Lab 4-final Documentation

Jeler Andrei-Iulian

Group 934

Github: https://github.com/SummerRolls99/FLCD/tree/main/lab%205%20-%20finite%20automatan%20%2B%20scanner

Problem statement:

Write a program that:

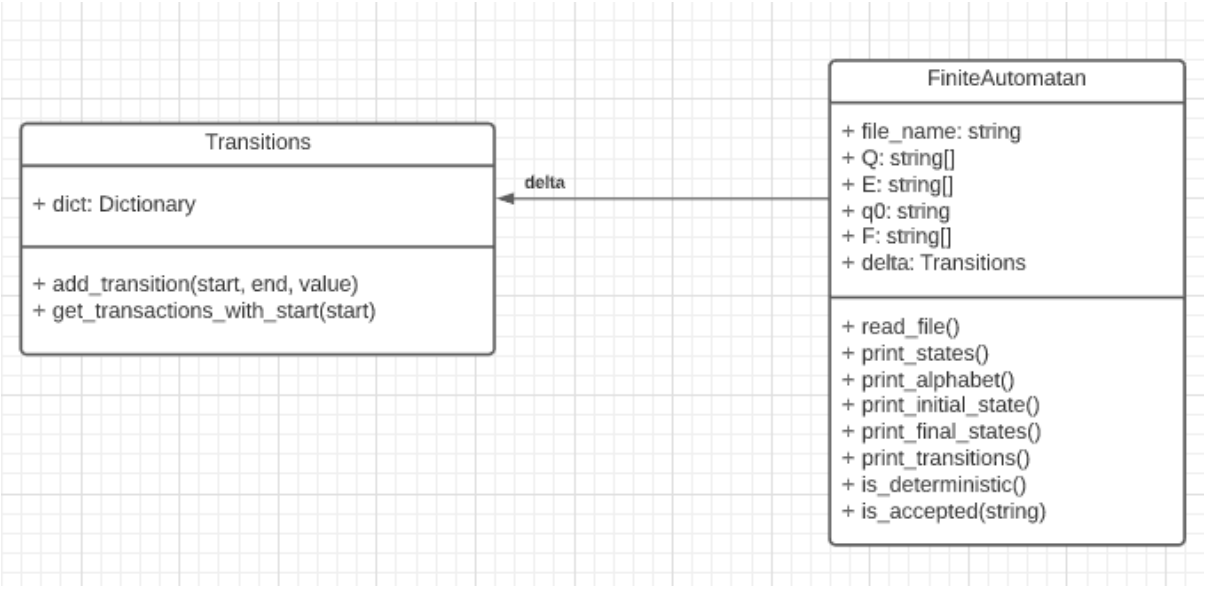
1. Reads the elements of a FA (from file)

2. Displays the elements of a finite automata, using a menu: the set of states, the alphabet, all the transitions, the set of final states.

3. For a DFA, verify if a sequence is accepted by the FA

Finite automatan class structure:

For the finite automatan I have stored the list of states as a set of string, initial state is a string, final states are a set of string, the alphabet is also a set of string. The transitions are stored in a class where a dictionary is used (the key is the starting state, the value is a tupple between the end state and value).



FA file structure:

- First line: list of all states
- Second line: alphabet
- Third line: Initial state
- Fourth line: Final states

- Remaining lines: transitions of form: initial_state end_state values

Is Deterministic:

Fot the is_deterministic method implementation we are taking each state from the list of states and check that there does not exist a pair of transition that has the same value.

Is Accepted:

The is_accepted method starts with a check to see if the FA is deterministic. After that we iterate the string char by char and start from the initial/starting state. The current state is checked for a transition that corresponds to the current char. If no transition is found, it means that the sequence is not accepted, and False can be returned. If a transition is found, the current state is changed to the coresponding state, and the process continues untill a transition cannot be found, or the string is empty, which means that the sequence is accepted, and True can be returned.

Tests:

```
def test():
    fa = FiniteAutomatan("fa.in")
    assert fa.is_deterministic() == True
    assert fa.is_accepted("aac") == True
    assert fa.is_accepted("aba") == False
    assert fa.is_accepted("aaaaaaaab") == True
    assert fa.is_accepted("aaaa") == False

    non_det = FiniteAutomatan("nonDet.in")
    assert non_det.is_deterministic() == False
```

DFA for identifier recognition:

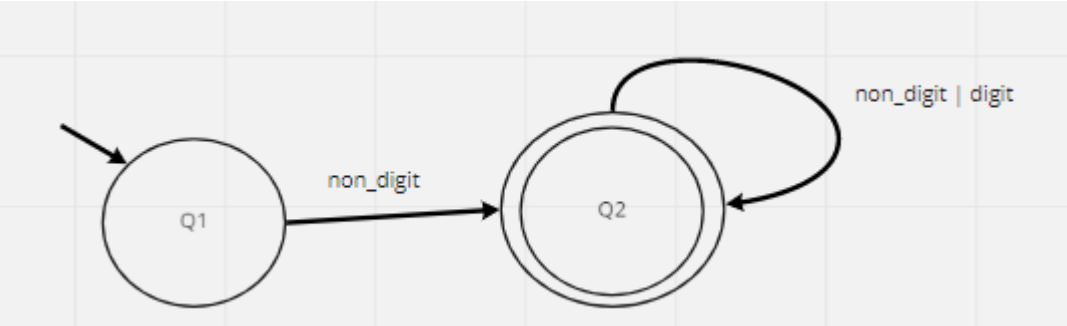
identifier = non_digit {non_digit | digit}

non_digit = "_" | letter

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

digit = "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"

```
Q1 Q2
q w e r t y u i o p a s d f g h j k l z x c v b n m Q W E R T Y U I O P A S D F G H J K L Z X C V B N M 0 1 2 3 4 5 6 7 8 9 _
Q1
Q2
Q1 Q2 q w e r t y u i o p a s d f g h j k l z x c v b n m Q W E R T Y U I O P A S D F G H J K L Z X C V B N M _
Q2 Q2 q w e r t y u i o p a s d f g h j k l z x c v b n m Q W E R T Y U I O P A S D F G H J K L Z X C V B N M _ 0 1 2 3 4 5 6 7 8 9
```



