Homework 2

1. What is the difference between *basic* regular expressions and *extended* regular expressions?

   Basic regular expressions are $\mathbf{0}$, $\mathbf{1}$, c, $r_1 + r_2$, $r_1 \cdot r_2$, r^*. The extended ones are the bounded repetitions, not, etc.

2. What is the language recognised by the regular expressions $(\mathbf{0}^*)^*$.

   $L(\mathbf{0}^{**}) = \{[]\}$, remember * always includes the empty string

3. Review the first handout about sets of strings and read the second handout. Assuming the alphabet is the set $\{a, b\}$, decide which of the following equations are true in general for arbitrary languages A, B and C:

$$\begin{aligned}
(A \cup B)@C &=^? A@C \cup B@C \\
A^* \cup B^* &=^? (A \cup B)^* \\
A^*@A^* &=^? A^* \\
(A \cap B)@C &=^? (A@C) \cap (B@C)
\end{aligned}$$

   In case an equation is true, give an explanation; otherwise give a counter-example.

   1 + 3 are equal; 2 + 4 are not. Interesting is 4 where $A = \{[a]\}$, $B = \{[]\}$ and $C = \{[a], []\}$ is an counter-example.

   For equations like 3 it is always a good idea to prove the two inclusions

$$A^* \subseteq A^*@A^* \qquad A^*@A^* \subseteq A^*$$

   This means for every string s we have to show

$$s \in A^* \textit{ implies } s \in A^*@A^* \qquad s \in A^*@A^* \textit{ implies } s \in A^*$$

   The first one is easy because $[] \in A^*$ and therefore $s@[] \in A^*@A^*$. The second one says that s must be of the form $s = s_1@s_2$ with $s_1 \in A^*$ and $s_2 \in A^*$. We have to show that $s_1@s_2 \in A^*$. If $s_1 \in A^*$ then there exists an n such that $s_1 \in A^n$, and if $s_2 \in A^*$ then there exists an m such that $s_2 \in A^m$.

   Aside: We are going to show the following power law

$$A^n \,@\, A^m = A^{n+m}$$

We prove that by induction on n.
Case $n = 0$: $A^0 \,@\, A^m = A^{0+m}$ holds because $A^0 = \{[]\}$ and $\{[]\} \,@\, A^m = A^m$ and $0 + m = m$.

Case $n + 1$: The induction hypothesis is

$$A^n \,@\, A^m = A^{n+m}$$

We need to prove

$$A^{n+1} \,@\, A^m = A^{(n+1)+m}$$

The left-hand side is $(A \,@\, A^n) \,@\, A^m$ by the definition of the power operation. We can rearrange that to $A \,@\, (A^n \,@\, A^m)$. [1]
By the induction hypothesis we know that $A^n \,@\, A^m = A^{n+m}$.
So we have $A \,@\, (A^{n+m})$. But this is $A^{(n+m)+1}$ again if we apply the definition of the power operator. If we rearrange that we get $A^{(n+1)+m}$ and are done with what we need to prove for the power law.

Picking up where we left, we know that $s_1 \in A^n$ and $s_2 \in A^m$. This now implies that $s_1 @ s_2 \in A^n @ A^m$. By the power law this means $s_1 @ s_2 \in A^{n+m}$. But this also means $s_1 @ s_2 \in A^*$.

4. Given the regular expressions $r_1 = \mathbf{1}$ and $r_2 = \mathbf{0}$ and $r_3 = a$. How many strings can the regular expressions r_1^*, r_2^* and r_3^* each match?

   r_1 and r_2 can match the empty string only, r_3 can match $[]$, a, aa, ...

5. Give regular expressions for (a) decimal numbers and for (b) binary numbers. Hint: Observe that the empty string is not a number. Also observe that leading 0s are normally not written—for example the JSON format for numbers explicitly forbids this. So 007 is not a number according to JSON.

   Just numbers without leading 0s: $0 + (1..9) \cdot (0..1)^*$; can be extended to decimal; similar for binary numbers

6. Decide whether the following two regular expressions are equivalent $(\mathbf{1} + a)^* \equiv^? a^*$ and $(a \cdot b)^* \cdot a \equiv^? a \cdot (b \cdot a)^*$.

[1] Because for all languages A, B, C we have $(A@B)@C = A@(B@C)$.

Both are equivalent, but why the second? Essentially you have to show that each string in one set is in the other. For 2 this means you can do an induction proof that $(ab)^n a$ is the same string as $a(ba)^n$, where the former is in the first set and the latter in the second.

