ECOTE - preliminary project

Semester: 21L

Author: Wiktor Bara

Subject: NFA construction using Thompson algorithm

I. General overview and assumptions

The task is to design a program reading regular expression, then constructing Nondeterministic Finite Automata, and checking if input strings are generated by this expression using already created automata. The program should take two files as an input, first one with one regular expression and second one with input strings, and return one file as an output that contains information whether given strings could be generated by given regular expression.
I assumed that the alphabet for the regular expression contains all numbers and letters (both small and uppercase). The end of line ('\n') will be treated like end of the expression, and any other characters, that are not contained in regular expression definition, will be treated as an error.

II. Functional requirements

The general purpose for NFA constructor is to create automata and check whether the strings that are given can be created by this automata. This means that the class NFA is needed, in which based on the regular expression the automata is stored to be used multiple times. This data structure is further described in *Data structures* section.

The NFA constructor evaluates each character from the regular expression file, changing its behavior whenever it encounters special symbol. The special symbols of the NFA constructor are:

Symbol	Meaning	Priority
(	left parenthesis that opens block with greater calculation priority	-
)	right parenthesis that closes the block with greater calculation priority	-
?	zero or one characters	5
+	one or more characters	4
*****	zero or more characters	3
.	concatenation	2
|	alteration	1

The symbols have their priorities that force to generate NFA in the appropriate way and determines the order of the characters.
The NFA creator can also output the errors and warnings, when incorrect, unexpected or inadvisable input is encountered. The program counts the lines in the input file to make the errors more useful and helpful for the user. The errors and warnings will be described in the *Input/output description* section.

III. Implementation

General architecture

The program consists of five main modules:

- Main Module – responsible for the Command Line Interface and File I/O
- NFA Creator Module – responsible for creation of the automata
- String Comparison Module – responsible for checking whether given strings can be created by automata
- Error Module – responsible for the list of descriptions of errors and warnings
- Symbol Module – describes special symbols and their priorities used by creator

The class diagram and description are presented below:

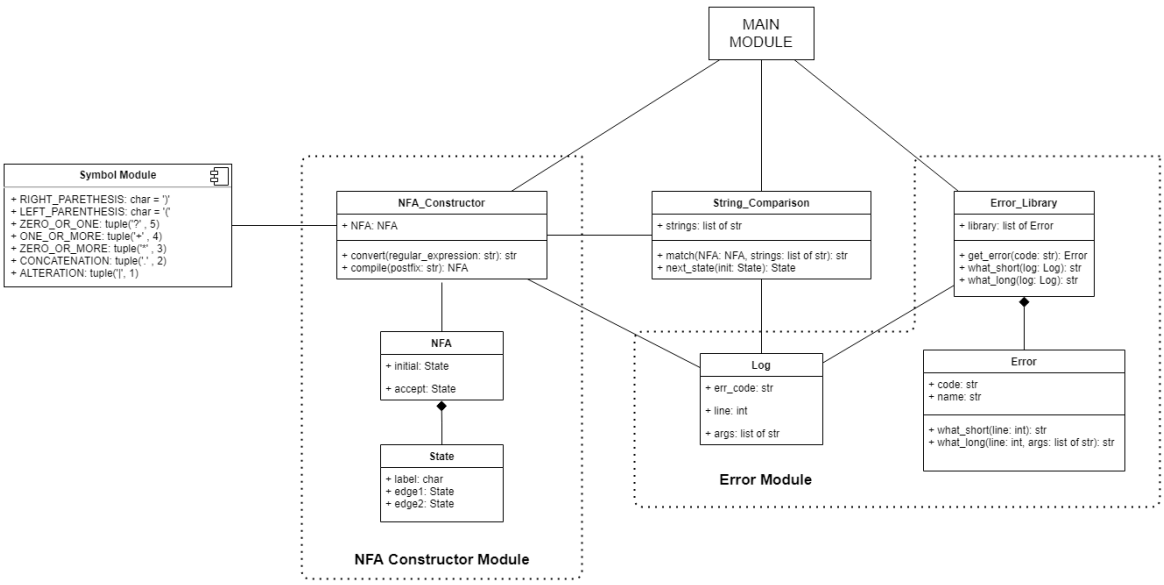

Figure 1. Class Diagram

Class NFA_Constructor

NFA: NFA	NFA object
convert(regular_expresion: str): str	The function transform source text from file with regular expression and converts it to the Postfix Notation (Reverse Polish notation) using Shunting-Yard Algorithm.
compile(postfix: str): NFA	The function takes already converted to Postfix Notation regular expression and creates NFA object that corresponds to the input

