Formal Languages and Compiler Design
Fourth laboratory

Finite Automata

Link to GIT repository: https://github.com/albcristi/formal-languages-and-compiler-design

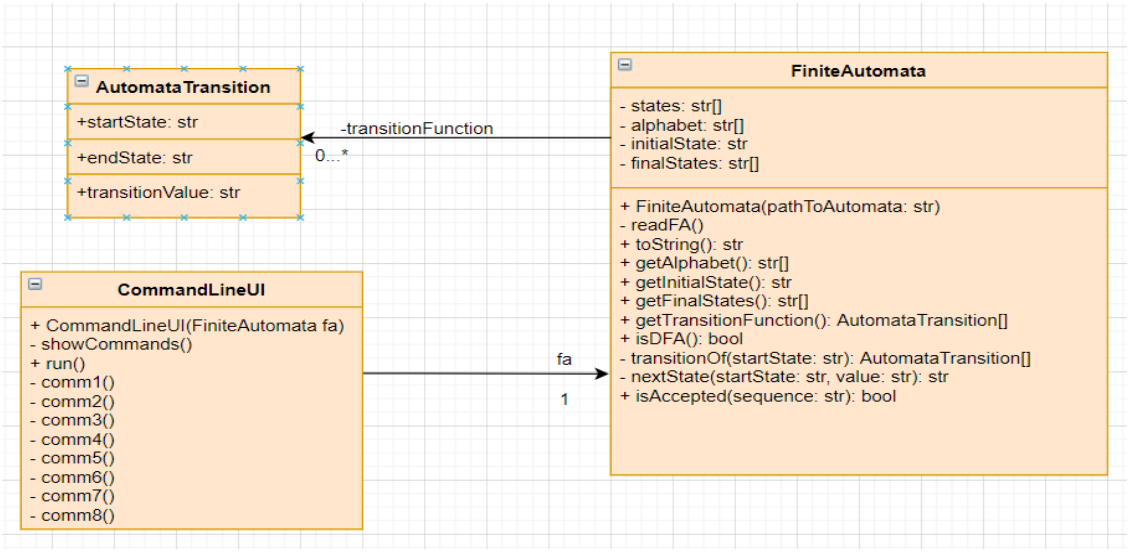
Implemented classes:

o AutomataTransition – this class stores the data of one transition from a transition function of a FA. It stores the starting state, ending state and the value of the transition

o FiniteAutomata – this class represents a FA, and it is formed by reading from a text file fa.in data that will form the FA, namely the set of states, alphabet, initial state, list of final states and list of transitions.

o CommandLineUI – represents the UI for the program, presenting to the user a set of possible commands and by the input of a command number, the program will execute the command on a predefined FA from fa.in. The user is able by command no.6 to change the path to the fa.in file, in order to take a self defined fa.in file.

Class diagram for the program:



- this is not integrated with the Scanner program, a diagram integrated with the scanner program will be presented in another part of the documentation

Alb Dan-Cristian
931

About the FA.in file:

1. File format:

1.1 Description

- the first line represents the list of states of the FA, separated by “,”
- the second line represent the alphabet of the FA, separated by “,”
- the third line will contain the initial state of the FA
- the fourth line will contain, separated by “,”, the list of final states of the FA
- the following line will contain the transition function, each row will be of form: state, alphabet element, state

1.2 EBFN describing the file format:

fa.in ::= stateLst alphabet initialState stateLst transitionLst

stateLst ::= state [{“,” state}] “\n”

alphabet ::= alphabetLetter [{“,” alphabetLetter}] “\n”

initialState ::= state “\n”

transitionLst ::= transition [{“\n” transition}]

transition ::= state “,” state “,” alphabetLetter

state ::= word

alphabetLetter ::=character

word ::= character{character}

character ::= digit | lowercase

digit ::= “0” | “1” | … | “9”

lowercase ::= “a” | “b” | … | “z”

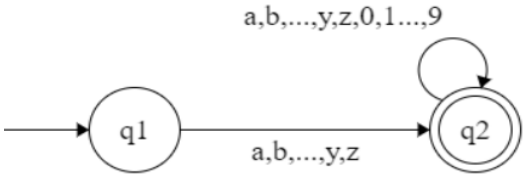
FAs and recognition of identifiers and integer constants:

It is known the fact that each language has a specific set of rules when it comes to defining identifiers and it is also easy to model some rules for a well formed integer constant, in this way we can use an FA to model identifiers and integer constant. We check if the FA accepts a sequence, if so, the sequence will be classified as an identifier or integer constant.

