Types and λ-calculus

Problem Sheet 5

The ***pure, untyped λ-calculus*** is the subset of PCF that does not include *any* constants. That is, *pure* terms are defined by the following grammar:

$$M, N ::= x \mid (M\,N) \mid (\lambda x.\, M)$$

In this problem sheet we are going to show that we can actually *define* a version of each PCF constant: Z, S, pred, ifz and fix, rather than considering them to be "baked-in" to the language. Each will be just an abbreviation for a *pure* λ-term.

The ***Church numeral*** for the number n, abbreviated $\ulcorner n \urcorner$, is:

$$\lambda f\,x.\, \underbrace{f\,(\cdots(f}_{n\text{-times}}\,x)\cdots)$$

In other words, the Church numeral for n is the pure term that takes a function f and an argument x and iterates f n-times on x. Let's list the first natural numbers, using Church numerals:

$$\begin{array}{rcl}
\ulcorner 0 \urcorner & = & \lambda f\,x.\,x \\
\ulcorner 1 \urcorner & = & \lambda f\,x.\,f\,x \\
\ulcorner 2 \urcorner & = & \lambda f\,x.\,f\,(f\,x) \\
\ulcorner 3 \urcorner & = & \lambda f\,x.\,f\,(f\,(f\,x)) \\
\ulcorner 4 \urcorner & = & \lambda f\,x.\,f\,(f\,(f\,(f\,x))) \\
 & \vdots &
\end{array}$$

We can define this formally (without the need to resort to $\cdots$), if we first define an auxilliary function $\mathsf{iter} : \mathsf{Var} \times \mathsf{Var} \to \mathbb{N} \to \Lambda$ which takes a pair of variables and a natural number and delivers a pure term.

$$\mathsf{iter}(f,\,x)(0) = x \tag{1}$$

$$\mathsf{iter}(f,\,x)(m+1) = f\ (\mathsf{iter}(f,\,x)(m)) \tag{2}$$

We have, for example, $\mathsf{iter}(f,\,x)(3) = f\ (f\ (f\ x))$. Then we can define Church Numerals by:

$$\ulcorner n \urcorner = \lambda f\,x.\,\mathsf{iter}(f,\,x)(n)$$

So that we get $\ulcorner 3 \urcorner = \lambda f\, x.\, \mathsf{iter}(f\ x)(3) = \lambda f\, x.\, f\ (f\ (f\ x))$, as expected.

We are going to replace our numerical constants by abbreviations using pure terms that manipulate Church numerals. To get started, let's define:

$$\begin{aligned}\underline{Z} &= \ulcorner 0 \urcorner \\ \underline{S} &= \lambda n.\, \lambda f\, x.\, f\ (n\, f\ x)\end{aligned}$$

** 1.

(a) Using induction on n, prove that, for all $n \in \mathbb{N}$, $\ulcorner n \urcorner\, g\, y \ \approx\ \mathsf{iter}(g, y)(n)$.

(b) Use this to show that, for all $n \in \mathbb{N}$, $\underline{S}\, \ulcorner n \urcorner \ \approx\ \ulcorner n + 1 \urcorner$.

Solution

(a) By induction on $n \in \mathbb{N}$.

- When $n = 0$, $\ulcorner 0 \urcorner\, g\, y = (\lambda f\, x.\, x)\, g\, y \ \rhd^*\ y = \mathsf{iter}(g, y)(0)$.
- When n is of shape $k + 1$, assume the induction hypothesis:

$$\ulcorner k \urcorner\, g\, y \ \approx\ \mathsf{iter}(g, y)(k)$$

Then reason equationally:

$$\begin{aligned}\ulcorner k + 1 \urcorner\, g\, y &= (\lambda f\, x.\, \mathsf{iter}(f, x)(k + 1))\, g\, y \\ &= (\lambda f\, x.\, f\ (\mathsf{iter}(f, x)(k)))\, g\, y \\ &\approx\ (\lambda f\, x.\, f\ (\ulcorner k \urcorner\, f\ x))\, g\, y \\ &\approx\ g\ (\ulcorner k \urcorner\, g\, y) \\ &\approx\ g\ (\mathsf{iter}(g, y)(k)) \\ &= \mathsf{iter}(g, y)(k + 1)\end{aligned}$$

Where we use the induction hypothesis twice: in different directions in the third and fifth lines.

(b) Let n be a natural number. We reason as follows:

$$\begin{aligned}\underline{S}\, \ulcorner n \urcorner &\approx\ \lambda f\, x.\, f\ (\ulcorner n \urcorner\, f\ x) \\ &\approx\ \lambda f\, x.\, f\ (\mathsf{iter}(f, x)(n)) \\ &= \lambda f\, x.\, \mathsf{iter}(f, x)(n + 1) \\ &= \ulcorner n + 1 \urcorner\end{aligned}$$

** 2. Construct a *pure* term $\underline{\mathsf{ifz}}$ satisfying, for all natural numbers $n \in \mathbb{N}$, all M, N:

$$\begin{aligned} \underline{\mathsf{ifz}}\ulcorner 0 \urcorner M\,N &\approx M \\ \underline{\mathsf{ifz}}\ulcorner n+1 \urcorner M\,N &\approx N \end{aligned}$$

Solution

$\underline{\mathsf{ifz}} = \lambda x y z.\, x\,(\lambda z_1.\, z)\, y$

** 3. The predecessor is somewhat more difficult to define as a pure term. We will need to recall that tuples and their projections, as we defined them, are themselves pure terms.

