Philipps Universität Marburg

Deadline: 27.01.2014

Submit either at the exercise session on 27.01.2014 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. How many lecture sessions did you attend this week?

   ☐ None. ☐ One session. ☐ Two sessions.

4. Did you attend the exercise session on 20.01.2014?

   ☐ Yes. ☐ No.

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

   ☐ I can evaluate lambda terms by hand.
   ☐ I understand the significance of congruence rules.
   ☐ I can construct typing derivations.
   ☐ I have tried to write a type checker in Scala.

Philipps Universität Marburg

Deadline: 27.01.2014

Submit either at the exercise session on 27.01.2014 or in box 137 outside the toilets of Kern D.

Matrikelnummer:	
Name:	

Homework 11 of the lecture "Programming languages and types"

1. Recall the syntax of lambda calculus on slides 5 and 6 of `24-stlc.pdf`.

$$\begin{aligned} t &::= x \ \mid\ \lambda x.\ t \ \mid\ t\ t \\ v &::= \lambda x.\ t \end{aligned}$$

   (a) Recall the evaluation rules on slide 8 of `24-stlc.pdf`. The rule E-AppAbs is called a *computation rule*; the rules E-App1 and E-App2 are called *congruence rules*.

$$(\lambda x.\ t_{12})\ v_2 \quad\longrightarrow\quad [x \mapsto v_2]t_{12} \qquad \text{(E-AppAbs)}$$

$$\frac{t_1 \quad\longrightarrow\quad t_1'}{t_1\ t_2 \quad\longrightarrow\quad t_1'\ t_2} \qquad \text{(E-App1)}$$

$$\frac{t_2 \quad\longrightarrow\quad t_2'}{v_1\ t_2 \quad\longrightarrow\quad v_1\ t_2'} \qquad \text{(E-App2)}$$

Evaluate the following terms. You may write a chain of reductions without naming the rules; just be careful not to use E-App2 where it does not apply.

$$(\lambda x.\ x\ x)\ ((\lambda y.\ y)\ (\lambda z.\ z)) \quad \longrightarrow$$

$$(\lambda w.\ \lambda x.\ x)\ ((\lambda y.\ y\ y)\ (\lambda z.\ z\ z)) \quad \longrightarrow$$

(b) Replace the evaluation rules of part (a) by the following. (N-AppAbs) is the computation rule, and (N-App) is the congruence rule.

$$(\lambda x.\ t_{12})\ t_2 \quad \longrightarrow \quad [x \mapsto t_2]t_{12} \qquad \text{(N-AppAbs)}$$

$$\frac{t_1 \quad \longrightarrow \quad t_1'}{t_1\ t_2 \quad \longrightarrow \quad t_1'\ t_2} \qquad \text{(N-App)}$$

Carry out the reductions again.

$$(\lambda x.\ x\ x)\ ((\lambda y.\ y)\ (\lambda z.\ z)) \quad \longrightarrow$$

$$(\lambda w.\ \lambda x.\ x)\ ((\lambda y.\ y\ y)\ (\lambda z.\ z\ z)) \quad \longrightarrow$$

(c) Do you notice any difference between the reductions in part (a) and the reductions in part (b)? What is the significance of congruence rules?

2. Below is a modified version of the grammar and typing rules of simply typed lambda calculus.

$$\begin{aligned} t &::= x \;\mid\; \lambda x : T.\ t \;\mid\; t\ t \\ T &::= T \to T \;\mid\; A \;\mid\; B \;\mid\; C \;\mid\; D \;\mid\; \cdots \end{aligned}$$

$$\frac{x : T \in \Gamma}{\Gamma \vdash x : T} \qquad \text{(T-VAR)}$$

$$\frac{\Gamma, x : T_1 \vdash t_2 : T_2}{\Gamma \vdash (\lambda x : T_1.\ t_2) : T_1 \to T_2} \qquad \text{(T-ABS)}$$

$$\frac{\Gamma \vdash t_1 : T_{11} \to T_{12} \qquad \Gamma \vdash t_2 : T_{11}}{\Gamma \vdash t_1\ t_2 : T_{12}} \qquad \text{(T-APP)}$$

In this version of lambda calculus, we know there are types A, B, C, D, et cetera, but we don't any constant of type A or B or C or D. Nevertheless, we can write meaningful terms. For example,

$$t_0 = \lambda x : A.\ x$$

It is possible to derive, from an empty context, the typing judgement

$$\vdash t_0 : A \to A.$$

We don't know what values of type A there are, but we can be sure that given any value $v : A$, the result $(t_0\ v)$ evaluates to v itself. In other words, t_0 expresses the identity function between values of type A.

(a) For each i, come up with a term t_i with the type specified below.

$$\vdash t_1 : A \to B \to A$$
$$t_1 =$$

$\vdash t_2 : A \to (A \to B) \to B$

$t_2 =$

$\vdash t_3 : (A \to B \to C) \to B \to A \to C$

$t_3 =$

$\vdash t_4 : (A \to B) \to (B \to C) \to A \to C$

$t_4 =$

$\vdash t_5 : (A \to B \to C) \to (A \to B) \to A \to C$

$t_5 =$

(b) Pick *one* t_i with $1 \leq i \leq 5$. Construct a typing derivation for t_i.

3. (Optional programming exercise. You may either email the code to

   `pllecture@informatik.uni-marburg.de`,

   attach a print-out here, or choose not to do the exercise.)

   This is a translation of the grammar of lambda terms, Booleans, pairs and tagged values. Design and implement a type checker in Scala based on the typing rules given in `24-stlc.pdf`.

```
sealed trait Type

// type of lambda abstractions
case class Fun(domain: Type, range: Type) extends Type

// type of Booleans
case object Bool extends Type

// type of pairs
case class Product(type1: Type, type2: Type) extends Type

// type of tagged values
case class Sum(type1: Type, type2: Type) extends Type

sealed trait Term

// lambda terms
case class Var(x: Symbol) extends Term
case class Abs(x: Symbol, xtype: Type, t: Term) extends Term
case class App(t1: Term, t2: Term) extends Term

// Boolean constants
case object True extends Term
case object False extends Term

// pairs
case class Pair(x: Term, y: Term) extends Term
case class Project1(pair: Term) extends Term
case class Project2(pair: Term) extends Term

// tagged values
// inl x as sumType
case class Inl(x: Term, sumType: Sum) extends Term
// inr y as sumType
case class Inr(y: Term, sumType: Sum) extends Term
// case t of
//     inl x  =>  case1
//   | inr y  =>  case2
case class Case(
  t: Term,
  x: Symbol, case1: Term,
  y: Symbol, case2: Term
) extends Term
```