

# TYPES AND $\lambda$ -CALCULUS

## Problem Sheet 5

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

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

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  $\lambda$ -term*.

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

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

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:

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

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

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

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

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

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

So that we get  $\ulcorner 3 \urcorner = \lambda f x. \text{iter}(f\ x)(3) = \lambda f x. f\ (f\ (f\ x))$ , 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  $n \in \mathbb{N}$ ,  $\ulcorner n \urcorner g\ y \approx \text{iter}(g, y)(n)$ .
- (b) Use this to show that, for all  $n \in \mathbb{N}$ ,  $\underline{S}\ulcorner n \urcorner \approx \ulcorner n + 1 \urcorner$ .

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

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

\*\* 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  $\text{pred}'$  on natural numbers.

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

where

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

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

- (a) What is the result of computing  $\text{foldn}\ 3\ \text{incr}\ (0, 0)$ ?
- (b) What is the result of computing  $\text{foldn}\ 4\ \text{incr}\ (0, 0)$ ?
- (c) Following the same strategy as the Haskell program, construct a *pure* term  $\underline{\text{pred}}$  that satisfies:

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

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

4. We have already seen in the previous problem sheet the pure term  $\underline{Y}$  which can be used as a fixed point combinator in place of  $\text{fix}$ .

(a) Use this to define a *pure* term  $\underline{\text{addcn}}$  which satisfies:

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

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

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

Do not unpack your definition of  $\underline{\text{addcn}}$ . 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!

Equipped with our pure abbreviations for all of the constants, we are now in a position to define also a version of  $\underline{\text{mult}}$  that works on Church numerals, a version of  $\underline{\text{leq}}$  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  $\lambda$ -definability will use Church numerals rather than PCF numerals. Then we will have shown that the pure  $\lambda$ -calculus is also Turing-complete.