Finite Automata Documentation

- link github fa: https://github.com/914-Mathe-Andrei/lftc/tree/main/lab4/fa
- link github compiler with fa: https://github.com/914-Mathe-Andrei/lftc/tree/main/lab4/compiler

I. Introduction

The entire implementation is written in python v3.11.

The program consists of a menu driven interface from where the user can interact with the finite automaton, by displayed the alphabet / states / transitions of the FA and by verify if a sequence is accepted. The FA is implemented as class named *FiniteAutomata* that receives a file that describes it. The input file is written in TOML with a fixed structure.

II. Details

main.py

The module represents the entry of the program and defines the menu and initializes the FA.

fa.py

The module contains the *FiniteAutomata* class and within the class there is a *State* structure representing a FA state that has a name and a list of transitions. A transitions is a key-value pair where the key is the symbol that the state can consume and the value is the next state.

▼ State

The class contains the name of the state and a dictionary of transitions. A transition is a key-value pair where the key is the symbol that the state is able to consume and the value is the next state if that symbol is going to be consumed.

▼ FiniteAutomata

The class contains the alphabet as a list of symbols (strings), the states as a dictionary, where the key is the name of the state and the value the corresponding *State* instance, the initial state as a *State* instance and, finally, the final states as a list of *State* instances. The constructor of the class requires as parameter a file where the FA is described in order to load its elements. The user can check if a sequence is accepted by the FA by calling the *check_acceptance()* class method that receives a sequence (actually a string) and return *True* if it is accepted, otherwise *False*.

III. FA input file

The FA input file is written in TOML and has a fixed structure. Here the FA's alphabet, states, inital / final states and transitions are described.

Format in EBNF

```
# ALPHABET
under_score = "_"

digit = "0" | ... | "9"

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

# STARTING RULE
start = alphabet_decl initial_state_decl final_states_decl states_decl

# GENERAL RULES
alphabet_decl = "alphabet" "=" symbols_array "\n"
initial_state_decl = "initial_state" "=" state "\n"
finial_states_decl = "final_states" "=" states_array "\n"
states_decl = { state_decl }

state_decl = "[[states]]" "\n" state_name_decl "\n" state_transitions_decl "\n"
state_name_decl = "name" "=" state
state_transitions_decl = "transitions" "=" transitions_array

transitions_array = "[" { transition "," } "]"
transition = "{" "symbol" "=" symbol "," "next_state" "=" state "}"

states_array = "[" { state "," } "]"
state = '"' ( underscore | letter | digit ) { underscore | letter | digit } '"'

symbols_array = "[" { symbol "," } "]"
symbol = '"' ( underscore | letter | digit ) { underscore | letter | digit } '"'
```

Example of input file

```toml
alphabet = [ "0", "1", "2", "3", "4", "5", "6", "7", "8", "9" ]
initial_state = "A"
final_states = [ "B", "C" ]

[[states]]
name = "A"
transitions = [
    { symbol = "0", next_state = "B" },
    { symbol = "1", next_state = "C" },
    { symbol = "2", next_state = "C" },
    { symbol = "3", next_state = "C" },
    { symbol = "4", next_state = "C" },
    { symbol = "5", next_state = "C" },
    { symbol = "6", next_state = "C" },
    { symbol = "7", next_state = "C" },
    { symbol = "8", next_state = "C" },
    { symbol = "9", next_state = "C" },
]

[[states]]
name = "B"
transitions = []

[[states]]
name = "C"
transitions = [
    { symbol = "0", next_state = "C" },
    { symbol = "1", next_state = "C" },
    { symbol = "2", next_state = "C" },
    { symbol = "3", next_state = "C" },
    { symbol = "4", next_state = "C" },
    { symbol = "5", next_state = "C" },
    { symbol = "6", next_state = "C" },
    { symbol = "7", next_state = "C" },
    { symbol = "8", next_state = "C" },
    { symbol = "9", next_state = "C" },
]
```