Semantics, WS 2005 – Assignment 5

Prof. Dr. Gert Smolka, Dipl.-Inform. Andreas Rossberg
http://www.ps.uni-sb.de/courses/sem-ws05/

Recommended reading: Types and Programming Languages, Chapter 11

For the Curry-Howard Correspondence we consider a simply typed lambda calculus ND with products and sums whose abstract syntax and typing relation are defined as follows:

$$
\begin{aligned}
X &\in TVar \\
T &\in Ty \;=\; X \mid T \to T \mid T \times T \mid T + T \mid 0 \mid 1 \\
x &\in Var \\
i &\in \{1, 2\} \\
t &\in Ter \;=\; x \mid \lambda x{:}T.t \mid t\;t \mid (t, t) \mid t.i \mid (i, t) \text{ as } T \mid \text{case } t\;t\;t \mid \delta\;t \mid () \\
\Gamma &\in TE \;=\; Var \rightharpoonup Ty
\end{aligned}
$$

$$
\frac{\Gamma x = T}{\Gamma \vdash x : T}
\qquad
\frac{\Gamma[x := T] \vdash t : T'}{\Gamma \vdash \lambda x{:}T.t : T \to T'}
\qquad
\frac{\Gamma \vdash t_1 : T \to T' \qquad \Gamma \vdash t_2 : T}{\Gamma \vdash t_1\, t_2 : T'}
$$

$$
\frac{\Gamma \vdash t_1 : T_1 \qquad \Gamma \vdash t_2 : T_2}{\Gamma \vdash (t_1, t_2) : T_1 \times T_2}
\qquad
\frac{\Gamma \vdash t : T_1 \times T_2}{\Gamma \vdash t.i : T_i}
\qquad
\frac{\Gamma \vdash t : T_i \qquad T = T_1 + T_2}{\Gamma \vdash (i, t) \text{ as } T : T}
$$

$$
\frac{\Gamma \vdash t : T_1 + T_2 \qquad \Gamma \vdash t_1 : T_1 \to T \qquad \Gamma \vdash t_2 : T_2 \to T}{\Gamma \vdash \text{case } t\; t_1\; t_2 : T}
$$

$$
\frac{\Gamma \vdash t : (T \to 0) \to 0}{\delta\; t : T}
\qquad
\frac{}{\Gamma \vdash () : 1}
$$

The reduction relation of ND is defined by means of the proper reduction rules

$$
(\lambda x{:}T.t)t' \to t[x := t'] \qquad (t_1, t_2).i \to t_i \qquad \text{case } ((i, t) \text{ as } T)\; t_1\; t_2 \to t_i\; t
$$

that can be applied to any subterm. Note that this means that the rules can be applied in any order and also within abstractions.

Exercise 5.1: Nondeterminism Find a term t such that there exist exactly three terms t' such that $t \to t'$.

Exercise 5.2: Properties

(a) State the preservation property for ND.

(b) State the confluence property for ND.

(c) State the termination property for ND.

Exercise 5.3: Reduction Contexts Define the reduction contexts for

(a) ND as defined above.

(b) SL as defined in Assignment 4.

Exercise 5.4: Reduction Discipline For this exercise we consider a version of ND where reduction is disallowed for the following subterms: (i) t in $\lambda x : T.t$; (ii) t_1 and t_2 in case $t\ t_1\ t_2$; (iii) t in $\delta\ t$.

(a) Define the reduction contexts that formalize this reduction discipline.

(b) State the descent rules that formalize this reduction discipline.

Exercise 5.5: Deterministic Reduction For this exercise we consider a computational variant of ND obtained by deleting the syntactic form $\delta\ t$ and by restricting the reduction discipline to be deterministic, call-by-value and left-to-right.

(a) Define the values for this language.

(b) State the proper reduction rules for this language.

(c) State the reduction contexts for this language.

(d) Define the evaluation relation $t \Downarrow v$ by means of inference rules (big-step semantics).

Exercise 5.6: Bool Show how the type *Bool* can be expressed with sums and unit:

$$\begin{aligned} \mathit{Bool} &\overset{\text{def}}{=} 1 + 1 \\ \mathit{false} &\overset{\text{def}}{=} \\ \mathit{true} &\overset{\text{def}}{=} \\ \text{if } t \text{ then } t_1 \text{ else } t_2 &\overset{\text{def}}{=} \end{aligned}$$

Exercise 5.7: 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 0 and 1. 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 $\overline{X} \stackrel{\text{def}}{=} X \to 0$. 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 5.8: 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 http://en.wikipedia.org/wiki/Peirce's_law.

Exercise 5.9: 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 5.10: 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
```

and 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 Exercise 5.7 in SML and check them with an interpreter.