CMPE 3004 Formal Languages & Automata Theory Project
Due May 15, 2013.

Regular Expression Engine

(This document contains 5 pages. Read carefully all parts of this document.)

Introduction

In this project, you will implement a "Regular Expression Engine." The engine will be used for searching patterns (which match a given regular expression) in a given text file. Your program will take two inputs: a regular expression and a text file. As output, your program will print those lines of the text file which contain strings that match the given regular expression.

Definition of a Regular Expression

A definition of regular expressions is given in your text book (Definition 1.52, p. 64). Additionally, we will assume the following:

- The alphabet, Σ, will consists of all printable ASCII characters except the metacharacters. Metacharacters are: (,), |, *, and &.
- We will omit the symbol 'o' in writing regular expressions. That is, instead of R_1 o R_2 , we will simply write R_1R_2.
- The textbook uses the symbol "∪" to denote the union operator. Since this symbol is a non-keyboard character, we will use the symbol "|" to denote the union operator.

Infix to Postfix

It is easier to process a regular expression if it is in postfix form. For example the postfix equivalent of the regular expression "**(a|b)*xz***" is given by the expression "**ab|*x&z*&**". Note that the concatenation symbol is explicitly shown as the symbol "&" in this postfix expression. This makes it easier to process the postfix expression. So, before processing the regular expression, you need to convert it into postfix form. A Java code which makes this conversion will be provided.

Project Parts

The project will consist of two parts:
Part I: Construction of an NFA from a given regular expression.
Part II: Processing the given text file with the NFA.

In part I, you should first convert the given (infix) regular expression to postfix form. Then, process this postfix expression and construct the NFA step by step. An example start-up algorithm (Algorithm 1) for processing the postfix expression and for building the final NFA is given in this document. For constructing the individual NFAs, use the construction methods which we have seen in class (Lemma 1.55, p. 67 in textbook).

After constructing the final NFA, you should process the given text file. Process the text file line by line. That is, begin with the first line of text and check whether the NFA accepts it or not. If it accepts then print that line, if it does not accept then print nothing. Proceed in this fashion until the end of the text. An example algorithm (Algorithm 2) for simulating the processing of an NFA for a given string is provided in this document.

For the first part you are given a class design with the accompanying code in Java, so it could be easier for you to start with it. It is up to you to decide whether to use that code or not, you may also use it partially. What is required is that your code should be able to build NFA that recognizes given regex after translating it to postfix form and then be able to process a file and output the matching lines as expected.

Bonus (30 points): Develop a simple text editor with the following properties: It can open a text file, it can get a regex, and highlight those parts of the text which matches the regex.

Project Groups
You can work in groups. There can be at most 2 people in a single group.

Submission and Grading: (Please read very carefully!)

1. You must provide source code and send it to cmpe4003@gmail.com before the deadline.

2. Comment and indent your code clearly.

3. You must prepare a clear detailed report of your work including your "reasoning" (design and implementation details) and at least "5 test runs" on large files with regular expressions of different complexities. Present this report both as hardcopy and also send it to the email address given above.

4. You must do your own work. You may discuss high level ideas with classmates, but you must write them ALONE in an isolated environment! If you get help, you must acknowledge that help in your writing. IT'S NOT ALLOWED TO USE SOMEBODY ELSE's CODE PARTLY or FULLY – THIS MAY BE A FRIEND, WEB PAGE OR A BOOK. WE HAVE ZERO TOLERANCE WHEN IT COMES TO CHEATING, UNIVERSITY'S RULES and REGULATIONS APPLY.

5. With appointment, we will ask you to present your work for 5-10 minutes. Questions will be asked to group members separately. If one of the group members cannot explain what they have done clearly, it is considered cheating. Therefore, do only what you can.

Algorithm 1 for constructing an NFA from a given postfix regular expression:

```
while (not end of postfix expression) {
        c = next character in postfix expression;
        if (c == '&') {
                nfa2 = pop();
                nfa1 = pop();
                push(NFA that accepts the concatenation of L(nfa1) followed by L(nfa2));
        }
        else if (c == '|') {
                nfa2 = pop();
                nfa1 = pop();
                push(NFA that accepts L(nfa1) ∪ L(nfa2));
        }
        else if (c == '*') {
                nfa = pop();
                push(NFA that accepts L(nfa)*);
        }
        else {
                push(NFA that accepts a single character c);
        }
}
```

Algorithm 2 for simulating the processing of an NFA, M = (Q, Σ, δ, S, F), for a given input string.

```
T = ∅; P = ∅;
while (true) {
        P = P ∪ S;
        // compute the epsilon closure of P.
        do {
                T = P;
                for (each δ(q, ε) where q∈P )
                        P = P ∪ δ(q, ε);
        } while (T ≠ P ); // loop again if P is modified

        if (T ∩ F ≠ ∅)
                break;

        // if it is the end of input string then break
        if (input empty)
                break;

        c = next character in the string

        // compute next state of the NFA
        T = ∅;
        for (each δ(q, c) where q∈P )
                T = T ∪ δ(q, c);

        if (T ∩ F ≠ ∅)
                break;
        P = T;
}
accept iff (T ∩ F ≠ ∅);
```

Sample Test run

```
USAGE: java Main '(infix regexp)' (file)

$ java Main 'tam|(b*az)' test.txt

Regex in Postfix form:
----------------------
ta&m&b*a&z&|

Following NFA was built:
------------------------
startState = 15
acceptStates = [13, 16]
allStates = [1, 2, 3, ...]
edges:
(5, 6, 'm')
(12, 13, ε)
(10, 11, ε)
(9, 10, ε)
(1, 2, 't')
(11, 12, 'a')
(7, 8, 'b')
(8, 10, ε)
...

NFA simulation results :

ACCEPTED LINE (0) : tamtamlar caliyor.
ACCEPTED LINE (4) : tatatamam.
ACCEPTED LINE (5) : mxazabbaaaabcaz
ACCEPTED LINE (6) : iki cambaz bir ipte oynamaz

NFA simulation took: 61.925702 ms
```