Coursework 8 (Scala, Regular Expressions

This coursework is worth 10%. It is about regular expressions and pattern matching. The first part is due on 30 November at 11pm; the second, more advanced part, is due on 7 December at 11pm. The second part is not yet included. For the first part you are asked to implement a regular expression matcher. Make sure the files you submit can be processed by just calling `scala <<filename.scala>>`.

Important: Do not use any mutable data structures in your submissions! They are not needed. This excluded the use of `ListBuffers`, for example. Do not use `return` in your code! It has a different meaning in Scala, than in Java. Do not use `var`! This declares a mutable variable. Make sure the functions you submit are defined on the "top-level" of Scala, not inside a class or object.

Disclaimer!!!!!!!!

It should be understood that the work you submit represents your own effort! You have not copied from anyone else. An exception is the Scala code I showed during the lectures or uploaded to KEATS, which you can freely use.

Part 1 (6 Marks)

The task is to implement a regular expression matcher based on derivatives of regular expressions. The implementation can deal with the following regular expressions, which have been predefined file re.scala:

r	$::=$	$\mathbf{0}$	cannot match anything
	$\mid$	$\mathbf{1}$	can only match the empty string
	$\mid$	c	can match a character (in this case c)
	$\mid$	$r_1 + r_2$	can match a string either with r_1 or with r_2
	$\mid$	$r_1 \cdot r_2$	can match the first part of a string with r_1 and then the second part with r_2
	$\mid$	r^*	can match zero or more times r

Why? Knowing how to match regular expressions and strings fast will let you solve a lot of problems that vex other humans. Regular expressions are one of the fastest and simplest ways to match patterns in text, and are endlessly useful for searching, editing and analysing text in all sorts of places. However, you need to be fast, otherwise you will stumble upon problems such as recently reported at

- http://stackstatus.net/post/147710624694/outage-postmortem-july-20-2016
- https://vimeo.com/112065252
- http://davidvgalbraith.com/how-i-fixed-atom/

Tasks (file re.scala)

(1a) Implement a function, called *nullable*, by recursion over regular expressions. This function test whether a regular expression can match the empty string.

$$
\begin{array}{lcl}
nullable(\mathbf{0}) & \stackrel{\text{def}}{=} & false \\
nullable(\mathbf{1}) & \stackrel{\text{def}}{=} & true \\
nullable(c) & \stackrel{\text{def}}{=} & false \\
nullable(r_1 + r_2) & \stackrel{\text{def}}{=} & nullable(r_1) \vee nullable(r_2) \\
nullable(r_1 \cdot r_2) & \stackrel{\text{def}}{=} & nullable(r_1) \wedge nullable(r_2) \\
nullable(r^*) & \stackrel{\text{def}}{=} & true
\end{array}
$$

[1 Mark]

(1b) Implement a function, called *der*, by recursion over regular expressions. It takes a character and a regular expression as arguments and calculates the derivative regular expression.

$$
\begin{array}{lcl}
der\ c\ (\mathbf{0}) & \stackrel{\text{def}}{=} & \mathbf{0} \\
der\ c\ (\mathbf{1}) & \stackrel{\text{def}}{=} & \mathbf{0} \\
der\ c\ (d) & \stackrel{\text{def}}{=} & if\ c = d\ then\ \mathbf{1}\ else\ \mathbf{0} \\
der\ c\ (r_1 + r_2) & \stackrel{\text{def}}{=} & (der\ c\ r_1) + (der\ c\ r_2) \\
der\ c\ (r_1 \cdot r_2) & \stackrel{\text{def}}{=} & if\ nullable(r_1) \\
 & & then\ ((der\ c\ r_1) \cdot r_2) + (der\ c\ r_2) \\
 & & else\ (der\ c\ r_1) \cdot r_2 \\
der\ c\ (r^*) & \stackrel{\text{def}}{=} & (der\ c\ r) \cdot (r^*)
\end{array}
$$

[1 Mark]

(1c) Implement the function *simp*, which recursively traverses a regular expression from inside to outside, and simplifies every sub-regular-expressions on the left to the regular expression on the right, except it does not simplify inside *-regular expressions.

$$
\begin{array}{lcl}
r \cdot \mathbf{0} & \mapsto & \mathbf{0} \\
\mathbf{0} \cdot r & \mapsto & \mathbf{0} \\
r \cdot \mathbf{1} & \mapsto & r \\
\mathbf{1} \cdot r & \mapsto & r \\
r + \mathbf{0} & \mapsto & r \\
\mathbf{0} + r & \mapsto & r \\
r + r & \mapsto & r
\end{array}
$$

For example

$$(r_1 + \mathbf{0}) \cdot \mathbf{1} + ((\mathbf{1} + r_2) + r_3) \cdot (r_4 \cdot \mathbf{0})$$

simplifies to just r_1. [1 Mark]

(1d) Implement two functions: The first, called *ders*, takes a list of characters as arguments and a regular expression and buids the derivative as follows:

$$\begin{array}{lcl} \mathit{ders}\ \mathit{Nil}\ (r) & \stackrel{\text{def}}{=} & r \\ \mathit{ders}\ c :: cs\ (r) & \stackrel{\text{def}}{=} & \mathit{ders}\ cs\ (\mathit{simp}(\mathit{der}\ c\ r)) \end{array}$$

The second, called *matcher*, takes a string and a regular expression as arguments. It builds first the derivatives according to *ders* and at the end tests whether the resulting redular expression can match the empty string (using *nullable*). For example the *matcher* will produce true if given the regular expression $a \cdot b \cdot c$ and the string *abc*. [1 Mark]

(1e) Implement the function *replace*: it searches (from the left to right) in string s_1 all the non-empty substrings that match the regular expression—these substrings are assumed to be the longest substrings matched by the regular expression and assumed to be non-overlapping. All these substrings in s_1 are replaced by s_2. For example given the regular expression

$$(a \cdot a)^* + (b \cdot b)$$

the string *aabbbaaaaaaabaaaaabbaaaabb* and replacement string *c* yields the string

ccbcabcaccc

[2 Mark]

Part 2 (4 Marks)

Coming soon.