Class NFA

initial: State	The initial state of NFA that can be connected to another NFA's, it is the first state of the NFA
accept: State	The accept state that can be connected to the other NFA's, it is the last state of the NFA

Class State

label: str	label defines the argument that has to be given to proceed the transition, when null or None the transition is equivalent to ε transition
edge1: State	edge1 defines one (first) of the connections to other states
edge2: State	edge1 defines one (second) of the connections to other states

Class String_Comparison

strings: list of str	The strings that was in the input file and where separated by end of line (‘\n’) are now in the list and every one of them will be checked whether can be created by the NFA
match(NFA: NFA, strings: list of str): str	Function checks whether the strings can be created by the NFA
next_state(init: State): list of State	returns set of States that can be reached from the given state with ε transition

Class Log

err_code: str	The error/warning code encountered
line: int	The line at which the error/warning occurred
args: list of str	List of additional arguments, used to produce the verbose error description

Class Error_Library

library: list of Error	A list of Error objects
get_error(code: str): Error	A method returning an Error object given the error code
what_short(log: Log): str	A method generating a short error description based on a Log object
what_long(log: Log): str	A method generating a verbose error description based on a Log object

Class Error

code: str	The error code
name: str	The name of the error
what_short(log: Log): str	Method returning short error description
what_long(log: Log): str	Method returning verbose error description

Data structures

Structure	Description
ErrorLibrary	List of error and warning definitions. It consists of Error objects. As the error codes are unique, it should be a list that allows such behavior
NFA	The data structure that consists of two States initial (the first state) and accept (the last state). The States that are between those two given can be accessed by the direct connections between States as an objects.
State	The data structure that connects to other instances of itself. It corresponds to nodes and edges in the NFA diagram.

Module descriptions

The most important part of the modules are presented below: the execution of the main module, the conversion to Postfix Notation, creation of the NFA

Main Module:

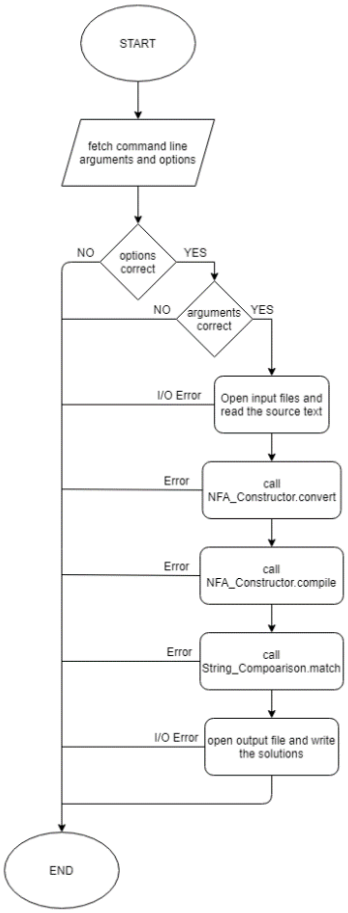

Figure 2. Main module

Conversion to Postfix Notation:

