Practicum 2:
Regular Expressions and Finite Automata

Contact info:
Address *ELIS (floor -3), Technicum (building I), Sint-Pietersnieuwstraat 41*
Email *compiler@elis.ugent.be*

IMPORTANT: You should always hand in your solutions using the Dropbox functionality of Minerva, both an intermediate version at the end of the lab as well as a final one before the deadline. Use the `make_tarball.sh` script to generate an archive, and make sure you send it to the teaching assistants.

DEADLINE: March 3, 23:59.

This is an **individual** assignment, each student from each group needs to hand in his or her own solution.

1 Goal

For this assignment you have to build an automaton for a given regular expression using *JFLAP*. JFLAP is a package of graphical tools which can be used as an aid in learning the basic concepts of Formal Languages and Automata Theory. You can find more information about JFLAP at `http://www.jflap.org`. You'll need to save the automata you've constructed as denoted in Section 2. Be aware that you **strictly follow the naming conventions**! To prevent mistakes it is recommended that you edit the files that can be found in the tarball for this lab session, since they already have the correct names.

2 Assignment

We start with the following lexical specification:

```
token A: (ab*|a)
token B: (a(b*)a)?
token C: (a?|b+)
```

1. For each of these tokens, construct an NFA that recognizes it. Strictly use the algorithm described in Chapter 2 of the course notes. Each of these automata must be saved as `token_a.jff`, `token_b.jff` and `token_c.jff` respectively. You can use the `step with closure` functionality under the `Input` menu in JFLAP to see which states your automaton visits for different inputs and whether or not it accepts those inputs.

2. Merge these automata into one big automaton that recognizes all three tokens. Keep the acceptance states for the tokens separated, so you can determine which token was recognize by just looking at the acceptance state. Save this automaton as `nfa.jff`. In order to make the next task easier you should try to optimize the automaton by merging consecutive λ edges, *if possible*.

3. Construct the equivalent DFA for the NFA you just created. Use the algorithm described in the textbook. Write down the steps of the algorithm down as shown in Table 1:

   Save this table in the file `nfa2dfa.csv`, which is also bundled with the initial files. Draw the resulting DFA and save it as `dfa.jff`.

4. For each acceptance state of this DFA, indicate which token is recognized. Use the following precedence rules:

   - a longer match always takes precedence on a shorter match
   - for two matches of equal length, the first specified token takes precedence (e.g. A takes precedence over B, which in turn takes precedence over C).

   Show which states the DFA goes through for the input `abaabbaba`. Which tokens will be recognized? Save your answer in a file `dfa-answer.txt`.

state D_x	input eg. a	$edge(D_x, a)$	$DFAedge(D_x, a)$	new state D_y
0	ε	...	...	D_0
D_0	a	...	...	D_1

Table 1: State and transition table of a finite automaton.