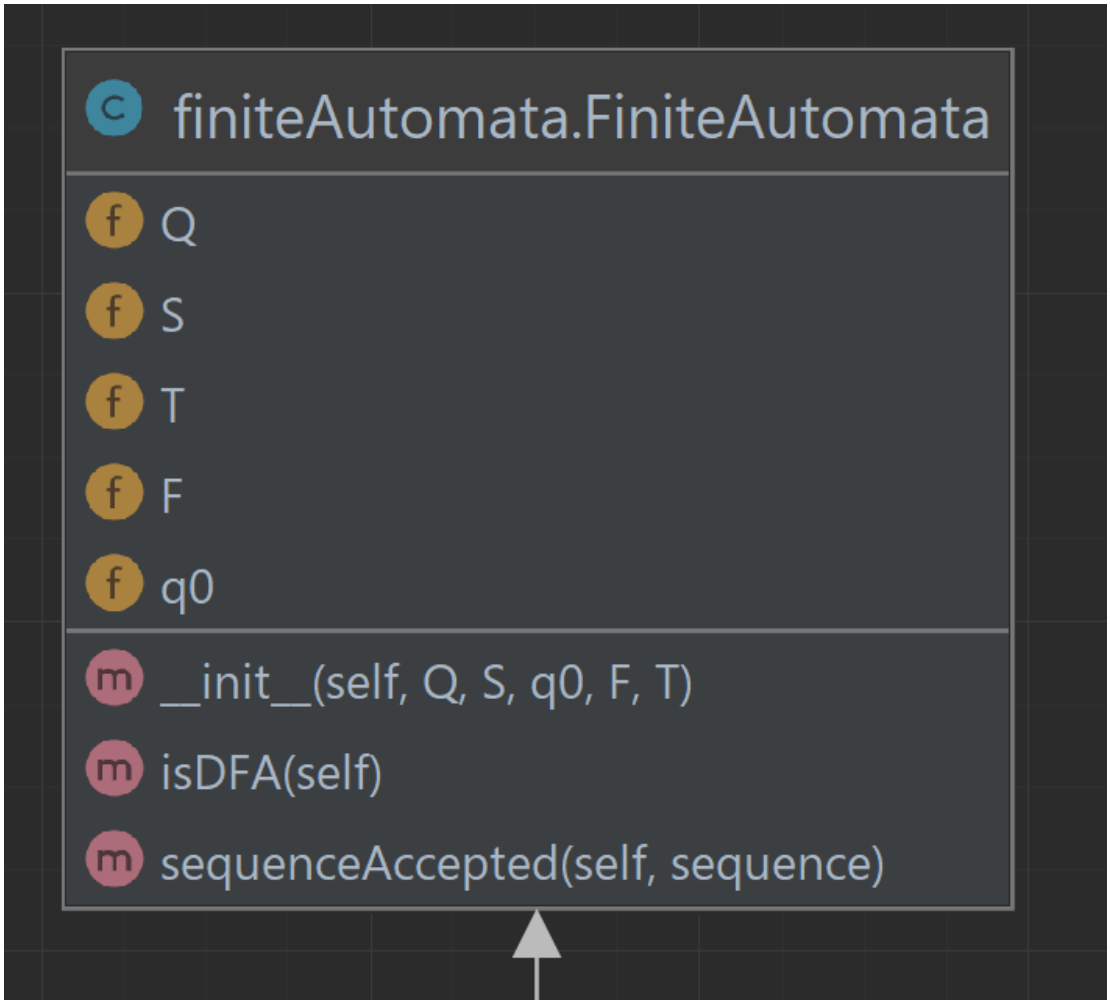
Lab 5 - FA documentation

Github links

FA integrated in the Scanner to detect tokens `<identifier>` and `<integer constant>`:
https://github.com/DiaconuAna/Formal-Languages-and-Compiler-Design/tree/main/Lab3 - Scanner

FA repository: https://github.com/DiaconuAna/Formal-Languages-and-Compiler-Design/tree/main/Lab4 - FA

Class diagram for FiniteAutomata



FiniteAutomata class attributes

- `Q`: finite set of states - python list
- `S`: finite alphabet - python list
- `q0`: initial state - python character
- `F`: set of final states - python list
- `T`: set of state transitions - dictionary where `key = (source, value), value = destination`, where `source` and `destination` must be states in `Q` and `value` must be a symbol in the alphabet `S`

Auxiliary functions

- `fileRead(fileName)`

  Reads the elements of a FA from a file.

  in: `fileName` - name of the file in which we store the FA

  out: the FA created reading the file or an error if any of the read data is invalid.

