Exercices on Typed Lambda-Calculus

Thierry Coquand
Chalmers University

October 1998

Exercice 1: We can define like in untyped lambda calculus the notion of term in normal form. Show that a term in normal form can be written $\lambda x_1 \ldots \lambda x_k\ x\ M_1 \ldots M_l$ where we can have $k = 0$ or $l = 0$ and $M_1, \ldots, M_l$ are in normal form.

Use this to enumerate the closed terms of the following types (ι is a ground type)

1. $\iota \to \iota$,
2. $\iota \to \iota \to \iota$,
3. $(\iota \to \iota) \to \iota \to \iota$,
4. $\iota \to (\iota \to \iota) \to \iota$,
5. $(\iota \to \iota) \to \iota$.
6. $((\iota \to \iota) \to \iota) \to \iota$.

Exercice 2: Cantor's theorem can be expressed as follows in higher-order logic: there is no surjective function $H : \iota \to (\iota \to \mathbf{o})$. It is enough for this to deduce a contradiction from the existence of a spliting function $A : (\iota \to \mathbf{o}) \to \iota$, that is a function which satisfies

$$(\forall f)(\forall x)\ [f\ x \iff H\ (A\ f)\ x].$$

For this, it is enough to find a function $f : \iota \to \mathbf{o}$ and an individual $x : \iota$ such that

$$f\ x = \neg\ (H\ (A\ f)\ x).$$

A typical unification problem is to find a suitable f and x. (Cantor's theorem has been proved automatically in Andrews' implementation of Church's higher-order logic.) For solving this unification problem, enumerate all terms of type ι and of type $\iota \to \mathbf{o}$ that are built from the constant

- $\neg : \mathbf{o} \to \mathbf{o}$,
- $H : \iota \to (\iota \to \mathbf{o})$,
- $A : (\iota \to \mathbf{o}) \to \iota$.

These two sets of terms are best described by a mutual inductive definition.

Exercice 3: Show that there exists no term $t : B{\to}B$ which may use the constant $0, 1 : B$ such that $t\ 0 = 1$ and $t\ 1 = 0$. (Hint: if such a term exists, we would have by the fundamental theorem of logical relation

$$(R)[R(0,0){\to}R(1,1){\to}R(0,1){\to}R(t\ 0, t\ 1)].)$$

Exercice 4: Show that the only uniform family $t_X : X^X$ is the identity function family. Show that the only uniform families of type $t_X : (X^X)^X$ are the two projection functions: $\lambda x \lambda y\ x$ and $\lambda x \lambda y\ y$. Show that the only uniform families $t_X : (X \times X)^{X \times X}$ are the identity and the maps $(u, v) \longmapsto (v, u)$, $(u, v) \longmapsto (u, u)$ and $(u, v) \longmapsto (v, v)$.

Exercice 5. (Manna) Let $G : \sigma{\to}\sigma$ and $P : \sigma{\to}\mathsf{B}$ be arbitrary functions, and $F : \tau{\to}\tau$ a strict function. Operationally this means that F uses its argument, and denotationally that we have $F\ \perp=\perp$. We define $H = Y\ K$, where

$$K = \lambda h, x, y \text{ if } P\ x \text{ then } y \text{ else } F\ (h\ (Gx)\ y);$$

in this way, H satisfies the recursive equation

$$H\ x\ y\ =\ \text{if } P\ x \text{ then } y \text{ else } F\ (H\ (Gx)\ y).$$

Show $F\ (H\ x\ y) = H\ x\ (F\ y)$ for all x, y via denotational semantics: H being the limit of $H_0 = \lambda x \lambda y\ \perp$, $H_1 = K\ H_0$, $H_2 = K\ H_1, \ldots$ show that $F\ (H_n\ x\ y) = H_n\ x\ (F\ y)$ implies $F\ (H_{n+1}\ x\ y) = H_{n+1}\ x\ (F\ y)$. It follows that the programs $\lambda x, y\ F\ (H\ x\ y)$ and $\lambda x, y\ H\ x\ (F\ y)$ are operationally equivalent by the adequacy theorem.

