Types for the Scott numerals

Martín Abadi Luca Cardelli Gordon Plotkin

February 18, 1993

In the untyped lambda calculus, the Scott numerals are defined by:

$$\begin{array}{rcl} 0 & = & \lambda x \lambda y.\, x \\ \mathit{succ} & = & \lambda n \lambda x \lambda y.\, yn \\ \mathit{case} & = & \lambda n \lambda a \lambda f.\, naf \end{array}$$

where $\mathit{case}(n)(a)(f)$ returns a if n is 0 and $f(x)$ if n is the successor of x (see, e.g., [2]). The Scott numerals are distinguished from the Church numerals by their "linearity": the bound variables of 0, *succ*, and *case* occur at most once in the bodies of these functions, and the predecessor function $\lambda n.\, n(0)(\lambda x.x)$ can be computed trivially.

The Scott numerals can be typed in an extension of System F with covariant recursive types. We can take the type of the Scott numerals to be the solution to the equation $S = \forall R.\, (R \to (S \to R) \to R)$:

$$\begin{array}{rcl} 0 & = & \Lambda R \lambda x : R \lambda y : (S \to R).\, x \\ & : & S \\ \mathit{succ} & = & \lambda n : S \Lambda R \lambda x : R \lambda y : (S \to R).\, yn \\ & : & S \to S \\ \mathit{case} & = & \lambda n : S \Lambda R \lambda a : R \lambda f : (S \to R).\, nRaf \\ & : & S \to \forall R.(R \to (S \to R) \to R) \end{array}$$

Since S is a covariant recursive type, it can be represented by the System F type $M = \mu X.\, G[X]$ where $G[X] = \forall R.\, (R \to (X \to R) \to R)$ and $\mu X.\, G[X] = \forall X.\, ((G[X] \to X) \to X)$. Let $N = G[M]$, and let $\mathit{in} : N \to M$ and $\mathit{out} : M \to N$ be the two halves of the isomorphism between M and N. Using *in* and *out*, we can give new System F versions of 0, *succ*, and *case*.

For this, it turns out to be easiest to use N rather than M as the type of numbers. We set:

$$\begin{array}{rcl} 0 & = & \lambda R \lambda x : R \lambda y : (M \to R).\, x \\ & : & N \\ \mathit{succ} & = & \lambda n : N \Lambda R \lambda x : R \lambda y : (M \to R).\, y(\mathit{in}(n)) \\ & : & N \to N \\ \mathit{case} & = & \lambda n : N \Lambda R \lambda a : R \lambda f : (N \to R).\, nRa(f \circ \mathit{out}) \\ & : & N \to \forall R.(R \to (N \to R) \to R) \end{array}$$

A numeral system based on M can be obtained by working through the isomorphism, or can be derived from scratch. These two approaches yield somewhat different results. An abundance of superficially different numeral systems can be obtained, in part because neither $\mathit{in} \circ \mathit{out}$ nor $\mathit{out} \circ \mathit{in}$ is $\beta\eta$-equivalent to the identity. Note that these numeral systems are not as efficient as the original one, or as the one based on the recursive type S: the two halves of the isomorphism *in* and *out* are not linear.

After considering these typed version of the Scott numerals, we may wish to check that they are in fact isomorphic to the standard natural numbers. A direct argument uses many of the datatype constructions studied in [1]:

$$\begin{array}{rcll} M & \equiv & N & \text{by unfolding} \\ & \equiv & \mu X \forall R.\, ((1 \to R) \to (X \to R) \to R) & \text{since } R \equiv (1 \to R) \\ & \equiv & \mu X \forall R.\, (((1 \to R) \times (X \to R)) \to R) & \text{by uncurrying} \\ & \equiv & \mu X \forall R.\, (((1 + X) \to R) \to R) & \text{turning a } \times \text{ into a } + \\ & \equiv & \mu X.\, (1 + X) & \text{as } 1 + X \equiv \mu R.\, (1 + X) \end{array}$$

Similarly, we can give Scott versions for other familiar datatypes using covariant recursive types. This works out particularly well when recursive types can be defined up to equality rather than just up to isomorphism.

References

[1] Gordon Plotkin and Martín Abadi. A logic for parametric polymorphism. To appear in *Proceedings of the International Conference on Typed Lambda Calculi and Applications*, March 1993.

[2] Christopher Wadsworth. Some unusual λ-calculus numeral systems. In *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, J.P. Seldin and J. R. Hindley, eds., Academic Press, 1980.