- `printMenu()`

  in: -

  out: -

  preconditions: -

  postconditions: prints the menu for displaying the FA elements

- `printStates(Q)`

  in: `Q` - set of states of a FA

  out: -

  preconditions: -

  postconditions: states of FA are printed on the screen

- `printAlphabet(S)`

  in: `S` - alphabet of FA

  out: -

  preconditions: -

  postconditions: FA alphabet is printed on the screen

- `printFinalStates(F)`

  in: `F` - set of final states of a FA

  out: -

  preconditions: -

  postconditions: final states of FA are printed on the screen

- `printTransitions(T)`

  in: `T` - set of states of a FA

  out: -

  preconditions: -

  postconditions: transitions of FA are printed on the screen

Methods

- `isDFA()`

  in: -

  out: `True` if the finite automaton is a determinist finite automaton, `False` otherwise

  Checks whether a FA is a DFA (deterministic finite automaton - there is at most a transition per symbol)

- `sequenceAccepted(sequence)`

We start from the initial state of the FA and successively check that each value from the sequence is a value that links the source to the destination, otherwise `False` is returned. Destination becomes the next starting point, until we reach the end where we check if the current state is among the final states of the automaton.

in: sequence to be accepted or not by the finite automaton

out: `True` if the sequence if the sequence is accepted by the finite automaton, `False` otherwise

preconditions: the finite automaton must be a deterministic finite automaton. The check is performed at the beginning of the function, resulting in `False` if the FA is not DFA.

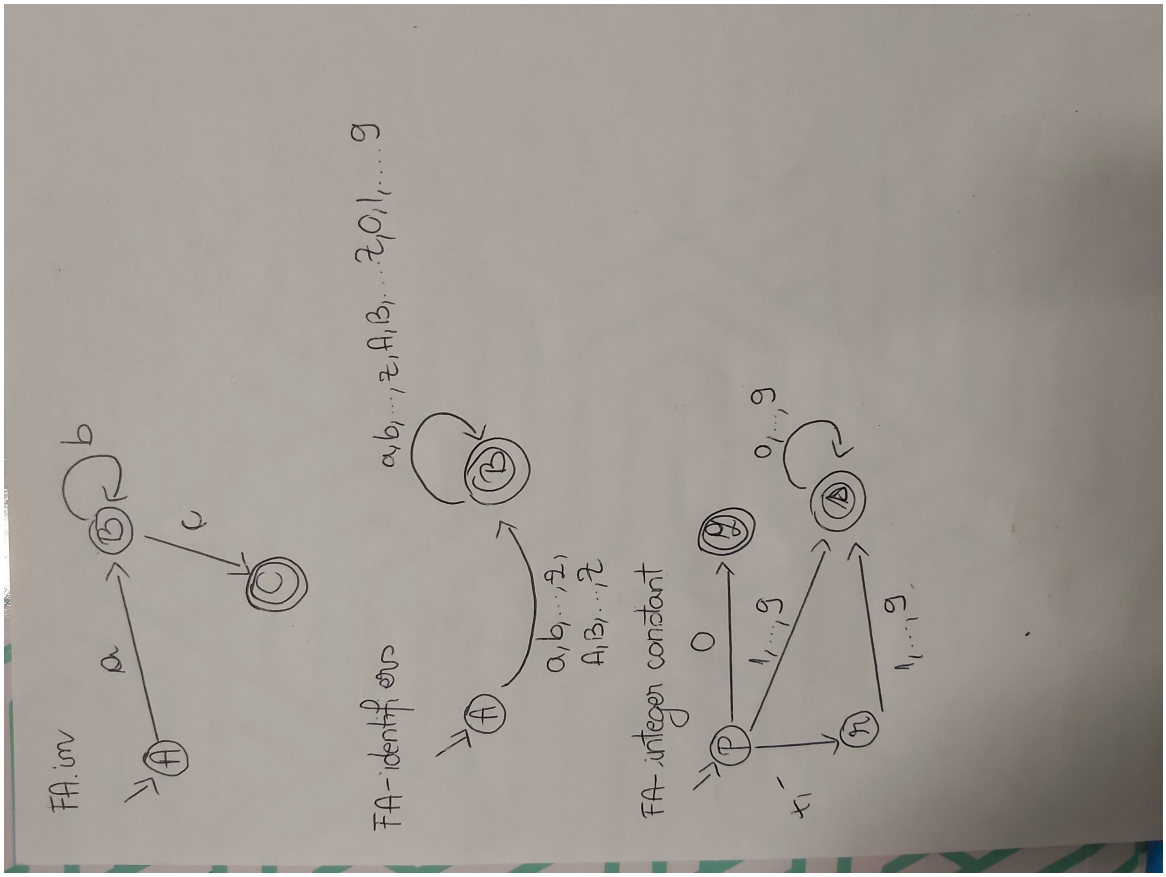
FA integrated in Scanner

2 more arguments are added to the `Scanner` constructor:

- `FAid`: Finite Automaton used to detect tokens `<identifier>`
- `FAint`: Finite Automaton used to detect tokens `<integer constant>`

Both are created using the `fileRead(fileName)` function described above and used in `classifyToken(token)` method by calling the FA `isSequenceAccepted(sequence)` method with the corresponding token.