Exercice 6: In Martin-Löf type theory, we can define $A = (\Pi X \in U)\ (X{\to}X){\to}(X{\to}X)$ and represent natural numbers at this type. Give a term of type $A{\to}A$ that represents the exponential function $n \longmapsto 2^n$.

1 Complementary Exercices

Exercice 1: This problem illustrates the importance of explicitely typing the variable. We choose here for simply typed lambda calculus that any variable comes with its type. An alternative notation would be to use the same term as in untyped lamba-calculus (untyped variable). For instance $\lambda x\ x$ would be both of type $\iota{\to}\iota$ and $(\iota{\to}\iota){\to}\iota{\to}\iota$. One can consider the typed variable as a pair of one untyped variable and a type. It is then possible to define a map $t \longmapsto |t|$ that to a typed term associate its "stripped" version where we forget about type annotations. In the previous example, if we give both terms and types, it is possible to reconstruct in an unique way a typed term t such that $|t| = \lambda x\ x$. Show that this is general for any untyped term in normal form u that there is at most on typed term t of a given type such that $|t| = u$.

Show that this may not the case for an arbitrary term. However, show that if $|t_1| = |t_2|$ and t_1, t_2 are of the same type then t_1 and t_2 are convertible.

Show that this is always the case for a λI term, by showing that if u reduces by head reduction to u' and the u has a unique typed decoration then u has a unique typed decoration, and using the normalisation theorem for typed lambda calculus.

Exercice 2: We describe Stoughton's algorithm on a simple concrete instance. Suppose that we want to find a term t of type $(B{\to}B){\to}B$ where $B = \{0, 1\}$ such that

$$t\ d_1 = 0, \qquad t\ d_2 = 1$$

where $d_1\ x = \neg\ x$ and $d_2\ x = x$. Furthermore this term should use only as constant 0 and 1.

Notice that we can enumerate all terms of type $(B{\rightarrow}B){\rightarrow}B$: they are of the form $\lambda f\ f^k\ 0$ or $\lambda f\ f^k\ 1$. But to try them one after the other gives only a semi-decision procedure. It is quite remarkable that this question is actually decidable.

By the fundamental theorem of logical relations, *if* such a term exists we have $R(t\ d_1, t\ d_2) = R(0,1)$ for any relation R such that

- $R(0,0)$ and $R(1,1)$,
- $R(x,y){\rightarrow}R(d_1\ x, d_2\ y)$.

What is remarkable is that conversely, if we have

$$(R)[R(0,0){\rightarrow}R(1,1){\rightarrow}[(x)(y)R(x,y){\rightarrow}R(d_1\ x, d_2\ y)]{\rightarrow}R(0,1)] \qquad (*)$$

then there exists a term t such that $t\ d_1 = 0$ and $t\ d_2 = 1$. This follows by taking for R the relation

$$\lambda x \lambda y\ \exists t\ [t\ d_1 = x \wedge t\ d_2 = y].$$

So (*) is actually a necessary and sufficient condition for the existence of a solution. It does not seem a priori algorithmic, since (*) is expressed with an universal quantification over all relations. Remark however that we can list all such relations. Furthermore, there is even a clever reading of (*) that gives an elegant algorithm to test (*): read the relation

$$\lambda x \lambda y\ (R)[R(0,0){\rightarrow}R(1,1){\rightarrow}[(x)(y)R(x,y){\rightarrow}R(d_1\ x, d_2\ y)]{\rightarrow}R(x,y)]$$

as meaning that (x,y) is in the set of pairs generated by the rule

- ${\rightarrow}(0,0)$,
- ${\rightarrow}(1,1)$,
- $(x,y){\rightarrow}(d_1\ x, d_2\ y)$.

Check that $(0,1)$ is in this set, and uses this derivation of $(0,1)$ to build t.