Deadline: 20.01.2014

Submit either at the exercise session on 20.01.2014 or in box 137 outside the toilets of Kern D.

Anonymous questionnaire about exercises
of the lecture "Programming languages and types"

- DO NOT write your name or student number (Matrikelnummer) on this page.
- DO NOT staple this page together with other pages of your homework submission.

The questionnaire is for statistics only and is in no way connected to the evaluation of your homework submission. By filling out this questionnaire, you declare to have answered as truthfully as your knowledge permits.

1. *Outside* the exercise session, how many hours did you spend doing the homework?

   _____ hours.

2. In your opinion, how many hours would it take an average student to do the homework?

   _____ hours.

3. Did you attend the exercise session on 13.01.2014?

   ☐ Yes.
   ☐ No.

4. Please cross *all* statements that are true about you.

   ☐ I can translate between abstract grammars and case classes.
   ☐ I can read reduction rules.
   ☐ I can do reduction by hand.
   ☐ I know how to construct derivation trees.
   ☐ I know how to do a proof by induction on syntax.

Deadline: 20.01.2014

Submit either at the exercise session on 20.01.2014 or in box 137 outside the toilets of Kern D.

Matrikelnummer:	
Name:	

Homework 10 of the lecture "Programming languages and types"

1. Abstract grammars are another way of looking at the abstract syntax trees in Scala that we have been writing up to now. `02-ae.scala` defines the following abstract syntax trees:

```
sealed abstract class Exp
case class Num(n: Int) extends Exp
case class Add(lhs: Exp, rhs: Exp) extends Exp
case class Mul(lhs: Exp, rhs: Exp) extends Exp
case class Id(x: Symbol) extends Exp
```

   This abstract grammar expresses the very same idea:

$$
\begin{aligned}
exp ::= \; & n \\
| \; & exp \texttt{ + } exp \\
| \; & exp \texttt{ * } exp \\
| \; & x
\end{aligned}
$$

   where n is an integer and x is a symbol.

(a) Convert the following abstract expression into a Scala value definable in `02-ae.scala`.

```
(x * 2) + (y + y)
```

(b) This is the abstract grammar in the slides `22-sos.pdf`. Convert it to a series of case class definitions.

t ::= `true`
| `false`
| `if` t `then` t `else` t
| `0`
| `succ` t
| `pred` t
| `iszero` t

2. The small grammar below describes a language of Boolean expressions.

$$
\begin{array}{rl}
t ::= & \texttt{true} \\
\mid & \texttt{false} \\
\mid & \texttt{if } t \texttt{ then } t \texttt{ else } t
\end{array}
$$

(a) These are reduction rules from the slides `22-sos.pdf`.

$$\texttt{if true then } t_2 \texttt{ else } t_3 \longrightarrow t_2 \qquad \text{(E-IfTrue)}$$

$$\texttt{if false then } t_2 \texttt{ else } t_3 \longrightarrow t_3 \qquad \text{(E-IfFalse)}$$

$$\frac{t_1 \longrightarrow t_1'}{\texttt{if } t_1 \texttt{ then } t_2 \texttt{ else } t_3 \longrightarrow \texttt{if } t_1' \texttt{ then } t_2 \texttt{ else } t_3} \qquad \text{(E-If)}$$

Carry out a sequence of reductions on the Boolean expression t until it reaches normal form.

$$
\begin{array}{rl}
t = & \texttt{if} \quad (\texttt{if true then true else false}) \\
& \texttt{then } (\texttt{if false then false else true}) \\
& \texttt{else false}
\end{array}
$$

(b) Suppose we define another reduction relation $\Longrightarrow$ by these rules:

$$\frac{t_1 \Longrightarrow \texttt{true} \qquad t_2 \Longrightarrow t_2'}{\texttt{if } t_1 \texttt{ then } t_2 \texttt{ else } t_3 \Longrightarrow t_2'} \qquad \text{(N-IfTrue)}$$

$$\frac{t_1 \Longrightarrow \texttt{false} \qquad t_3 \Longrightarrow t_3'}{\texttt{if false then } t_2 \texttt{ else } t_3 \Longrightarrow t_3'} \qquad \text{(N-IfFalse)}$$

Carry out reduction on the Boolean expression t. Can you structure your solution so that it looks like a tree?

$t =$ `if` (`if true then true else false`)
`then` (`if false then false else true`)
`else false`

3. (OPTIONAL: This problem is not required, but solving it *will* make you a better person.)

Consider the Boolean expressions again.

$$\begin{aligned} t ::= & \texttt{true} \\ & \mid \texttt{false} \\ & \mid \texttt{if } t \texttt{ then } t \texttt{ else } t \end{aligned}$$

Define functions *size* and *depth* inductively thus:

$$\begin{aligned} \mathit{size}(\texttt{true}) &= 1 \\ \mathit{size}(\texttt{false}) &= 1 \\ \mathit{size}(\texttt{if } t_1 \texttt{ then } t_2 \texttt{ else } t_3) &= \mathit{size}(t_1) + \mathit{size}(t_2) + \mathit{size}(t_3) + 1 \end{aligned}$$

$$\begin{aligned} \mathit{depth}(\texttt{true}) &= 1 \\ \mathit{depth}(\texttt{false}) &= 1 \\ \mathit{depth}(\texttt{if } t_1 \texttt{ then } t_2 \texttt{ else } t_3) &= \max(\mathit{depth}(t_1), \mathit{depth}(t_2), \mathit{depth}(t_3)) + 1 \end{aligned}$$

(a) Calculate $\mathit{size}(t)$ and $\mathit{depth}(t)$ for

$$\begin{aligned} t = \texttt{if} \quad & (\texttt{if true then true else false}) \\ \texttt{then} \ & (\texttt{if false then false else true}) \\ \texttt{else} \ & \texttt{false} \end{aligned}$$

(b) Let t be any Boolean expression. Prove by induction on t that

$$\mathit{depth}(t) \leq \mathit{size}(t).$$

(c) Let t be any Boolean expression. Prove by induction on t that

$$\mathit{size}(t) \leq \frac{3^{\mathit{depth}(t)} - 1}{2}.$$