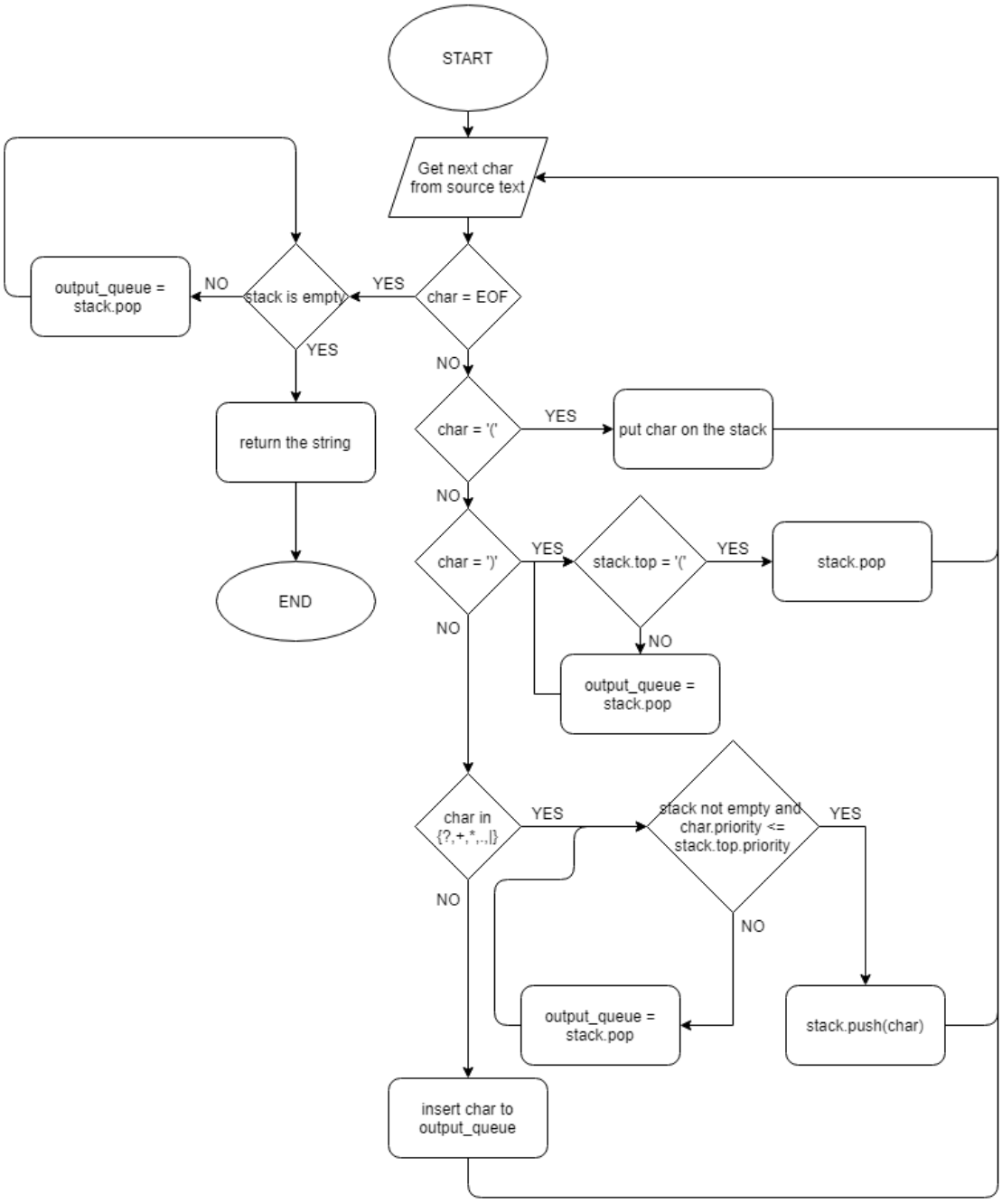

Figure 3. Conversion to Postfix Notation

Creation of the NFA:

