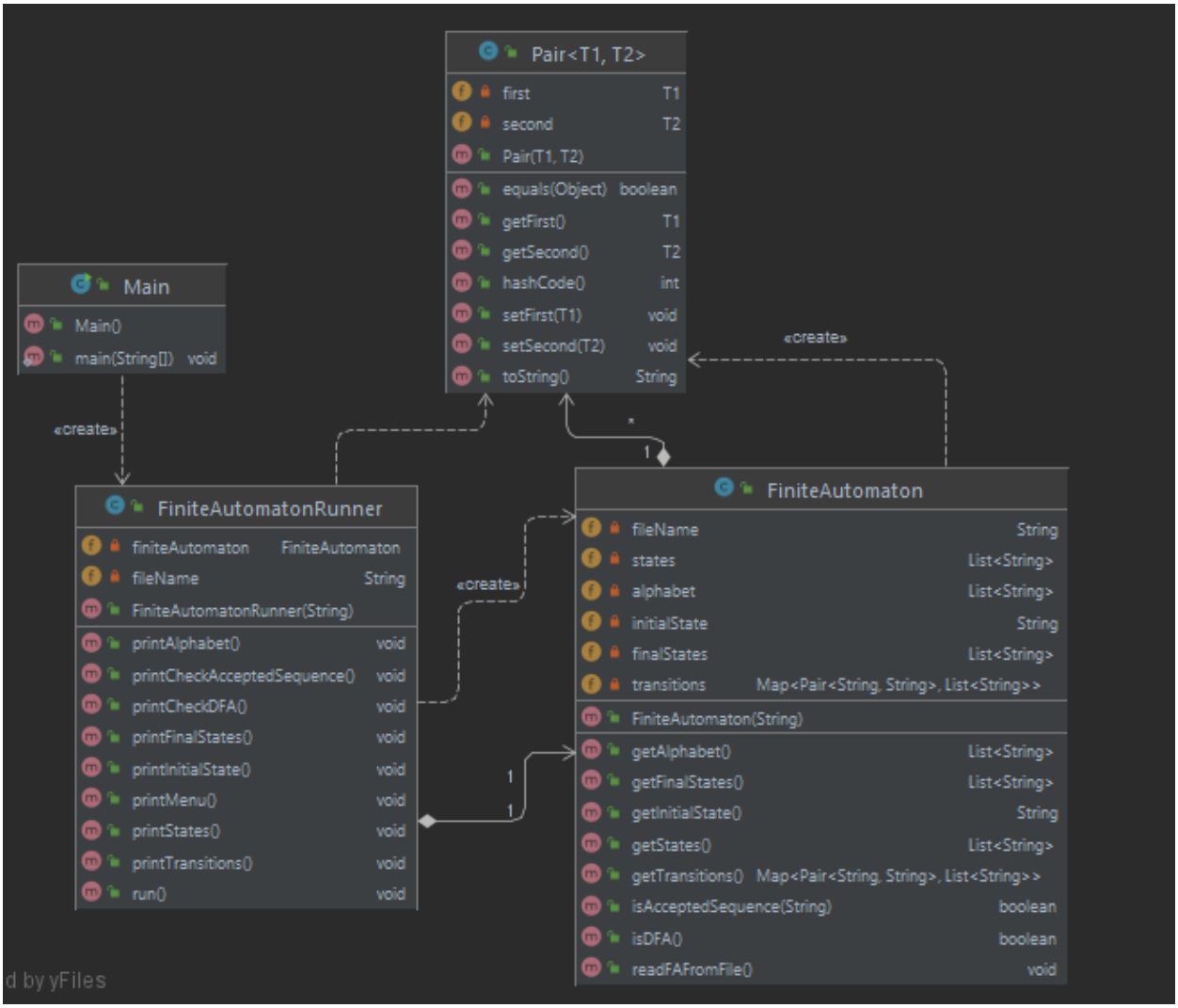
initial_state::=identifier

list_final_states::=identifier|identifier "," list_final_states

list_transitions::=transition|transition "new_line" list_transtitions

transition::=identifier "->" list_alphabet "|-" list_states

Implementation



The method chosen to solve the problem uses 2 important classes: **FiniteAutomaton** and **FiniteAutomatonRunner**.

FiniteAutomatonRunner is a class that prints a menu to test and to get the properties of a finite automaton given in a specified file.

