Coursework 8 (Scala, Regular Expressions

This coursework is worth 10% and is due on XXXX at 16:00. 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. Feel free to copy any code you need from files `knight1.scala`, `knight2.scala` and `knight3.scala`. 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.

Task

The task is to implement a regular expression matcher based on derivatives of regular expressions. The implementation should be able to deal with the usual (basic) regular expressions

r	$::=$	$\mathbf{0}$	cannot match anything
	$\mid$	$\mathbf{1}$	can only match the empty string
	$\mid$	c	can match a character c
	$\mid$	$r_1 + r_2$	can match either with r_1 or with r_2
	$\mid$	$r_1 \cdot r_2$	can match first with r_1 and then with r_2
	$\mid$	r^*	can match zero or more times r
	$\mid$	$r^{\{\uparrow n\}}$	can match zero upto n times r
	$\mid$	$r^{\{n\}}$	can match exactly n times r

Implement a function called *nullable* by recursion over regular expressions:

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

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

Be careful that your implementation of *nullable* and *der* c satisfies for every r the following two properties (see also Question 2):

- $\mathit{nullable}(r)$ if and only if $[] \in L(r)$
- $L(\mathit{der}\, c\, r) = \mathit{Der}\, c\, (L(r))$

Important! Your implementation should have explicit cases for the basic regular expressions, but also explicit cases for the extended regular expressions. That means do not treat the extended regular expressions by just translating them into the basic ones. See also Question 2, where you are asked to explicitly give the rules for *nullable* and *der* c for the extended regular expressions.

Question 1

What is your King's email address (you will need it in Question 3)?

Question 2

This question does not require any implementation. From the lectures you have seen the definitions for the functions *nullable* and *der* c for the basic regular

expressions. Give the rules for the extended regular expressions:

$$
\begin{array}{ll}
\mathit{nullable}([c_1 c_2 \ldots c_n]) & \stackrel{\text{def}}{=} ? \\
\mathit{nullable}(r^+) & \stackrel{\text{def}}{=} ? \\
\mathit{nullable}(r^?) & \stackrel{\text{def}}{=} ? \\
\mathit{nullable}(r^{\{n,m\}}) & \stackrel{\text{def}}{=} ? \\
\mathit{nullable}(\sim r) & \stackrel{\text{def}}{=} ? \\
\\
\mathit{der}\, c\, ([c_1 c_2 \ldots c_n]) & \stackrel{\text{def}}{=} ? \\
\mathit{der}\, c\, (r^+) & \stackrel{\text{def}}{=} ? \\
\mathit{der}\, c\, (r^?) & \stackrel{\text{def}}{=} ? \\
\mathit{der}\, c\, (r^{\{n,m\}}) & \stackrel{\text{def}}{=} ? \\
\mathit{der}\, c\, (\sim r) & \stackrel{\text{def}}{=} ?
\end{array}
$$

Remember your definitions have to satisfy the two properties

- $\mathit{nullable}(r)$ if and only if $[] \in L(r)$
- $L(\mathit{der}\, c\, r)) = \mathit{Der}\, c\, (L(r))$

Question 3

Implement the following regular expression for email addresses

$$([a\text{-}z0\text{-}9\_.-]^+) \cdot @ \cdot ([a\text{-}z0\text{-}9.-]^+) \cdot . \cdot ([a\text{-}z.]^{\{2,6\}})$$

and calculate the derivative according to your email address. When calculating the derivative, simplify all regular expressions as much as possible by applying the following 7 simplification rules:

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

Write down your simplified derivative in a readable notation using parentheses where necessary. That means you should use the infix notation $+$, $\cdot$, * and so on, instead of code.

Question 4

Suppose *[a-z]* stands for the range regular expression $[a, b, c, \ldots, z]$. Consider the regular expression $/ \cdot * \cdot (\sim ([a\text{-}z]^* \cdot * \cdot / \cdot [a\text{-}z]^*)) \cdot * \cdot /$ and decide wether the following four strings are matched by this regular expression. Answer yes or no.

1. `"/**/"`
2. `"/*foobar*/"`
3. `"/*test*/test*/"`
4. `"/*test/*test*/"`

Also test your regular expression matcher with the regular expression $a^{\{3,5\}}$ and the strings

5. `aa`
6. `aaa`
7. `aaaaa`
8. `aaaaaa`

Does your matcher produce the expected results?

Question 5

Let r_1 be the regular expression $a \cdot a \cdot a$ and r_2 be $(a^{\{19,19\}}) \cdot (a^?)$. Decide whether the following three strings consisting of as only can be matched by $(r_1^+)^+$. Similarly test them with $(r_2^+)^+$. Again answer in all six cases with yes or no.

These are strings are meant to be entirely made up of as. Be careful when copy-and-pasting the strings so as to not forgetting any a and to not introducing any other character.

1. ```
   "aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
   aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
   aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa"
   ```
2. ```
   "aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
   aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
   aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa"
   ```
3. ```
   "aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
   aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
   aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa"
   ```
   ```