Some answers to the Exercises on Lecture 8

Herman Geuvers

June 23, 2008

Recall: \( \bot := \forall \alpha. \alpha \), \( \top := \forall \alpha. \alpha \to \alpha \). In these exercises, the main “clue” is what to instantiate the \( \forall \alpha \) * quantifier with. This is not made explicit in the Curry system, but the derivations should make it clear. If not, write down the Chur=rch variant of the same term with the same derivation.

1. (basic) Verify that in Church \( \lambda 2 \): \( \lambda x : \top . x \top : \top \to \top \).

\[
\begin{array}{l}
1 \quad x : \forall \alpha. \alpha \to \alpha \\
2 \quad x \top : \top \to \top \quad \text{app, 1} \\
3 \quad x \top x : \top \quad \text{app, 2} \\
4 \quad \lambda x : \top . x \top x : \top \to \top \quad \lambda\text{-rule, 1, 3}
\end{array}
\]

2. (medium) Verify that in Curry \( \lambda 2 \): \( \lambda x x : \top \to \top \).

\[
\begin{array}{l}
1 \quad x : \forall \alpha. \alpha \to \alpha \\
2 \quad x : \top \to \top \quad \text{app, 1} \\
3 \quad x x : \top \quad \text{app, 2} \\
4 \quad \lambda x x : \top \to \top \quad \lambda\text{-rule, 1, 3}
\end{array}
\]

3. (medium) Find a type in Curry \( \lambda 2 \) for \( \lambda x x x x \).

\[
\begin{array}{l}
1 \quad x : \forall \alpha. \alpha \to \alpha \\
2 \quad x : \top \to \top \quad \text{app, 1} \\
3 \quad x x : \top \quad \text{app, 2} \\
4 \quad x x : \top \to \top \quad \text{app, 3} \\
5 \quad x x x : \top \quad \text{app, 4} \\
6 \quad \lambda x x x : \top \to \top \quad \lambda\text{-rule, 1, 5}
\end{array}
\]

OR:
4. (medium) Find a type in Curry $\lambda 2$ for $\lambda x.(x x)(x x)$

1. (medium) Define $\text{inl} : \sigma \to \sigma + \tau$
   Recall that $\sigma + \tau ::= \forall \alpha. \sigma \to \alpha \to (\tau \to \alpha) \to \alpha$
   Answer:
   $$\lambda x : \sigma. \lambda \alpha. \lambda f : \sigma \to \alpha. \lambda g : \tau \to \alpha. f\ x$$

2. (medium) Define $\text{pairing} : [-, -] : \sigma \to \tau \to \sigma \times \tau$
   Recall that $\sigma \times \tau ::= \forall \alpha. (\sigma \to \tau \to \alpha) \to \alpha,$
   Answer:
   $$\lambda x : \sigma. \lambda y : \tau. \lambda \alpha. \lambda h : \sigma \to \tau \to \alpha. h\ x\ y$$
   NB You can only “validate” this definition if you define projections $\pi_1$ and $\pi_2$ and show that $\pi_1[a, b] =_\beta a$ and $\pi_2[a, b] =_\beta b.$ Try to do that. (Here is the definition of $\pi_1$: $\lambda z : \sigma \times \tau. \z \sigma (\lambda x : \sigma. \lambda y : \tau. x)$)

3. (advanced) Define $\text{join} : \text{Tree}_{A,B} \to \text{Tree}_{A,B} \to A \to \text{Tree}_{A,B}$
   Recall that
   $$\text{Tree}_{A,B} ::= \forall \alpha. \text{Tree}_{A,B}(B \to \alpha) \to (A \to \alpha \to \alpha) \to \alpha$$
   Now, $\text{join}$ is defined as follows:
   $$\text{join} ::= \lambda t_1 : \text{Tree}_{A,B}. \lambda t_2 : \text{Tree}_{A,B}. \lambda a : A.
   \lambda \alpha. \lambda f : B \to \alpha. \lambda h : A \to \alpha \to \alpha. h\ a\ (t_1\ \alpha\ f\ h)(t_2\ \alpha\ f\ h)$$
   Why is this the right answer?
   (1) There is a very general way to define the constructors for a data type defined in $A 2,$ but I haven’t shown that to you. (The general method has first been described in C. Böhm and A. Berarducci, *Automatic synthesis of typed lambda programs on term algebras.* Theoretical Computer Science, 39(2-3):135–153, Aug. 1985.)
(2) Another answer is: Given $t_1$, $t_2$ and $a$, we have to define a term of type $\text{Tree}_{A,B}$. This will have the shape

$$\lambda \alpha. \lambda f : B \rightarrow \alpha. \lambda h : A \rightarrow \alpha \rightarrow \alpha.$$ 

with $? : \alpha$. We can view $h$ as the “internal” join function and $t_1 \alpha fh$ is the “internal” representation of $t_1$ and $t_2 \alpha fh$ is the “internal” representation of $t_2$, so we need to apply $h$ to these terms, taking $a$ as the node label.

...This works well as an intuition, but I agree that it’s vague ...

(3) The best answer is: define your destructors and show that they “work” with join. So: define “left” and “right” and show that left (join $a \ t_1 \ t_2$) $=_{\beta} t_1$ and similarly for “right” and $t_2$.

$$\text{left} := \lambda t : \text{Tree}_{A,B}. \ t \ \text{Tree}_{A,B} \ \text{leaf} \ (\lambda a : A \ \lambda t_1, t_2 : \text{Tree}_{A,B} \ . \ t_1)$$

where $\text{leaf} : B \rightarrow \text{Tree}_{A,B}$ is the function

$$\lambda b : B. \ \lambda \alpha. \ \lambda f : B \rightarrow \alpha. \ \lambda h : A \rightarrow \alpha \rightarrow \alpha. \ f \ b$$

3