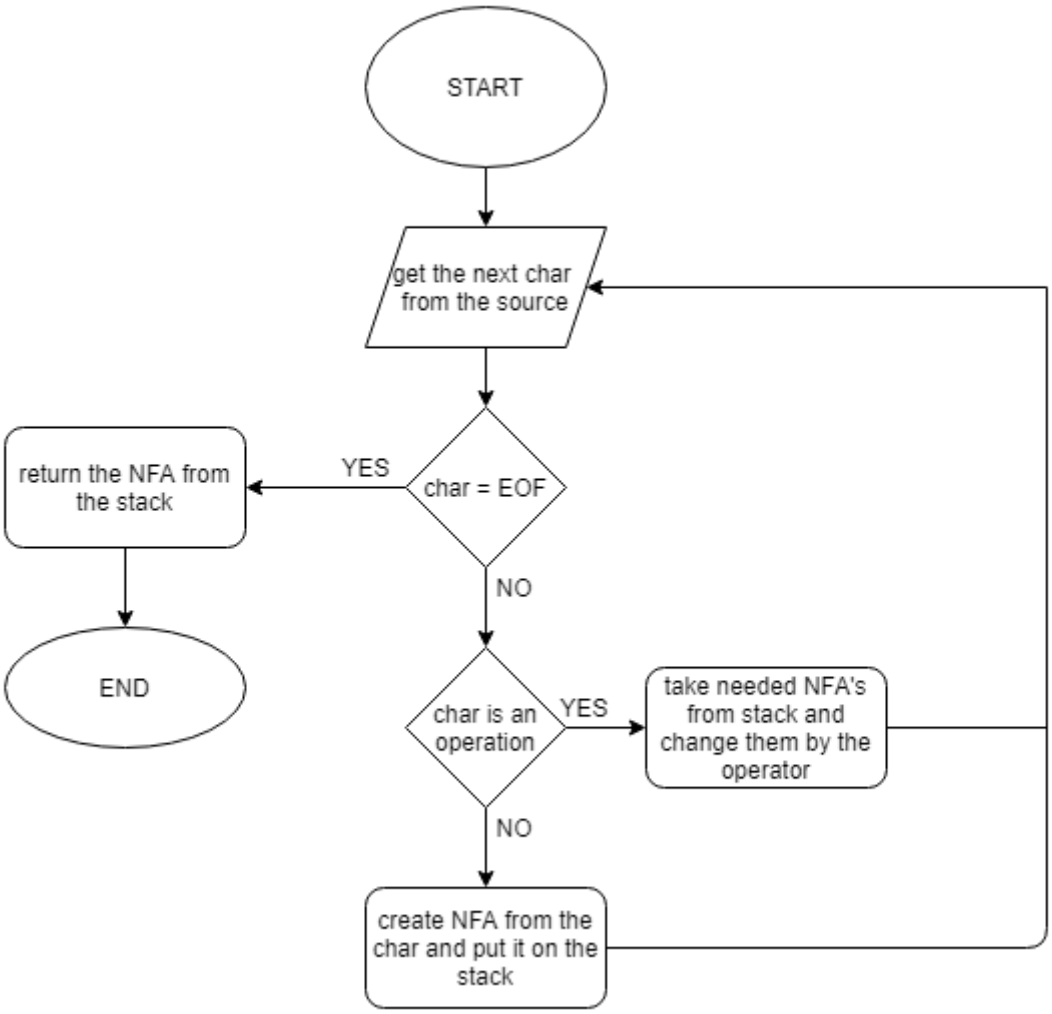

Figure 4. Creation of the NFA

The flowcharts for conversion and NFA creation does not include error handling. At any point in the execution, when an error occurs, an exception will be thrown, or a warning will be added to the log list.

Input/output description

The program uses two text files as input, and outputs text into another file. The NFA creation and string checking works as previously described. Below are some basic examples of usage of this program.

Input		Output
file1:	1*0	False – RE: 1*0, String: 1111
file2:	1111	True – RE: 1*0, String: 10
	10	True – RE: 1*0, String: 0
	0	False – RE: 1*0, String: 100
	100	

Example 1: Basic check of some strings

Input	Output
file1: 1*0 **file2:** $22 1110	Error e30: Not defined character in String file

Example 2: Incorrect usage – the string characters not in alphabet *(more about errors in Errors and Warnings section)*

Input	Output
file1: 1)*0 **file2:** 22 1110	Error e11: Lack of left parenthesis

Example 3: Incorrect usage – lack of left parenthesis *(more about errors in Errors and Warnings section)*

Command Line Interface

The program is operated with a command line interface (CLI). The basic usage of CLI is presented below:

> const regural_expression_file strings_file output_file

The program takes 'regural_expression_file' as an input to the NFA constructor and 'strings_file' to the strings checker. The output text is written into 'output_file'. The errors and warnings are outputted to stdout. When there are only two arguments the program will write the outputs to the default file named 'output' while when there is less than two or too many arguments the error is raised.