Consider the following Haskell program pred′ on natural numbers.

$$\mathsf{pred}'\ n = \mathsf{fst}\ (\mathsf{foldn}\ n\ \mathsf{incr}\ (0,0))$$

$$\mathsf{where}$$

$$\begin{aligned} \mathsf{incr}\ (n,0) &= (n,1) \\ \mathsf{incr}\ (n,1) &= (n+1,1) \end{aligned}$$

$$\begin{aligned} \mathsf{foldn}\ 0\ f\ x &= x \\ \mathsf{foldn}\ n\ f\ x &= f\ (\mathsf{foldn}\ (n-1)\ f\ x) \end{aligned}$$

(a) What is the result of computing $\mathsf{foldn}\ 3\ \mathsf{incr}\ (0,0)$?

(b) What is the result of computing $\mathsf{foldn}\ 4\ \mathsf{incr}\ (0,0)$

(c) Following the same strategy as the Haskell program, construct a *pure* term $\underline{\mathsf{pred}}$ that satisfies:

$$\underline{\mathsf{pred}}\ulcorner 0 \urcorner \approx \ulcorner 0 \urcorner \quad \text{and} \quad \underline{\mathsf{pred}}\ulcorner k+1 \urcorner \approx \ulcorner k \urcorner$$

Hint: Compare the behaviour of $\mathsf{foldn}\ n$ and $\ulcorner n \urcorner$.

Solution

(a) (2,1)

(b) (3,1)

(c) Let $\underline{\mathsf{incr}}$ be the term:

$$\lambda p.\, \underline{\mathsf{ifz}}\ (\underline{\mathsf{proj}}_2^2\ p)\ (\underline{\mathsf{proj}}_1^2\ p, \ulcorner 1 \urcorner)\ (\underline{\mathsf{S}}\ (\underline{\mathsf{proj}}_1^2\ p), \ulcorner 1 \urcorner)$$

Then we can define $\underline{\mathsf{pred}} = \lambda n.\, \underline{\mathsf{proj}}_1^2\ (n\ \underline{\mathsf{incr}}\ (\ulcorner 0 \urcorner, \ulcorner 0 \urcorner))$.

4. We have already seen in the previous problem sheet the pure term $\underline{\mathsf{Y}}$ which can be used as a fixed point combinator in place of fix.
   (a) Use this to define a *pure* term $\underline{\mathsf{addcn}}$ which satisfies:

$$\begin{aligned} \underline{\mathsf{addcn}} \ulcorner 0 \urcorner \ulcorner m \urcorner &\approx \ulcorner m \urcorner \\ \underline{\mathsf{addcn}} \ulcorner n+1 \urcorner \ulcorner m \urcorner &\approx \underline{\mathsf{S}} (\underline{\mathsf{addcn}} \ulcorner n \urcorner \ulcorner m \urcorner) \end{aligned}$$

   (b) Prove, by induction on $n \in \mathbb{N}$ that, therefore, $\underline{\mathsf{addcn}}$ satisfies:

$$\underline{\mathsf{addcn}} \ulcorner n \urcorner \ulcorner m \urcorner \approx \ulcorner n+m \urcorner$$

   Do not unpack your definition of $\underline{\mathsf{addcn}}$. Instead, just use the two equations from part (a). As long as you used the recipe correctly, your definition will be guaranteed to satisfy these two equations - that is the whole point of following the recipe!

Solution

(a) $\underline{\mathsf{Y}} (\lambda f\, x\, y. \underline{\mathsf{ifz}}\, x\, y\, (\underline{\mathsf{S}} (f\, (\underline{\mathsf{pred}}\, x)\, y)))$

(b) By induction on n:

- When $n = 0$, our goal is to show $\underline{\mathsf{addcn}} \ulcorner 0 \urcorner \ulcorner m \urcorner \approx \ulcorner 0+m \urcorner = \ulcorner m \urcorner$, and this is just the first equation given in (a).
- When n is of shape $k+1$, our goal is to show that

$$\underline{\mathsf{addcn}} \ulcorner k+1 \urcorner \ulcorner m \urcorner \approx \ulcorner k+1+m \urcorner$$

  Assume the induction hypothesis:

$$\underline{\mathsf{addcn}} \ulcorner k \urcorner \ulcorner m \urcorner \approx \ulcorner k+m \urcorner$$

  We reason equationally, using the second of the two equations given in (a), the induction hypothesis and the equation from Q1:

$$\begin{aligned} \underline{\mathsf{addcn}} \ulcorner k+1 \urcorner \ulcorner m \urcorner &\approx \underline{\mathsf{S}} (\underline{\mathsf{addcn}} \ulcorner k \urcorner \ulcorner m \urcorner) \\ &\approx \underline{\mathsf{S}} \ulcorner m+k \urcorner \\ &\approx \ulcorner m+k+1 \urcorner \end{aligned}$$

Equipped with our pure abbreviations for all of the constants, we are now in a position to define also a version of $\underline{\mathsf{mult}}$ that works on Church numerals, a version of $\underline{\mathsf{leq}}$ and so on. By doing so, we can build a translation from While programs to *pure* terms, following the same pattern as we have done for PCF terms, except that λ-*definability* will use Church numerals rather than PCF numerals. Then we will have shown that the pure λ-calculus is also Turing-complete.