Homework 9

Please submit your solutions via email. Please submit only PDFs! Every solution should be preceeded by the corresponding question text, like:

Qn:	**…a difficult question from me…**
A:	**…an answer from you …**
Q$n+1$	**…another difficult question…**
A:	**…another brilliant answer from you…**

Solutions will only be accepted until 20th December! Please send only one homework per email.

1. Describe what is meant by *eliminating tail recursion*? When can this optimization be applied and why is it of benefit?

2. A programming language has arithmetic expression. For an arithmetic expression the compiler of this language produces the following snippet of JVM code.

   ```
   ldc 1
   ldc 2
   ldc 3
   imul
   ldc 4
   ldc 3
   isub
   iadd
   iadd
   ```

   Give the arithmetic expression that produced this code. Make sure you give all necessary parentheses.

3. Describe what the following JVM instructions do!

   ```
   ldc 3
   iload 3
   istore 1
   ifeq label
   if_icmpge label
   ```

4. What does the following JVM function calculate?

```
.method public static bar(I)I
.limit locals 1
.limit stack 9
   iload 0
   ldc 0
   if_icmpne If_else_8
   ldc 0
   goto If_end_9
If_else_8:
   iload 0
   ldc 1
   if_icmpne If_else_10
   ldc 1
   goto If_end_11
If_else_10:
   iload 0
   ldc 1
   isub
   invokestatic bar(I)I
   iload 0
   ldc 2
   isub
   invokestatic bar(I)I
   iadd
If_end_11:
If_end_9:
   ireturn
.end method
```

5. What does the following LLVM function calculate? Give the corresponding arithmetic expression .

```
define i32 @foo(i32 %a, i32 %b)  {
  %1 = mul i32 %a, %a
  %2 = mul i32 %a, 2
  %3 = mul i32 %2, %b
  %4 = add i32 %1, %3
  %5 = mul i32 %b, %b
  %6 = add i32 %5, %4
  ret i32 %6
}
```

6. As an optimisation technique, a compiler might want to detect ‘dead code’ and not generate anything for this code. Why does this optimisation technique have the potential of speeding up the run-time of a program? (Hint:

On what CPUs are programs run nowadays?)

7. In an earlier question, we analysed the advantages of having a lexer-phase before running the parser (having a lexer is definitely a good thing to have). But you might wonder if a lexer can also be implemented by a parser and some simple grammar rules. Consider for example:

$$\begin{aligned}
\boldsymbol{S} &::= (\boldsymbol{Kw} \mid \boldsymbol{Id} \mid \boldsymbol{Ws}) \cdot \boldsymbol{S} \;\mid\; \epsilon \\
\boldsymbol{Kw} &::= \texttt{if} \mid \texttt{then} \mid \ldots \\
\boldsymbol{Id} &::= (\texttt{a} \mid \ldots \mid \texttt{z}) \cdot \boldsymbol{Id} \;\mid\; \epsilon \\
\boldsymbol{Ws} &::= \ldots
\end{aligned}$$

   What is wrong with implementing a lexer in this way?

8. What is the difference between a parse tree and an abstract syntax tree? Give some simple examples for each of them.

9. Give a description of how the Brzozowski matcher works. The description should be coherent and logical.

10. Give a description of how a compiler for the While-language can be implemented. You should assume you are producing code for the JVM. The description should be coherent and logical.

11. **(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.