7. Give an argument for why the following holds: if r is nullable then $r^{\{n\}} \equiv r^{\{..n\}}$.

   This requires an inductive proof. There are a number of ways to prove this. It is clear that if $s \in L(r^{\{n\}})$ then also $s \in L(r^{\{..n\}})$.
   So one way to prove this is to show that if $s \in L(r^{\{..n\}})$ then also $s \in L(r^{\{n\}})$ (under the assumption that r is nullable, otherwise it would not be true). The assumption $s \in L(r^{\{..n\}})$ means that $s \in L(r^{\{i\}})$ with $i \leq n$ holds and we have to show that $s \in L(r^{\{n\}})$ holds.
   One can do this by induction for languages as follows:

   $$\textit{if } [] \in A \textit{ and } s \in A^n \textit{ then } s \in A^{n+m}$$

   The proof is by induction on m. The base case $m = 0$ is trivial. For the $m + 1$ case we have the induction hypothesis:

   $$\textit{if } [] \in A \textit{ and } s \in A^n \textit{ then } s \in A^{n+m}$$

   and we have to show

   $$s \in A^{n+m+1}$$

   under the assumption $[] \in A$ and $s \in A^n$. From the assumptions and the IH we can infer that $s \in A^{n+m}$. Then using the assumption $[] \in A$ we can infer that also

   $$s \in A \,@\, A^{n+m}$$

   which is equivalent to what we need to show $s \in A^{n+m+1}$.
   Now we know $s \in L(r^{\{i\}})$ with $i \leq n$. Since $i + m = n$ for some m we can conclude that $s \in L(r^{\{n\}})$. Done.

8. Given the regular expression $r = (a \cdot b + b)^*$. Compute what the derivative of r is with respect to a, b and c. Is r nullable?

9. Define what is meant by the derivative of a regular expressions with respect to a character. (Hint: The derivative is defined recursively.)

   The recursive function for der.

10. Assume the set *Der* is defined as

$$Der\,c\,A \stackrel{\text{def}}{=} \{s \mid c :: s \in A\}$$

What is the relation between *Der* and the notion of derivative of regular expressions?

Main property is $L(der\,c\,r) = Der\,c\,(L(r))$.

11. Give a regular expression over the alphabet $\{a, b\}$ recognising all strings that do not contain any substring bb and end in a.

$((ba)^* \cdot (a)^*)^* \cdot a + ba$ Not sure whether this can be simplified.

12. Do $(a+b)^* \cdot b^+$ and $(a^* \cdot b^+) + (b^* \cdot b^+)$ define the same language?

No, the first one can match for example ababababababbbbb while the second can only match for example aaaaaabbbbb or bbbbbbb

13. Define the function *zeroable* by recursion over regular expressions. This function should satisfy the property

$$zeroable(r) \;\text{ if and only if }\; L(r) = \{\} \qquad (*)$$

The function *nullable* for the not-regular expressions can be defined by

$$nullable(\sim r) \stackrel{\text{def}}{=} \neg(nullable(r))$$

Unfortunately, a similar definition for *zeroable* does not satisfy the property in $(*)$:

$$zeroable(\sim r) \stackrel{\text{def}}{=} \neg(zeroable(r))$$

Find a counter example?

Here the idea is that nullable for NOT can be defined as

$$nullable(\sim r) \stackrel{\text{def}}{=} \neg(nullable(r))$$

This will satisfy the property $nullable(r)$ if and only if $[] \in L(r)$. (Remember how $L(\sim r)$ is defined).

But you cannot define

$$zeroable(\sim r) \stackrel{\text{def}}{=} \neg(zeroable(r))$$

because if r for example is $\mathbf{1}$ then $\sim \mathbf{1}$ can match some strings (all non-empty strings). So *zeroable* should be false. But if we follow the above definition we would obtain $\neg(zeroable(\mathbf{1}))$. According to the definition of *zeroable* for $\mathbf{1}$ this would be false, but if we now negate false, we get actually true. So the above definition would not satisfy the property

$$zeroable(r) \text{ if and only if } L(r) = \{\}$$

14. Give a regular expressions that can recognise all strings from the language $\{a^n \mid \exists k.\ n = 3k+1\}$.

    $a(aaa)^*$

15. Give a regular expression that can recognise an odd number of as or an even number of bs.

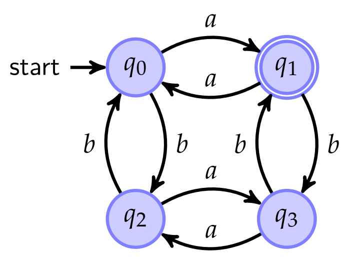
    If the a's and b's are meant to be separate, then this is easy

    $$a(aa)^* + (bb)^*$$

    If the letters are mixed, then this is difficult

    $$(aa|bb|(ab|ba)\cdot(aa|bb)^*\cdot(ba|ab))^*\cdot(b|(ab|ba)(bb|aa)^*\cdot a)$$

    (copied from somewhere ;o)
    The idea behind this monstrous regex is essentially the DFA



    Maybe a good idea to reconsider this example in Lecture 3 where the Brzozowski algorithm for DFA $\rightarrow$ Regex can be used.

16. **(Optional)** This question is for you to provide regular feedback to me: for example what were the most interesting, least interesting, or confusing parts in this lecture? Any problems with my Scala code? Please feel free to share any other questions or concerns. Also, all my material is ~~crap~~ imperfect. If you have any suggestions for improvement, I am very grateful to hear.

    If *you* want to share anything (code, videos, links), you are encouraged to do so. Just drop me an email or send a message to the Forum.