6CCS3CFL – Course Work 1
Luca-Dorin Anton- 1710700

Q1: What is your King's email address (you will need it in Question 5)?

A: luca-dorin.anton@kcl.ac.uk

Q2: Can you please list all programming languages in which you have already written programs (like spent at least a good working day fiddling with the program or programs)? This is just for my curiosity to estimate what your background is.

A: Python, Scala, JavaScript, Java, C++, C, Rust, Haskell, Lua, Assembly

Q3: From the lectures you have seen the definitions for the functions nullable and der for the basic regular expressions. Implement and write down rules for the extended regular expressions:

$$
\begin{array}{ll}
\text{nullable}([c1, c2, \ldots, cn]) & \stackrel{\text{def}}{=} \text{false} \\
\text{nullable}(r^{+}) & \stackrel{\text{def}}{=} \text{nullable}(r) \\
\text{nullable}(r^{?}) & \stackrel{\text{def}}{=} \text{true} \\
\text{nullable}(r^{\{n\}}) & \stackrel{\text{def}}{=} \text{if } n == 0 \text{ true else nullable}(r) \\
\text{nullable}(r^{\{..m\}}) & \stackrel{\text{def}}{=} \text{true} \\
\text{nullable}(r^{\{n..\}}) & \stackrel{\text{def}}{=} \text{if } n == 0 \text{ true else nullable}(r) \\
\text{nullable}(r^{\{n..m\}}) & \stackrel{\text{def}}{=} \text{if } n == 0 \text{ true else nullable}(r) \\
\text{nullable}(\sim r) & \stackrel{\text{def}}{=} \neg\, \text{nullable}(r)
\end{array}
$$

$$
\begin{array}{ll}
\text{der c } ([c_1, c_2, \ldots, c_n]) & \stackrel{\text{def}}{=} \text{if c in } [c_1, c_2, \ldots, c_n] \text{ then } 1 \text{ else } 0 \\
\text{der c } (r^{+}) & \stackrel{\text{def}}{=} (\text{der c r}) \bullet r^{*} \\
\text{der c } (r^{?}) & \stackrel{\text{def}}{=} \text{der c r} \\
\text{der c } (r^{\{n\}}) & \stackrel{\text{def}}{=} \text{if } n = 0 \ \mathbf{0} \text{ else } (\text{der c r}) \bullet r^{\{n-1\}} \\
\text{der c } (r^{\{..m\}}) & \stackrel{\text{def}}{=} \text{if } n = 0 \ \mathbf{0} \text{ else } (\text{der c r}) \bullet r^{\{..n-1\}} \\
\text{der c } (r^{\{n..\}}) & \stackrel{\text{def}}{=} \text{if } n = 0 \ \mathbf{1} \text{ else } (\text{der c r}) \bullet r^{\{n-1..\}} \\
\text{der c } (r^{\{n..m\}}) & \stackrel{\text{def}}{=} \text{if } n = m = 0 \ \mathbf{1} \\
 & \quad \text{else if } n = 0 \ (\text{der c r}) \bullet r^{\{..m-1\}} \\
 & \quad \text{else } \text{der}(c, r) \bullet r^{\{n-1, m-1\}} \\
\text{der c } (\sim r) & \stackrel{\text{def}}{=} \sim(\text{der c r})
\end{array}
$$

Given the definitions of nullable and der, it is easy to implement a regular expression matcher. Test your regular expression matcher with (at least) the examples:

string	$a^{\{3\}}$	$(a^?)^{\{3\}}$	$a^{\{..3\}}$	$a^{\{3..5\}}$	$(a^?)^{\{3..5\}}$
[]	false	true	true	false	true
a	false	true	true	false	true
aa	false	true	true	false	true
aaa	true	true	true	true	true
aaaa	false	false	false	true	true
aaaaa	false	false	false	true	true

Does your matcher produce the expected results?

Yes, the matcher does indeed produce the expected results.

Q4:
The idea is that the function f determines which character(s) are matched, namely those where f returns true. In this question implement CFUN and define

nullable(CFUN(f)) $\stackrel{\text{def}}{=}$ false
der c (CFUN(f)) $\stackrel{\text{def}}{=}$ if f(c) is true **1** else **0**

in your matcher and then also give definitions for

c $\stackrel{\text{def}}{=}$ CFUN(f_1), where $f_1(x)$ = if x = c true else false
$[c_1, c_2, \ldots, c_n]$ $\stackrel{\text{def}}{=}$ CFUN(f_2), where $f_2(x)$ = if x in $[c_1, c_2, \ldots, c_n]$ true else false
ALL $\stackrel{\text{def}}{=}$ CFUN(f_3) where $f_3(x)$ = true

You can either add the constructor CFUN to your implementation in Question 3, or you can implement this questions first and then use CFUN instead of RANGE and CHAR in Question 3.

Q5:
Suppose [a-z0-9_ .-] stands for the regular expression

[a, b, c, . . . , z, 0, . . . , 9, _, ., -] .

Define in your code the following regular expression for email addresses

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

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

(omitted because no point in writing them again)

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

The resulting derivate regular expression is:

$r = ([a\text{-}z0\text{-}9\_.\text{-}]^{*}\ [a\text{-}z0\text{-}9\_.\text{-}]\ [a\text{-}z.]^{\{2..6\}}) + [a\text{-}z.]^{\{0..4\}}$

Note: Scala reports the CFUN functions as 'function1' no matter what. So, I'm trying to make some educated guesses about the actual nature of the character classes in the resulting regex.

Q6: Implement the simplification rules in your regular expression matcher. Consider the regular expression (too hard to copy from pdf, you know which one it is) and decide whether the following four strings are matched by this regular expression. Answer yes or no.

1. "/**/" -- **Yes**
2. "/*foobar*/" -- **Yes**
3. "/*test*/test*/" -- **No**
4. "/*test/*test*/" -- **Yes**

Q7: Let r1 be the regular expression a • a • a and r2 be $(a^{\{19,19\}}) \bullet (a^{?})$. 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.

5. "aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa"

$(r_1^{+})^{+}$ => **yes** $(r_2^{+})^{+}$ => **yes**

6. "aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa"

$(r_1^{+})^{+}$ => **no** $(r_2^{+})^{+}$ => **no**

7. "aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa"

(r_1^+)$^+$ => **no** (r_2^+)$^+$ => yes