Homework 6

1. (i) Give the regular expressions for lexing a language consisting of whitespaces, identifiers (some letters followed by digits), numbers, operations `=`, `<` and `>`, and the keywords `if`, `then` and `else`. (ii) Decide whether the following strings can be lexed in this language?

   (a) `"if y4 = 3 then 1 else 3"`

   (b) `"if33 ifif then then23 else else 32"`

   (c) `"if x4x < 33 then 1 else 3"`

   In case they can, give the corresponding token sequences. (Hint: Observe the maximal munch rule and priorities of your regular expressions that make the process of lexing unambiguous.)

2. Suppose the grammar

$$\begin{aligned} \boldsymbol{E} &::= \boldsymbol{F} \;\mid\; \boldsymbol{F}\cdot * \cdot \boldsymbol{F} \;\mid\; \boldsymbol{F}\cdot \backslash \cdot \boldsymbol{F} \\ \boldsymbol{F} &::= \boldsymbol{T} \;\mid\; \boldsymbol{T}\cdot + \cdot \boldsymbol{T} \;\mid\; \boldsymbol{T}\cdot - \cdot \boldsymbol{T} \\ \boldsymbol{T} &::= \text{num} \;\mid\; (\cdot E \cdot) \end{aligned}$$

   where ***E***, ***F*** and ***T*** are non-terminals, ***E*** is the starting symbol of the grammar, and *num* stands for a number token. Give a parse tree for the string `(3+3)+(2*3)`. Note that ***F*** and ***T*** are "exchanged" in this grammar in comparison with the usual grammar for arithmetic expressions. What does this mean in terms of precedences of the operators?

   For the second part "1+2*3" will be parsed as (1+2)*3, meaning + and - bind tighter than * and \

3. Define what it means for a grammar to be ambiguous. Give an example of an ambiguous grammar.

   Already the grammar

$$\boldsymbol{E} ::= \boldsymbol{E} \cdot + \cdot \boldsymbol{E} \mid num$$

   is ambiguous because a string like "1+2+3" can be parsed as (1+2)+3 or 1+(2+3).

4. Suppose boolean expressions are built up from

   1.) tokens for `true` and `false`,
   2.) the infix operations $\wedge$ and $\vee$,
   3.) the prefix operation $\neg$, and
   4.) can be enclosed in parentheses.

(i) Give a grammar that can recognise such boolean expressions and (ii) give a sample string involving all rules given in 1.-4. that can be parsed by this grammar.

5. Parsing combinators consist of atomic parsers, alternative parsers, sequence parsers and semantic actions. What is the purpose of (1) atomic parsers and of (2) map-parsers (also called semantic actions)?

   Atomic parsers look at a concrete prefix of the input, like num-tokens or identifiers. Map-parsers apply a function to the output of a parser. In this way you can transform the output from, for example, a string to an integer.

6. Parser combinators can directly be given a string as input, without the need of a lexer. What are the advantages to first lex a string and then feed a sequence of tokens as input to the parser?

   Reason 1 you can filter out whitespaces and comments, which makes the grammar rules simpler. If you have to make sure that a whitespace comes after a variable say, then your parser rule for variables gets more complicated. Same with comments which do not contribute anything to the parse tree.

   Reason 2) The lexer can already classify tokens, for example as numbers, keywords or identifiers. This again makes the grammar rules more deterministic and as a result faster to parse.

   The point is that parser combinators can be used to process strings, but in case of compilers where whitespaces and comments need to be filtered out, the lexing phase is actually quite useful.

7. The injection function for sequence regular expressions is defined by three clauses:

$$\begin{array}{ll}
inj\,(r_1 \cdot r_2)\,c\;\; Seq(v_1, v_2) & \stackrel{\text{def}}{=} Seq(inj\,r_1\,c\,v_1, v_2)\\
inj\,(r_1 \cdot r_2)\,c\;\; Left(Seq(v_1, v_2)) & \stackrel{\text{def}}{=} Seq(inj\,r_1\,c\,v_1, v_2)\\
inj\,(r_1 \cdot r_2)\,c\;\; Right(v) & \stackrel{\text{def}}{=} Seq(mkeps(r_1), inj\,r_2\,c\,v)
\end{array}$$

   Explain why there are three cases in the injection function for sequence regular expressions.

   This is because the derivative of sequences can be of the form

   - $(der\,c\,r_1) \cdot r_2$
   - $(der\,c\,r_1) \cdot r_2 \;+\; der\,c\,r_2$

   In the first case the value needs to be of the form Seq, in the second case *Left* or $Right$. Therefore 3 cases.

8. **(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.