Errors and Warnings

During computation the program can encounter errors and warnings that are caused by incorrect or unexpected input. In case when there is an error raised the program informs user of the type of error and stops further execution. In case of a warning program informs user of the type of warning and continues execution. The errors and warnings are presented in the tables below.

Category	**Code**	**Error Name**	**Description**
Conversion	e10	Not defined character in RE	The regular expression contains character that is not number or a letter or one of the special symbols { (,) ,? , + , * , . , | }
	e11	Lack of left parenthesis	The regular expression reader encountered ')' while there was not corresponding symbol '(' before
	e12	Lack of right parenthesis	The regular expression reader encountered '(' that was not closed by the ')' symbol
	e13	Invalid operation symbol placement - start	The regular expression reader encounter on one on the following symbols at the beginning of the RE { ? , + , * , . , | }

	e14	Invalid operation symbol placement - neighbour	Pair of the following symbols were encountered next to each { . , | } other
	e15	Invalid operation symbol placement - pharenthesis	One of the following was placed after '(' symbol { ? , + , * , . , | } or one of the following was placed before ')' symbol { . , | }
Strings	e30	Not defined character in String file	The strings contains character that is not a number or a letter
CLI	e83	No Input File	The input file was not specified
Other	e98	I/O Error	There was an error with the file I/O
	e99	Internal Error	There was an internal error caused by a third party

Category	**Code**	**Warning Name**	**Description**
Conversion	w10	RE Whitespace	The regular expression contains whitespace character at the end, anything that is placed after that character will not be read
	w11	Repetition operators close to each other	Pair of the following symbols were encountered next to each other what can cause unexpected results { ? , + , * }
String	w30	Whitespace argument	One of the Strings is a whitespace
CLI	w80	Overwrite Warning	The input file is the same as the output file

Others

I intend to implement project in Python.

IV. Functional test cases

The following test cases check all the warnings and errors available in the MacroGenerator and a few correct usage cases. The tests don't include the CLI tests.

Correct usage

Test case	Input	Output
Basic RE with one string that can be created	file1: 1* file2: 111	True – RE: 1*, String: 111
Basic RE with one string that cannot be created	file1: 1* file2: 110	False – RE: 1*, String: 110
Basic RE with multiple string that can be created	file1: 1* file2: 111 1	True – RE: 1*, String: 111 True – RE: 1*, String: 1

Basic RE with multiple string that cannot be created	file1: file2:	1* 110 0xa	False – RE: 1*, String: 110 False – RE: 1*, String: 0xa
Basic RE with multiple string	file1: file2:	1*(0|a)+.x 110ax aaaax 1111x	True – RE: 1*(0|a)+.x , String: 110ax True – RE: 1*(0|a)+.x , String: aaaax False – RE: 1*(0|a)+.x , String: 1111x

Errors

Test case	Input		Output
Not defined character in RE	file1: file2:	1*/a 111	*error e10*
Lack of left parenthesis	file1: file2:	1*a|c) 110	*error e11*
Lack of right parenthesis	file1: file2:	1*(a|c 111	*error e12*
Invalid operation symbol placement - start	file1: file2:	+1* 111	*error e13*
Invalid operation symbol placement - neighbour	file1: file2:	1*a.|c 111	*error e14*
Invalid operation symbol placement - parenthesis	file1: file2:	1(*a.|)c 111	*error e15*
Not defined character in String file	file1: file2:	1* 11*1	*error e30*

Warnings

Test case	Input		Output
RE Whitespace	file1: file2:	1* xa 111	True – RE: 1*, String: 111 *warning w10*
Repetition operators close to each other	file1: file2:	1*+ 111	True – RE: 1*, String: 111 *warning w11*
Whitespace argument (I present '_' as a ' ')	file1: file2:	1* _	True – RE: 1*, String: _ *warning w30*