Laboratory 4

Finite Automata

A Deterministic Finite Automaton consists of 5 tuples M=(Q, Σ, δ, q_0, F) where:

- Q = finite set of states (|Q|< ∞)
- Σ = finite alphabet (|Σ|< ∞)
- δ = transition function: δ:Q × Σ → P(Q)
- q_0 = initial state ($q_0 \in Q$)
- F ⊆ Q = set of final states

Class Implementation

Fields:

- states: List of strings – holds the list of all the states (Q)
- alphabet: List of strings – holds the list of input symbols (Σ)
- transitionFunctions: Dictionary that maps a pair of 2 two strings, the current state and the current input symbol, to a string that represents the next state
- initialState: String – the initial state (q_0)
- finalStates: List of string – holds the list of final states (F)
- *validInput: Boolean – initially is true, will be set to false if the input data from "FA.in" isn't valid (e.g. initial state doesn't belong to the list of all the states, transition function contains a non-existent input symbol)

Methods:

isValidState(state: String) – if the argument belongs to the list of all the states, return true, otherwhile returns false

readFromFile() – reads each line from the "FA.in" file; for each line, it checks if a certain keyword is found (e.g. "all_states: ", "initial_state"), then removes that part of the line, considering the remainder a list of states/symbols/functions delimited by ',' or by ';' in case of transition functions. The string is split and the elements are added to their respective list/dictionary.

displayMenu() – shows the possible commands with the corresponding keys

formatTransitions() – prints the set of transition functions in a pretty format

sequenceChecker(sequence: String) – checks if a given sequence is accepted by the FA; for each character of the sequence, it checks if there is any transition function that satisfies the current state and the current input symbol being the current character from the sequence. If the condition is satisfied, the next state from the function is taken, and we move on to the next character from the sequence. If we check all the transition functions and none are compatible with our sequence character, we assume that the sequence is invalid and it returns false. If the whole sequence has been

iterated through and the current state belongs to the final state list, the sequence is valid and it returns true, otherwise vice-versa.

checkSequence() – the user is asked to enter a sequence to be validated; *checkSequence()* is called for that sequence and the result is printed.

start() – if the read input is valid, the user is asked for the key corresponding to one of the six commands. When the user wants to stop the application, they should type “end”

BNF format of FA.in input file

<FA.in> ::= <states><inputSymbols><initialState><finalState><transitionFunctions>

<states> ::= ”states: “<listOfStates>

<listOfStates> ::= <state>|<state>”,”<listOfStates>

<state> ::= <letter>

<inputSymbols> ::= “input_symbols: “<listOfSymbols>

<listOfSymbols> ::= <symbol>|<symbol>”,”<listOfSymbols>

<symbol> ::= <character>

<initialState> ::= “initial_state: ”<state>

<finalStates> ::= “final_states: “<listOfStates>

<transitionFunctions> ::= “transition_function: “<listOfTransitions>

<listOfTransitions> ::= <transition>|<transition>”;”<listOfTransitions>

<transition> ::= “(“<state>”,”<symbol>”,”<state>”)”