DFA for integers:

Integer = zero_digit | [sign] non_zero_digit { digit}

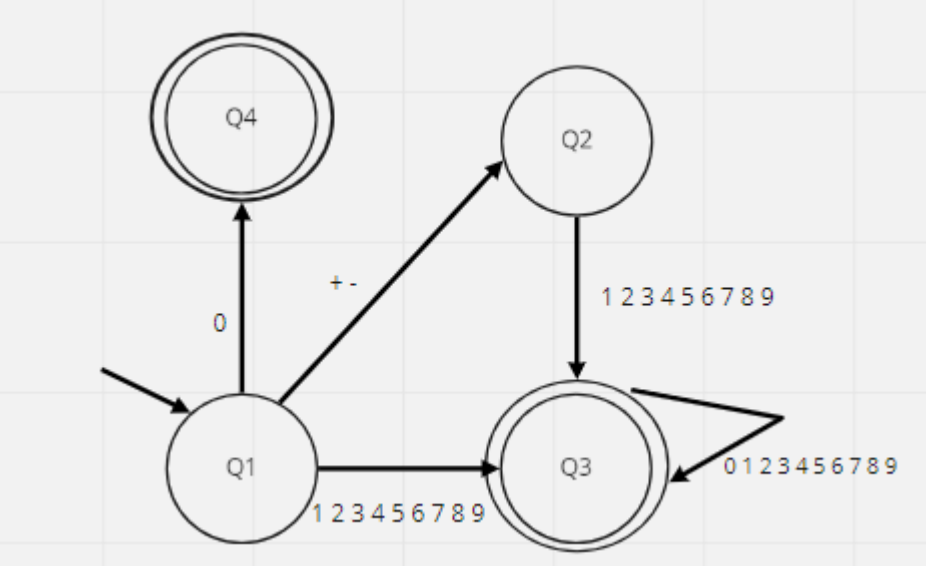
zero_digit = "0"

non_zero_digit = "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"

digit = zero_digit | non_zero_digit

sign = "+" | "-"

```
Q1 Q2 Q3 Q4
0 1 2 3 4 5 6 7 8 9 - +
Q1
Q3 Q4
Q1 Q2 + -
Q1 Q4 0
Q1 Q3 1 2 3 4 5 6 7 8 9
Q2 Q3 1 2 3 4 5 6 7 8 9
Q3 Q3 0 1 2 3 4 5 6 7 8 9
```



DFA for string constants:

string = " \" " {letter | digit | special_char} " \" "

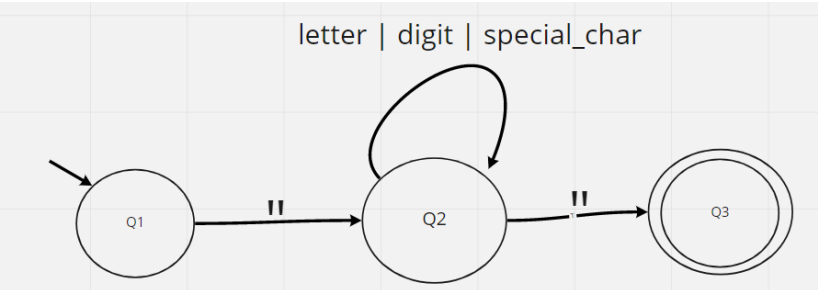
char = " ' " letter | digit | special_char " ' "

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

digit = "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"

special_char = "_" | "." | "," | ";" | ":" | " " | "?" | "!" | "@" | " " " |" /" | "(" | ")" | "|" | "-" | "+" | "=" | "{" | "}" | "*" | "[" | "]" | "$" | "%" | "^"

```
Q1 Q2 Q3
" 1 2 3 4 5 6 7 8 9 0 q w e r t y u i o p a s d f g h j k l z x c v b n m Q W E R T Y U I O P A S D F G H J K L Z X C V B N M , . ? ! _ ; : ? ! @ / ( ) | - + = { } * [ ] $ % ^
Q1
Q3
Q1 Q2 "
Q2 Q3 "
Q2 Q2 1 2 3 4 5 6 7 8 9 0 q w e r t y u i o p a s d f g h j k l z x c v b n m Q W E R T Y U I O P A S D F G H J K L Z X C V B N M , . _ ; : ? ! @ / ( ) | - + = { } * [ ] $ % ^
```



Use FA to detect tokens <identifier> and <integer constant> in the scanner program

Scanner implementation:

In order to integrate the finite automatan lab with the scanner one, i introduced a finite automatan for identifiers, integers and string constants, and modified the check. Now, they are not checked using regular expressions, they are checked using the is_accepted method defined in the Finite Automatan implementation.