Assignment 7
Semantics, WS 2007/08

Prof. Dr. Gert Smolka, Dr. Jan Schwinghammer
www.ps.uni-sb.de/courses/sem-ws07/

Read Chapter 4 of the lecture notes and Chapters 9 and 11 of Pierce's TAPL

Exercise 7.1 (Syntax, typing and semantics of simply typed lambda calculus) Make sure that you understand the definitions and the basic properties of the simply typed λ-calculus. In particular, you should be able to define or state the following:

a) Types and terms.

b) The typing relation.

c) Inversion.

d) Unique Types, Coincidence, and Relevance.

e) Replacement, Substitution, and Compatibility with β and α.

f) Call-by-value and full reduction.

g) Preservation and Progress.

Exercise 7.2 (Bool) Show how the type *Bool* can be expressed with sums and unit:

$$\begin{aligned} \mathit{Bool} &:= \\ \mathit{false} &:= \\ \mathit{true} &:= \\ \text{if } t \text{ then } t_1 \text{ else } t_2 &:= \end{aligned}$$

Exercise 7.3 (Natural Deduction) The types of ND can be seen as Boolean formulas, where X is a Boolean variable, $T_1 \to T_2$ is an implication, $T_1 \times T_2$ is a conjunction, $T_1 + T_2$ is a disjunction, and 0 and 1 are false and true. A term t is called a *proof for a formula* T iff $\emptyset \vdash t : T$. One can show that a formula has a proof if and only if it is valid. We use the abbreviation $\overline{X} := X \to 0$. Find proofs for the following formulas:

a) $((X \to Y) \times X) \to Y$

b) $(X + Y) \to \overline{\overline{X} \times \overline{Y}}$

c) $(X \times Y) \to \overline{\overline{X} + \overline{Y}}$

d) $\overline{X \times \overline{X}}$

e) $\overline{X + Y} \to (\overline{X} \times \overline{Y})$

f) $(\overline{X} \times \overline{Y}) \to \overline{X + Y}$

g) $0 \to X$

h) $(\overline{X} \to \overline{Y}) \to (Y \to X)$

Exercise 7.4 (Peirce's Law) Peirce's Law is the Boolean formula

$$((X \to Y) \to X) \to X$$

This formula is valid. Hence it can be proven in ND. One can show that every proof for Peirce's law in ND must involve a subterm formed with δ. This is somewhat surprising since Peirce's law just employs implication while δ must be used with terms whose type involves 0.

a) Find a proof for Peirce's Law in ND.

b) Read the Wikipedia entry for Peirce's law.

Exercise 7.5 (Fix) If we extend ND with a recursion operator fix with the usual typing rule

$$\frac{\Gamma \vdash t : T \to T}{\Gamma \vdash \text{fix } t : T}$$

we can prove everything.

a) Let T be a type. Find a proof for T in ND extended with fix.

b) Let T be a type. Find a proof for T in ND extended with fix that applies fix only to terms of the form $\lambda x_1{:}T_1.\lambda x_2{:}T_2.t$.

Exercise 7.6 (Proof Checker) A SML interpreter provides a proof checker for ND. Type variables, procedure types, products and 1 (unit) are built in. Sum types can be obtained with

```
datatype ('a,'b) sum = L of 'a | R of 'b
```

The type 0 can be realized as follows:

```
datatype null = N of null
val delta : (('a -> null) -> null) -> 'a = fn _ => raise Match
```

Now we have a proof checker for ND. First we try a proof for $0 \to X$:

```
fn n:null => delta (fn f:'a->null => n)
```

fn : null → α

Since SML has type reconstruction, proofs can be written without type annotations:

```
fn n => delta (fn _ => n)
```
fn : null → α

If we bind the proof to an identifier p

```
val p = fn n => delta (fn _ => n)
```
val p : null → α

we obtain a polymorphic proof of $0 \to T$ for all types T. Here is a proof for $((X + Y) \times (\overline{X} + Z)) \to (Y + Z)$ that exploits SML's pattern matching and the polymorphic proof p:

```
fn (R y, _) => L y
 | (_, R z) => R z
 | (L x, L f) => p (f x)
```
*fn : (α, β) sum * (α → null, γ) sum → (β, γ) sum*

Write your proofs for Exercises 7.3 and 7.4 in SML and check them with an interpreter.