Homework 3

The objective of this lab is to develop a tool for regular expression matching. Given a regular expression and a text, your program should print either "ACCEPT" or "REJECT". The task is accomplished by carrying out the following steps [1]:

1. Read and parse the regular expression
2. Construct the corresponding ε-NFA
3. Convert the ε-NFA into an equivalent DFA
4. Minimize the DFA
5. Simulate the resulting DFA, with input text

You are free to choose the programming language in which you implement the algorithms.

Regular Expressions

The syntax of the expressions you should be able to handle is described by the following grammar:

$$R \rightarrow a \mid \ R1|R2 \ \mid R_1R_2 \mid R^* \mid (R)$$

where *a* ranges over Σ (in this case, alphanumeric characters). The expressions are interpreted according to Table 1.

Expression	Matches
a	The corresponding ASCII character
$R_1\|R_2$	R_1 or R_2
R_1R_2	R_1 followed by R_2
R*	Any number of repetitions of R
(R)	R

Table 1: Interpretation of this lab's regular expressions.

Implementation

Reading and parsing the regular expression

Construction of the ε-NFA (The Thompson-McNaughton-Yamada construction -aka Thompson construction- algorithm.)

In the textbook, it's stated that for each regular expression there exists an ε-NFA accepting the same language. The proof presented in class and in the book is constructive – it uses an inductive construction, which produces from a regular expression a corresponding ε-NFA. Reading through and understanding this proof will help you carry out this step.

Converting the ε-NFA into an equivalent DFA (The subset/powerset construction algorithm)

The construction converting an ε-NFA into an equivalent DFA is described in the text book.

Minimizing the DFA

The previous step is essentially a subset construction that usually results in large automata with many equivalent states. In order to minimize the DFA, refer to the textbook and the Wikipedia article on DFA minimization [2].

Simulating the DFA

To search for strings matching the original regular expression, the DFA should be simulated. Your tool should inspect the input text and look if it matches the regular expression.

Input regular expression and text will be provided as command line arguments hence your program should read them accordingly. For example:

> *match* (a|b)*b aaabbbaabb

ACCEPT

References

[1] Russ Cox. Regular Expression Matching Can Be Simple and Fast, 2007 (last access: 1/12/2017) https://swtch.com/~rsc/regexp/regexp1.html

[2] Wikipedia. DFA Minimization. (last access: 1/12/2017) https://en.wikipedia.org/wiki/DFA_minimization