Formal Languages and Compiler Design – Lab 4: Finite Automaton Implementation

Link to GitHub Repository: https://github.com/912-Crismariu-Ioana/FLCD/tree/main/Lab4

In order to integrate a Finite Automaton into the previous laboratory, I defined the class called FiniteAutomaton having the following fields:

- **alphabet**: a finite set of input symbols used to construct the language accepted by the automaton
- **initialState**: the initial state of the automaton
- states: finite set of possible states of the automaton
- **finiteStates**: set of final states
- **transitions**: set of transitions from one state to another, labeled with characters from the alphabet

These fields are populated through an input file provided in the constructor of the automaton.

The input file must obey the following structure, given in EBNF:

input_file = definition_block{"\n"definition_block}

definition_block = alphabet_block|states_block|transitions_block

alphabet_block = "@alphabet\n"alphabet_line{"\n"alphabet_line}

alphabet_line = input_symbol{ input_symbol}

input_symbol = letter|digit|"_"|"$"

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

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

state = letter ["initial"|"final"]

states_block = "@states\n"state{"\n"state}

transition_block = "@transitions\n"transition{"\n"transition}

transition = state input_symbol state

In order to represent the transitions, a utility class called Transition is used. The class has as fields the current state, an input symbol and the set of possible next states that can be reached from the current state with the given input**. In order to check whether our Finite Automaton is deterministic, it is enough to iteratively go through the set of transitions and check, for each transition, whether the set of possible next states contains more than one element. If it does, we know that our Finite Automaton is not deterministic.**

In order to check whether a sequence is accepted by the automaton, the class provides a method called **isSequenceAccepted** that takes a given input sequence, iterates over it character by character and tries to reach a final state from the initial state by using, from the set of transitions, the one that matches our current state and the character we’re iterating over. Upon reaching a final state, we know our sequence is accepted. If no transition is found, we stop and conclude that the sequence is not accepted.