1. FA for idenrifiers

    1.1. Rule: identifier::= lowercase (digit | lowercase)*

    1.2. Finite Automata – graphical representation



2. FA for integer constants

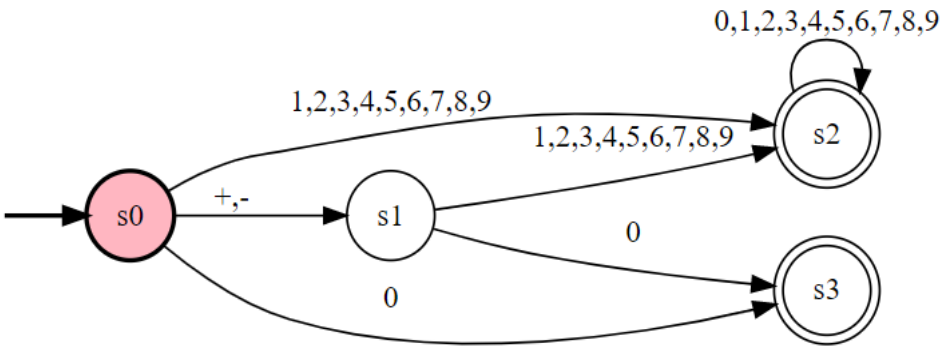
    2.1 Rule: integerct ::= [“+” | “-”] num

    num::= “0” | nonzero{digit}

    digit::= “0” | nonzero

    nonzero::= “1” | “2” | … | “9”

    2.2 FA – graphical representation:

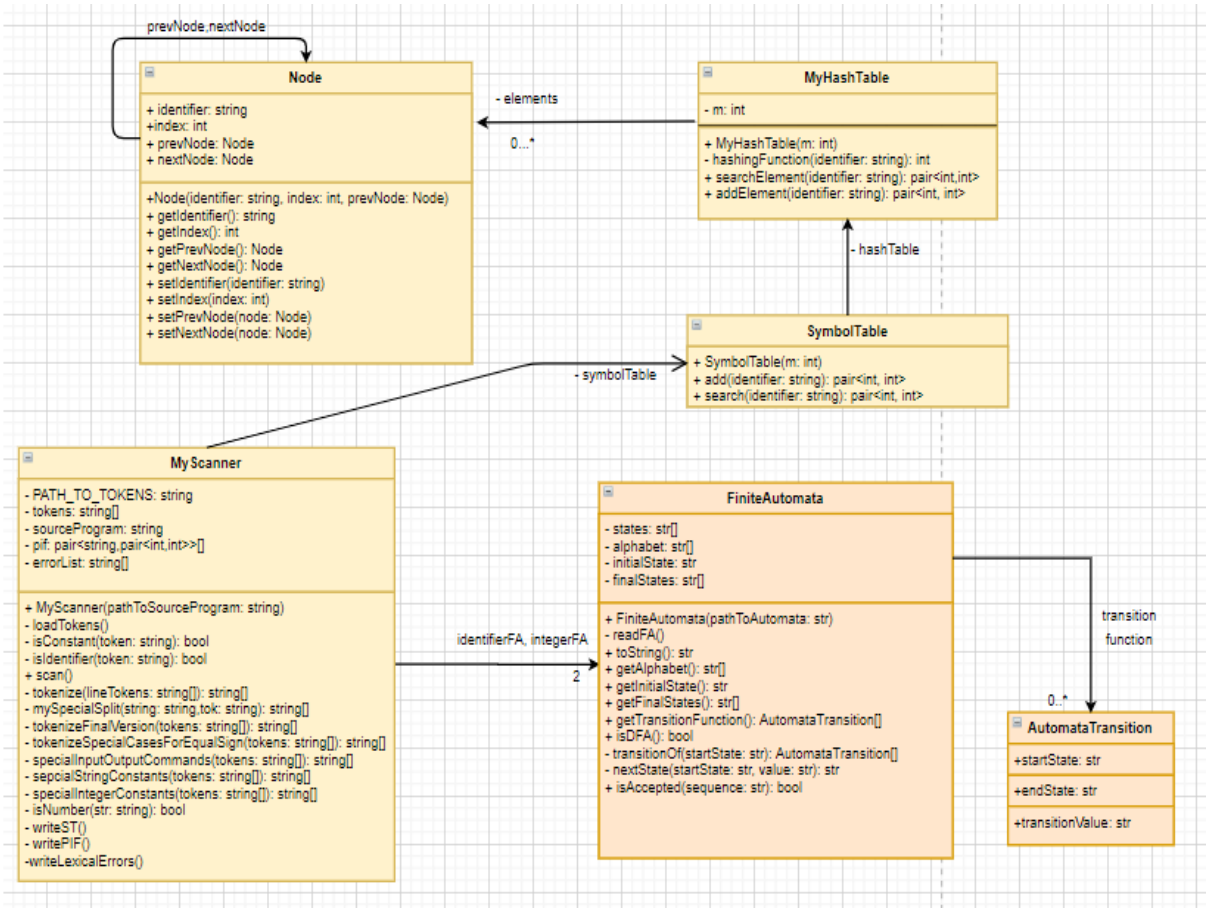


FA and integration with the Scanner Program:

Alb Dan-Cristian
931

Given the following FAs we can decide whether a token is an identifier or a integer constant by verifying if the token is accepted by one of the above FAs, then we can classify that token as an identifier or an integer constant.

Class Diagram



* open draw.io and use the flcd.drawio file to open the above diagram