▼ FA graphical representation



▼ FA in BNF

```
<newline> ::= `\n`.
<FA> ::= <states_set><newline><alphabet><newline><initial_state><newline><final_states><newline><transitions>.
```

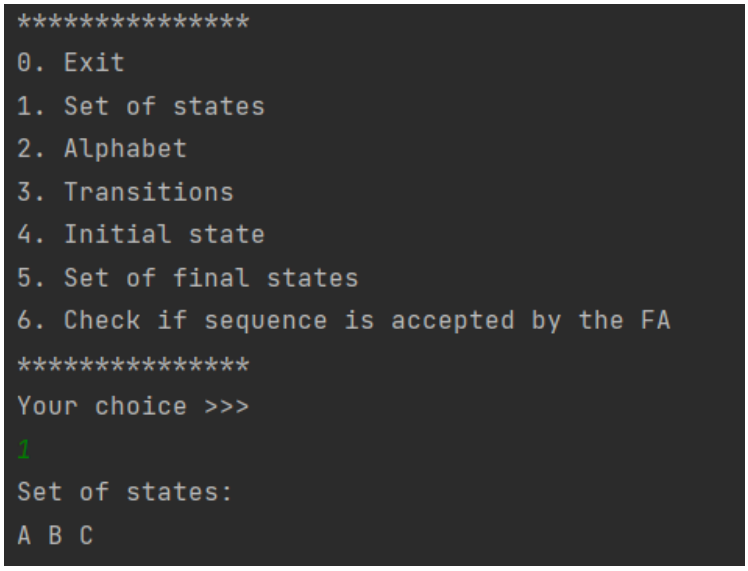
```
<state> ::= `A`|`B`|...|`Z`|`a`|`b`|...|`z`.
<states> ::= <state>|<state> ` ` <states>.
<states_set> ::= `Q = `<states_set>.

<alphabet_symbol> ::= `A`|`B`|...|`Z`|`a`|`b`|...|`z`|`0`|`1`|...|`9`.
<alphabet_set> ::= <alphabet_symbol>|<alphabet_symbol><alphabet_set>.
<alphabet> ::= `S = ` <alphabet_set>.

<initial_state> ::= `q0 = ` <state>.

<final_states> ::= `F = ` <states_set>.

<transition> ::= `(` <state> `,` <alphabet_symbol> `->` <state>.
<transition_set> ::= <transition><newline>|<transition><newline><transition_set>.
<transitions> ::= `T=` <newline> <transitions_set>.
```

Examples and files

▼ FA.in

```
Q = A B C
S = a b c
q0 = A
F = C
T =
(A,a) -> B
(B,b) -> B
(B,c) ->   C
```

▼ Set of states



▼ Alphabet

```
0. Exit
1. Set of states
2. Alphabet
3. Transitions
4. Initial state
5. Set of final states
6. Check if sequence is accepted by the FA
***************
Your choice >>>
2
FA alphabet:
a b c
***************
```

▼ Set of transitions

```
0. Exit
1. Set of states
2. Alphabet
3. Transitions
4. Initial state
5. Set of final states
6. Check if sequence is accepted by the FA
***************
Your choice >>>
3
Set of transitions:
(A, a) -> ['B']
(B, b) -> ['B']
(B, c) -> ['C']
***************
```

▼ Initial state

```
0. Exit
1. Set of states
2. Alphabet
3. Transitions
4. Initial state
5. Set of final states
6. Check if sequence is accepted by the FA
***************
Your choice >>>
4
Initial state: A
***************
```

▼ Set of final states

```
0. Exit
1. Set of states
2. Alphabet
3. Transitions
4. Initial state
5. Set of final states
6. Check if sequence is accepted by the FA
***************
Your choice >>>
5
Set of final states:
C
***************
```

▼ Check if sequence is accepted by the FA

```
0. Exit
1. Set of states
2. Alphabet
3. Transitions
4. Initial state
5. Set of final states
6. Check if sequence is accepted by the FA
***************
Your choice >>>
6
Sequence:
abc
Sequence accepted
***************
```

```
0. Exit
1. Set of states
2. Alphabet
3. Transitions
4. Initial state
5. Set of final states
6. Check if sequence is accepted by the FA
***************
Your choice >>>
6
Sequence:
ab
Sequence not accepted
***************
```

```
0. Exit
1. Set of states
2. Alphabet
3. Transitions
4. Initial state
5. Set of final states
6. Check if sequence is accepted by the FA
***************
Your choice >>>
6
Sequence:
fdjfdj
Sequence not accepted
***************
```