Coursework 1 (Strand 1)

This coursework is worth 4% and is due on 11 October at 18:00. You are asked to implement a regular expression matcher and submit a document containing the answers for the questions below. You can do the implementation in any programming language you like, but you need to submit the source code with which you answered the questions, otherwise a mark of 0% will be awarded. You can submit your answers in a txt-file or pdf. Code send as code.

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.

If you have any questions, please send me an email in **good** time.

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

$$\mathbf{0},\ \mathbf{1},\ c,\ r_1 + r_2,\ r_1 \cdot r_2,\ r^*$$

but also with the following extended regular expressions:

$[c_1, c_2, \ldots, c_n]$	a set of characters—for character ranges
r^+	one or more times r
$r^?$	optional r
$r^{\{n\}}$	exactly n-times
$r^{\{..m\}}$	zero or more times r but no more than m-times
$r^{\{n..\}}$	at least n-times r
$r^{\{n..m\}}$	at least n-times r but no more than m-times
$\sim r$	not-regular-expression of r

You can assume that n and m are greater or equal than 0. In the case of $r^{\{n,m\}}$ you can also assume $0 \leq n \leq m$.

Important! Your implementation should have explicit case classes for the basic regular expressions, but also explicit case classes for the extended regular expressions.[1] That means do not treat the extended regular expressions by just translating them into the basic ones. See also Question 3, where you are asked

[1] Please call them `RANGE`, `PLUS`, `OPTIONAL`, `NTIMES`, `UPTO`, `FROM` and `BETWEEN`.

to explicitly give the rules for *nullable* and *der* for the extended regular expressions. So something like $der\, c\, (r^+) \stackrel{\text{def}}{=} der\, c\, (r \cdot r^*)$ is *not* allowed.

The meanings of the extended regular expressions are

$$\begin{array}{rcl} L([c_1, c_2, \ldots, c_n]) & \stackrel{\text{def}}{=} & \{[c_1], [c_2], \ldots, [c_n]\} \\ L(r^+) & \stackrel{\text{def}}{=} & \bigcup_{1 \leq i} . L(r)^i \\ L(r^?) & \stackrel{\text{def}}{=} & L(r) \cup \{[]\} \\ L(r^{\{n\}}) & \stackrel{\text{def}}{=} & L(r)^n \\ L(r^{\{..m\}}) & \stackrel{\text{def}}{=} & \bigcup_{0 \leq i \leq m} . L(r)^i \\ L(r^{\{n..\}}) & \stackrel{\text{def}}{=} & \bigcup_{n \leq i} . L(r)^i \\ L(r^{\{n..m\}}) & \stackrel{\text{def}}{=} & \bigcup_{n \leq i \leq m} . L(r)^i \\ L(\sim r) & \stackrel{\text{def}}{=} & \Sigma^* - L(r) \end{array}$$

whereby in the last clause the set Σ^* stands for the set of *all* strings over the alphabet Σ (in the implementation the alphabet can be just what is represented by, say, the type `Char`). So $\sim\, r$ means in effect "all the strings that r cannot match".

Be careful that your implementation of *nullable* and *der* satisfies for every regular expression r the following two properties (see also Question 3):

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

Question 1 (Unmarked)

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

Question 2 (Unmarked)

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.

Question 3

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

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

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

Remember your definitions have to satisfy the two properties

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

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	$a^?$	$\sim a$	$a^{\{3\}}$	$(a^?)^{\{3\}}$	$a^{\{..3\}}$	$(a^?)^{\{..3\}}$	$a^{\{3..5\}}$	$(a^?)^{\{3..5\}}$
[]								
a								
aa								
aaa								
aaaaa								
aaaaaa								

Does your matcher produce the expected results? Make sure you also test corner-cases, like $a^{\{0\}}$!

Question 4

As you can see, there are a number of explicit regular expressions that deal with single or several characters, for example:

c	matches a single character
$[c_1, c_2, \ldots, c_n]$	matches a set of characters—for character ranges
ALL	matches any character

The latter is useful for matching any string (for example by using ALL^*). In order to avoid having an explicit constructor for each case, we can generalise all these cases and introduce a single constructor $CFUN(f)$ where f is a function from characters to booleans. In Scala code this would look as follows:

```
abstract class Rexp
...
case class CFUN(f: Char => Boolean) extends Rexp
```

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

$$\begin{array}{ll} nullable(CFUN(f)) & \stackrel{\text{def}}{=} \; ? \\ der\, c\, (CFUN(f)) & \stackrel{\text{def}}{=} \; ? \end{array}$$

in your matcher and then also give definitions for

$$\begin{array}{ll} c & \stackrel{\text{def}}{=} CFUN(?) \\ [c_1, c_2, \ldots, c_n] & \stackrel{\text{def}}{=} CFUN(?) \\ ALL & \stackrel{\text{def}}{=} CFUN(?) \end{array}$$

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.

Question 5

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

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

Define in your code 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 own 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 raw code.

Question 6

Implement the simplification rules in your regular expression matcher. Consider the regular expression $/ \cdot * \cdot (\sim\ (ALL^* \cdot * \cdot / \cdot ALL^*)) \cdot * \cdot /$ and decide whether the following four strings are matched by this regular expression. Answer yes or no.

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

Question 7

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.

5. `"aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa`
   `aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa`
   `aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa"`
6. `"aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa`
   `aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa`
   `aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa"`
7. `"aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa`
   `aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa`
   `aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa"`