

## Coursework 1 (Update for 2025: CW is OPTIONAL but recommended)

### 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

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

but also with the following extended regular expressions:

$[c_1, c_2, \dots, c_n]$	a set of characters—for character ranges
$r^+$	one or more times $r$
$r^?$	optional $r$
$r_1 \& r_2$	intersection (matched by both $r_1$ and $r_2$ )
$r^{\{n\}}$	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.<sup>1</sup> That means do not treat the extended regular expressions by just translating them into the basic ones. See also Task 1, where you are asked to explicitly give the rules for *nullable* and *der* for the extended regular expressions. Something like

$$\text{der } c(r^+) \stackrel{\text{def}}{=} \text{der } c(r \cdot r^*)$$

is *not* allowed as answer in Task 1 and *not* allowed in your code.

The meanings of the extended regular expressions are

---

<sup>1</sup>Please call them RANGE, PLUS, OPTIONAL, INTER, NTIMES, UPTO, FROM and BETWEEN.

$$\begin{aligned}
L([c_1, c_2, \dots, c_n]) &\stackrel{\text{def}}{=} \{[c_1], [c_2], \dots, [c_n]\} \\
L(r^+) &\stackrel{\text{def}}{=} \bigcup_{1 \leq i} L(r)^i \\
L(r^?) &\stackrel{\text{def}}{=} L(r) \cup \{\epsilon\} \\
L(r_1 \& r_2) &\stackrel{\text{def}}{=} L(r_1) \cap L(r_2) \\
L(r^{\{n\}}) &\stackrel{\text{def}}{=} L(r)^n \\
L(r^{\{..m\}}) &\stackrel{\text{def}}{=} \bigcup_{0 \leq i \leq m} L(r)^i \\
L(r^{\{n..\}}) &\stackrel{\text{def}}{=} \bigcup_{n \leq i} L(r)^i \\
L(r^{\{n..m\}}) &\stackrel{\text{def}}{=} \bigcup_{n \leq i \leq m} L(r)^i \\
L(\sim r) &\stackrel{\text{def}}{=} \Sigma^* - L(r)
\end{aligned}$$

whereby in the last clause the set  $\Sigma^*$  stands for the set of *all* strings over the alphabet  $\Sigma$  (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 Task 1):

- *nullable*( $r$ ) if and only if  $\epsilon \in L(r)$
- $L(\text{der } c \ r) = \text{Der } c \ (L(r))$

## Task 1

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 (see questionnaire at the end).

Remember your definitions have to satisfy the two properties

- *nullable*( $r$ ) if and only if  $\epsilon \in L(r)$
- $L(\text{der } c \ r) = \text{Der } c \ (L(r))$

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

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

## Task 2

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
$[c_1, c_2, \dots, c_n]$	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

$$\begin{aligned} nullable(CFUN(f)) &\stackrel{\text{def}}{=} ? \\ der\ c\ (CFUN(f)) &\stackrel{\text{def}}{=} ? \end{aligned}$$

in your matcher and then also give definitions for

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

You can either add the constructor  $CFUN$  to your implementation in Task 3, or you can implement this questions first and then use  $CFUN$  instead of  $RANGE$  and  $CHAR$  in Task 3. In an ideal world one would do this task first, but this might confuse you with what you need to do in the previous question.

## Task 3

Suppose  $[a\text{-}z0\text{-}9\_.\text{-}]$  stands for the regular expression

$$[a, b, c, \dots, z, 0, \dots, 9, \_., \text{-}] .$$

Define in your code the following regular expression for email addresses

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

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:

$$\begin{aligned}
r \cdot 0 &\mapsto 0 \\
0 \cdot r &\mapsto 0 \\
r \cdot 1 &\mapsto r \\
1 \cdot r &\mapsto r \\
r + 0 &\mapsto r \\
0 + r &\mapsto r \\
r + r &\mapsto r
\end{aligned}$$

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.

#### Task 4

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*/"`

#### Task 5

Let  $r_1$  be the regular expression  $a \cdot a \cdot a$  and  $r_2$  be  $(a^{\{19,19\}}) \cdot (a^2)$ .