FiniteAutomaton is the representation of a finite automaton

It has the following properties:

fileName: string (the file to read the automaton from)

states: String[]

alphabet: String[]

initialState: String

finalStates: String[]

transitions: Map((String, String), String[]) – transitions are represented as a map in which the key is a pair (state, alphabet_letter) and the value is a list of states

It has the following functions:

readFAFromFile()

description: read the finite automaton from the given file

pre: fileName is a valid file and respects the rules described at the beginning

post: the properties of the finite automaton were stored

isDFA()

description: check if the finite automaton is deterministic by going through each

key of the transitions map: if there is one corresponding list that has the size > 1, then the FA is not deterministic

post: return true/false if the FA is DFA/is not DFA

isAcceptedSequence(sequence: String)

description: check if a sequence is accepted by a DFA – beginning with the

initial state and first letter of the sequence, at each step try to find a pair

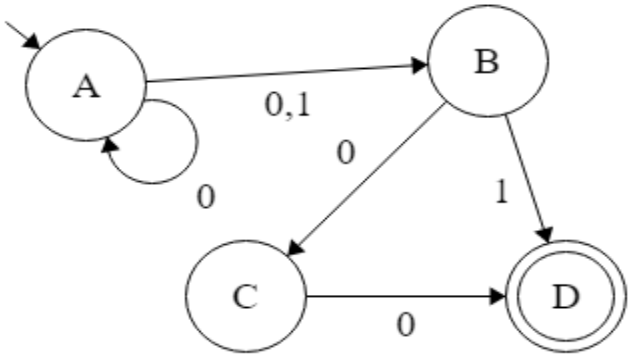
(currentState, currentLetterFromSequence) in the transitions map and if at the end every letter of the sequence was parsed and the final result state is among the final states then the sequence is accepted

post: return trun/false if the sequence is accepted/not accepted or throw an error if

the FA is not a DFA

Testing

1. **NFA:**



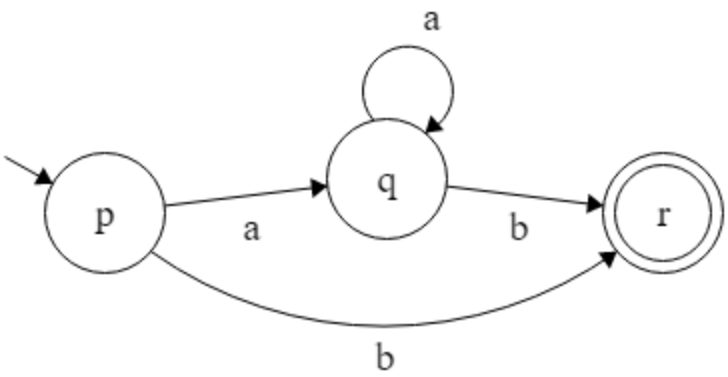
FA.in:

```
A, B, C, D
0,1
A
D
A -> 0, 1 |- B
A -> 0 |- A
B -> 0 |- C
C -> 0 |- D
B -> 1 |- D

A, B, C, D
0,1
A
D
A -> 0, 1 |- B
A -> 0 |- A
B -> 0 |- C
C -> 0 |- D
B -> 1 |- D
```

```
-------------MENU-------------
1. Print the set of states
2. Print the alphabet
3. Print the initial state
4. Print the the set of final states
5. Print transitions
6. Check if DFA
7. Check if a sequence is accepted
0. Exit
Your option:
6
IS DFA
false
```

2. **FA:**



FA.in:

```
p, q, r
a, b
p
r
p -> a |- q
p -> b |- r
q -> a |- q
q -> b |- r
```

```
-------------MENU-------------
1. Print the set of states
2. Print the alphabet
3. Print the initial state
4. Print the the set of final states
5. Print transitions
6. Check if DFA
7. Check if a sequence is accepted
0. Exit
Your option:
6
IS DFA
true
```

