

Proving with Computer Assistance, 2IF65

Herman Geuvers, TUE

Some answers to the Exercises on Lecture 4

1. Recall: $\perp := \forall\alpha.\alpha$, $\top := \forall\alpha.\alpha \rightarrow \alpha$. 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 Church variant of the same term with the same derivation.

(a) Verify that in Church $\lambda 2$: $\lambda x:\top.x\top x : \top \rightarrow \top$.

$$\begin{array}{l|l} 1 & \frac{}{x : \forall\alpha.\alpha \rightarrow \alpha} \\ 2 & \frac{}{x\top : \top \rightarrow \top} \quad \text{app, 1} \\ 3 & \frac{}{x\top x : \top} \quad \text{app, 2} \\ 4 & \frac{}{\lambda x:\top.x\top x : \top \rightarrow \top} \quad \lambda\text{-rule, 1, 3} \end{array}$$

(b) Verify that in Curry $\lambda 2$: $\lambda x.xx : \top \rightarrow \top$

$$\begin{array}{l|l} 1 & \frac{}{x : \forall\alpha.\alpha \rightarrow \alpha} \\ 2 & \frac{}{x : \top \rightarrow \top} \quad \text{app, 1} \\ 3 & \frac{}{xx : \top} \quad \text{app, 2} \\ 4 & \frac{}{\lambda x.xx : \top \rightarrow \top} \quad \lambda\text{-rule, 1, 3} \end{array}$$

(c) Find a type in Curry $\lambda 2$ for $\lambda x.x x x$

$$\begin{array}{l|l} 1 & \frac{}{x : \forall\alpha.\alpha \rightarrow \alpha} \\ 2 & \frac{}{x : \top \rightarrow \top} \quad \text{app, 1} \\ 3 & \frac{}{xx : \top} \quad \text{app, 2} \\ 4 & \frac{}{xx : \top \rightarrow \top} \quad \text{app, 3} \\ 5 & \frac{}{xxx : \top} \quad \text{app, 4} \\ 6 & \frac{}{\lambda x.x x x : \top \rightarrow \top} \quad \lambda\text{-rule, 1, 5} \end{array}$$

OR:

$$\begin{array}{l|l} 1 & \frac{}{x : \perp} \\ 2 & \frac{}{x : \perp \rightarrow \perp \rightarrow \perp} \quad \text{app, 1} \\ 3 & \frac{}{xx : \perp \rightarrow \perp} \quad \text{app, 2, 1} \\ 4 & \frac{}{xxx : \perp} \quad \text{app, 3, 1} \\ 5 & \frac{}{\lambda x.x x x : \perp \rightarrow \perp} \quad \lambda\text{-rule, 1, 4} \end{array}$$

(d) Find a type in Curry $\lambda 2$ for $\lambda x.(xx)(xx)$

1	$x : \perp$	
2	$x : \perp \rightarrow \perp$	app, 1
3	$xx : \perp$	app, 2, 1
4	$xx : \perp \rightarrow \perp$	app, 3
5	$(xx)(xx) : \perp$	app, 4, 3
6	$\lambda x.(xx)(xx) : \perp \rightarrow \perp$	λ -rule, 1, 5

2. (a) Define $\text{inl} : \sigma \rightarrow \sigma + \tau$
 Recall that $\sigma + \tau := \forall \alpha. (\sigma \rightarrow \alpha) \rightarrow (\tau \rightarrow \alpha) \rightarrow \alpha$
 Answer:

$$\lambda x : \sigma. \lambda \alpha. \lambda f : \sigma \rightarrow \alpha. \lambda g : \tau \rightarrow \alpha. f x$$

- (b) Define pairing : $[-, -] : \sigma \rightarrow \tau \rightarrow \sigma \times \tau$
 Recall that $\sigma \times \tau := \forall \alpha. (\sigma \rightarrow \tau \rightarrow \alpha) \rightarrow \alpha$,
 Answer:

$$\lambda x : \sigma. \lambda y : \tau. \lambda \alpha. \lambda h : \sigma \rightarrow \tau \rightarrow \alpha. h x y$$

NB You can only “validate” this definition if you define projections π_1 and π_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 π_1 : $\lambda z : \sigma \times \tau. z \sigma (\lambda x : \sigma. \lambda y : \tau. x)$)

- (c) Define $\text{join} : \text{Tree}_{A,B} \rightarrow \text{Tree}_{A,B} \rightarrow A \rightarrow \text{Tree}_{A,B}$ Recall that

$$\text{Tree}_{A,B} := \forall \alpha. (B \rightarrow \alpha) \rightarrow (A \rightarrow \alpha \rightarrow \alpha \rightarrow \alpha) \rightarrow \alpha$$

Now, join is defined as follows:

$$\begin{aligned} \text{join} &:= \lambda t_1 : \text{Tree}_{A,B}. \lambda t_2 : \text{Tree}_{A,B}. \lambda a : A. \\ &\lambda \alpha. \lambda f : B \rightarrow \alpha. \lambda h : A \rightarrow \alpha \rightarrow \alpha \rightarrow \alpha. ha(t_1 \alpha f h)(t_2 \alpha f h) \end{aligned}$$

Why is this the right answer?

(1) There is a very general way to define the constructors for a data type defined in $\lambda 2$, 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

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

with $? : \alpha$. We can view h as the “internal” join function and $t_1 \alpha f h$ is the “internal” representation of t_1 and $t_2 \alpha f h$ 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 $\text{left}(\text{join } a \ t_1 \ t_2) =_{\beta} t_1$ 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

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