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. `"aa  
aa  
aa"`
2. `"aa  
aa  
aa"`
3. `"aa  
aa  
aa"`

## Answers

Name: \_\_\_\_\_

BSc / MSci                      Year: \_\_\_\_\_

Programming Languages: \_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

### Task 1:

$\text{nullable}([c_1, c_2, \dots, c_n]) \stackrel{\text{def}}{=} \underline{\hspace{10em}}$

$\text{nullable}(r^+)$   $\stackrel{\text{def}}{=} \underline{\hspace{10em}}$

$\text{nullable}(r^?)$   $\stackrel{\text{def}}{=} \underline{\hspace{10em}}$

$\text{nullable}(r_1 \& r_2)$   $\stackrel{\text{def}}{=} \underline{\hspace{10em}}$

$\text{nullable}(r^{\{n\}})$   $\stackrel{\text{def}}{=} \underline{\hspace{10em}}$

\_\_\_\_\_

$\text{nullable}(r^{\{..m\}})$   $\stackrel{\text{def}}{=} \underline{\hspace{10em}}$

\_\_\_\_\_

$\text{nullable}(r^{\{n..\}})$   $\stackrel{\text{def}}{=} \underline{\hspace{10em}}$

\_\_\_\_\_

$\text{nullable}(r^{\{n..m\}})$   $\stackrel{\text{def}}{=} \underline{\hspace{10em}}$

\_\_\_\_\_

$\text{nullable}(\sim r)$   $\stackrel{\text{def}}{=} \underline{\hspace{10em}}$

$der\ c\ ([c_1, c_2, \dots, c_n]) \stackrel{\text{def}}{=} \underline{\hspace{10cm}}$

$der\ c\ (r^+) \stackrel{\text{def}}{=} \underline{\hspace{10cm}}$

$der\ c\ (r^?) \stackrel{\text{def}}{=} \underline{\hspace{10cm}}$

$der\ c\ (r_1 \ \& \ r_2) \stackrel{\text{def}}{=} \underline{\hspace{10cm}}$

$der\ c\ (r^{\{n\}}) \stackrel{\text{def}}{=} \underline{\hspace{10cm}}$

$der\ c\ (r^{\{n\}}) \stackrel{\text{def}}{=} \underline{\hspace{10cm}}$

$der\ c\ (r^{\{..m\}}) \stackrel{\text{def}}{=} \underline{\hspace{10cm}}$

$der\ c\ (r^{\{n.. \}}) \stackrel{\text{def}}{=} \underline{\hspace{10cm}}$

$der\ c\ (r^{\{n..m\}}) \stackrel{\text{def}}{=} \underline{\hspace{10cm}}$

$der\ c\ (r^{\{n..m\}}) \stackrel{\text{def}}{=} \underline{\hspace{10cm}}$

$der\ c\ (r^{\{n..m\}}) \stackrel{\text{def}}{=} \underline{\hspace{10cm}}$

$der\ c\ (\sim r) \stackrel{\text{def}}{=} \underline{\hspace{10cm}}$

$der\ c\ (\sim r) \stackrel{\text{def}}{=} \underline{\hspace{10cm}}$

$der\ c\ (\sim r) \stackrel{\text{def}}{=} \underline{\hspace{10cm}}$

**Task 2:**

$nullable(CFUN(f)) \stackrel{\text{def}}{=} \underline{\hspace{10cm}}$

$der\ c\ (CFUN(f)) \stackrel{\text{def}}{=} \underline{\hspace{10cm}}$

$der\ c\ (CFUN(f)) \stackrel{\text{def}}{=} \underline{\hspace{10cm}}$

$der\ c\ (CFUN(f)) \stackrel{\text{def}}{=} \underline{\hspace{10cm}}$

$c \stackrel{\text{def}}{=} CFUN(\underline{\hspace{10cm}})$

$[c_1, c_2, \dots, c_n] \stackrel{\text{def}}{=} CFUN(\underline{\hspace{10cm}})$

$ALL \stackrel{\text{def}}{=} CFUN(\underline{\hspace{10cm}})$

**Task 3 ('mathematical' notation):**

---

---

---

---

---

**Task 4:**

**1)** Yes / No      **2)** Yes / No      **3)** Yes / No      **4)** Yes / No

**Task 5:**

	$r_1$	$r_2$
1.		
2.		
3.		