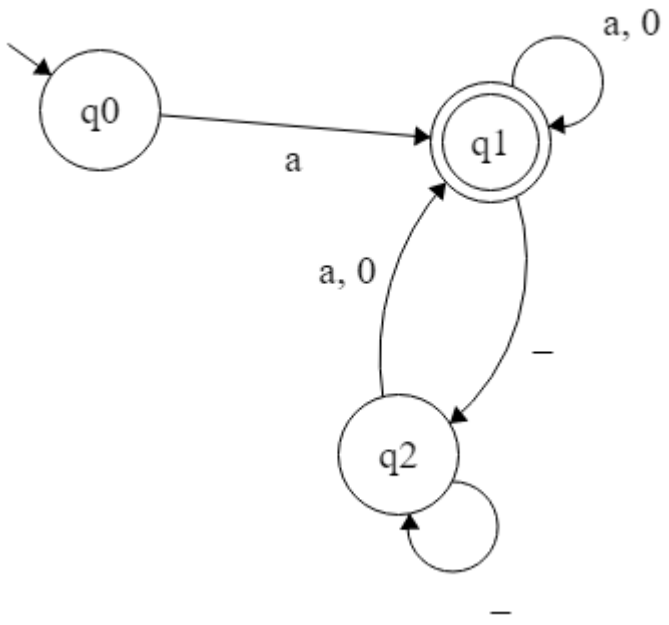
```
Your option:
7
CHECK ACCEPTED SEQUENCE
Give sequence:
aaab
true
```

```
Your option:
7
CHECK ACCEPTED SEQUENCE
Give sequence:
baa
false
```

Scanner Integration

Use FA implementation to detect identifiers and integer constants in scanner: to achieve this in the scanner only the functions that check that a given token is identifier/integer constant are changed to use the isAcceptedSequence(token) function of a specific finite automaton.

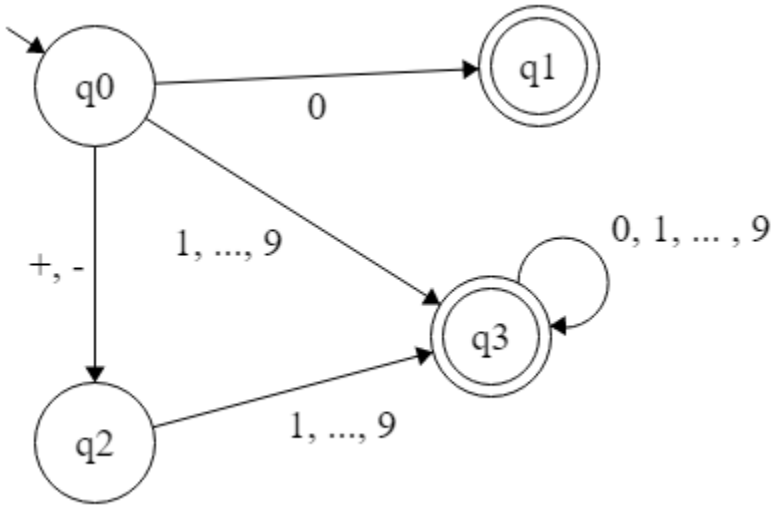
FA for identifiers: (for simplicity of the drawing consider “a” as a shorthand for a…zA…Z and “0” as a shorthand for 0…9)



FA_identifiers.in

```
q0, q1, q2
a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s,t,u,v,w,x,y,z,A,B,C,D,E,F,G,H,I,J
,K,L,M,N,O,P,Q,R,S,T,U,V,X,Y,Z,0,1,2,3,4,5,6,7,8,9,_
q0
q1
q0 ->
a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s,t,u,v,w,x,y,z,A,B,C,D,E,F,G,H,I,J
,K,L,M,N,O,P,Q,R,S,T,U,V,X,Y,Z  |- q1
q1 ->
a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s,t,u,v,w,x,y,z,A,B,C,D,E,F,G,H,I,J
,K,L,M,N,O,P,Q,R,S,T,U,V,X,Y,Z,0,1,2,3,4,5,6,7,8,9 |- q1
q1 -> _ |- q2
q2 -> _ |- q2
q2 ->
a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s,t,u,v,w,x,y,z,A,B,C,D,E,F,G,H,I,J
,K,L,M,N,O,P,Q,R,S,T,U,V,X,Y,Z,0,1,2,3,4,5,6,7,8,9 |- q1
```

FA for integer constants:



FA_integer_constant.in:

```
q0, q1, q2, q3
+, -, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
q0
q1, q3
q0 -> 0 |- q1
q0 -> +,- |- q2
q0 -> 1,2,3,4,5,6,7,8,9 |- q3
q2 -> 1,2,3,4,5,6,7,8,9 |- q3
q3 -> 0,1,2,3,4,5,6,7,8,9 |- q3
```