https://github.com/PascalauMihai/Lab4-FLCD

Finite Automata

A Finite Automata receives a file name as a parameter when it is declared. The given file contains all the data that it needs in the following format:

- First line: finite set of states (Q) – (letters + digits – example: q0, r, qf)
- Second line: finite alphabet (Σ) – (letters, digits, symbols)
- Third line: initial state (q0) – (q0 ∈ Q)
- Fourth line: set of final states (F) – (F ∈ Q)
- Fifth line – end of the file: list of transitions, each one written on one line, respecting the following format:
    - state1, alphabetComponent -> state2 – (example: q0, ‘a’ -> q1)

After the declaration, in the initialization, the FA reads the data from the given file and keeps it in the following way:

- The set of states is kept into a list: ([state1, state2, etc.])
- The alphabet is kept as a list: ([element1, element2, etc.])
- The initial state is kept as an element: (String state)
- The final states are added into a list: ([finalState1, finalState2, etc.])
- For the transitions, we create a dictionary that has as the key a tuple containing the pair of the first state and the element from the alphabet ((state1, alphabetComponent)). The value can be either a state or a list of states, in case there is a transition that starts from the same state, having the same element from the alphabet, but going into multiple states ([state1, state2, etc.] - nondeterministic)

After the initialization was performed, we can test if given sequences are accepted by the FA using the function validateSequence.

Representation:

- **FiniteAutomata:**
    - fileName: String
    - states: List
    - inputSymbols: List
    - initialState: String
    - finalStates: List
    - transitions: Dictionary

Interface:

- init(FA, givenFileName):
  - input: givenFileName – the file name of the Finite Automata elements
  - pre: true
  - desc: creates the Finite Automata keeping the given file name
  - post:
    - states – filled with states read from the file
    - inputSymbols – filled with the alphabet read from the file
    - initialState – initialized with the read initial state
    - finalStates – filled with the given final state/states
    - transitions – filled with the given transition functions
  - output: -

- readInputFromFile(FA):
  - input: -
  - pre: true
  - desc: reads the elements from the given file and fills the internal representation
  - post:
    - states – filled with states read from the file
    - inputSymbols – filled with the alphabet read from the file
    - initialState – initialized with the read initial state
    - finalStates – filled with the given final state/states
    - transitions – filled with the given transition functions
  - output: -

- readStates(FA, givenCurrentLine, givenFileReader):
  - input:
    - givenCurrentLine – the current read line from the file
    - givenFileReader – the fileReader initialized in readInputFromFile
  - pre: file reader exists
  - desc: reads the states line, reading all states and filling the states list
  - post:
    - states – filled with states read from the file
  - output: -

- readInputSymbols(FA, givenCurrentLine, givenFileReader):
  - input:
    - givenCurrentLine – the current read line from the file
    - givenFileReader – the fileReader initialized in readInputFromFile
  - pre: file reader exists
  - desc: reads the input symbols line, reading the alphabet and filling the inputSymbols list
  - post:
    - inputSymbols – filled with the alphabet read from the file
  - output: -

- readInitialState(FA, givenCurrentLine, givenFileReader):
  - input:
    - givenCurrentLine – the current read line from the file
    - givenFileReader – the fileReader initialized in readInputFromFile
  - pre: file reader exists
  - desc: reads the initial state line, and initializes the initialState variable
  - post:
    - initialState – initialized with the read initial state
  - output: -

- readFinalStates(FA, givenCurrentLine, givenFileReader):
  - input:
    - givenCurrentLine – the current read line from the file
    - givenFileReader – the fileReader initialized in readInputFromFile
  - pre: file reader exists
  - desc: reads the final states line, reading all states and filling the finalStates list
  - post:
    - finalStates – filled with the given final state/states
  - output: -

- readTransitions(FA, givenCurrentLine, givenFileReader):
  - input:
    - givenCurrentLine – the current read line from the file
    - givenFileReader – the fileReader initialized in readInputFromFile

- pre: file reader exists
- desc: reads all the remaining lines, getting from each line a transition function, then filling the transition dictionary
- post:
  - transitions – filled with the given transition functions
- output: -

- _validateSequence(FA, currentState, givenSequence):
  - input:
    - currentState – the current state in which the verification is at the point of entering the function
    - givenSequence – the given sequence that needs to be verified
  - pre: the FA was filled with the required data
  - desc: internal function that verifies if a given sequence is accepted by the FA
  - post: the sequence was verified, returning True if it is accepted and False otherwise
  - output:
    - True – if the sequence was accepted
    - False - otherwise

- validateSequence(FA, givenSequence):
  - input:
    - currentState – the current state in which the verification is at the point of entering the function
    - givenSequence – the given sequence that needs to be verified
  - pre: the FA was filled with the required data
  - desc: function that verifies if a given sequence is accepted by the FA
  - post: the sequence was verified, returning True if it is accepted, None if the FA is not deterministic or False otherwise
  - output:
    - True – if the sequence was accepted
    - None – the FA is not deterministic
    - False – otherwise
- print(FA):
  - input: -
  - pre: FA elements were read from the file
  - desc: prints the contents of the FA
  - post: the contents of the FA were printed
  - output: the contents of the FA were printed

FA.in in EBNF form:

program = states endLine inputSymbols endLine initialState endLine finalStates endLine transitions

endLine = "\n"

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

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

symbol = "+" | "-" | "*" | "/" | "% | "=" | "_" | "<" | ">" | "," | "." | "?" | "!" | "@" | "#"
| "$" | "^" | "&" | "(" | ")" | "[" | "]" | "{" | "}" | "\" | "|" | ";" | ":" | "'" | "'"

letterOrDigit = letter | digit

state = letter {letterOrDigit}

states = state | {state}

inputSymbol = letter | digit | symbol

inputSymbols = inputSymbol | inputSymbols {inputSymbol}

initialState = state

finalStates = states

transition = state ", " inputSymbol " -> " state endLine

transitions = transition | {transition}

Example:

p q r

a b

p

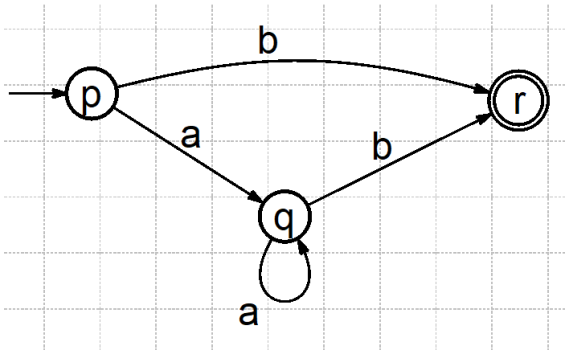
r

p, a -> q

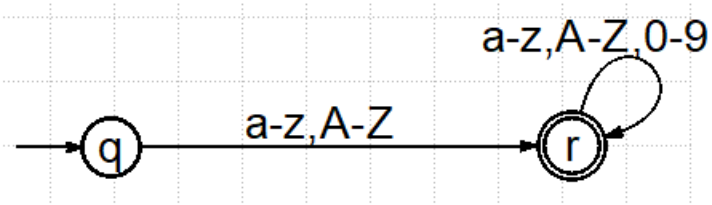
p, b -> r

q, a -> q

q, b -> r



Finite Automata Identifier:



Finite Automata Integer Constant: