Proving with Computer Assistance, 2IMF15

Herman Geuvers, TUE

Exercises on Polymorphic type Theory, some answers

1. Recall: $\bot := \forall\alpha.\alpha$, $\top := \forall\alpha.\alpha{\to}\alpha$. In these exercises, the main "clue" is what to instantaite 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 Chut=rch variant of the same term witrh the same derivation.

   (a) Verify that in Church $\lambda 2$: $\lambda x{:}\top.x\top x : \top{\to}\top$.

1	$x : \forall\alpha.\alpha{\to}\alpha$	
2	$x\,\top : \top \to \top$	app, 1
3	$x\,\top\,x : \top$	app, 2
4	$\lambda x{:}\top.x\,\top\,x : \top \to \top$	λ-rule, 1, 3

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

1	$x : \forall\alpha.\alpha{\to}\alpha$	
2	$x : \top \to \top$	app, 1
3	$x\,x : \top$	app, 2
4	$\lambda x.x\,x : \top \to \top$	λ-rule, 1, 3

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

1	$x : \forall\alpha.\alpha{\to}\alpha$	
2	$x : \top \to \top$	app, 1
3	$x\,x : \top$	app, 2
4	$x\,x : \top \to \top$	app, 3
5	$x\,x\,x : \top$	app, 4
6	$\lambda x.x\,x\,x : \top \to \top$	λ-rule, 1, 5

   OR:

1	$x : \bot$	
2	$x : \bot \to \bot \to \bot$	app, 1
3	$x\,x : \bot \to \bot$	app, 2, 1
4	$x\,x\,x : \bot$	app, 3, 1
5	$\lambda x.x\,x\,x : \bot \to \bot$	λ-rule, 1, 4

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

1	$x : \bot$	
2	$x : \bot \to \bot$	app, 1
3	$x\,x : \bot$	app, 2, 1
4	$x\,x : \bot \to \bot$	app, 3
5	$(x\,x)\,(x\,x) : \bot$	app, 4, 3
6	$\lambda x.(x\,x)(x\,x) : \bot \to \bot$	λ-rule, 1, 5

2. Let $x : \top$ and remember that $\top := \forall\alpha{:}*.\alpha{\to}\alpha$. Give a type to the term

$$\lambda y.x\,y\;x(\lambda z.z\,x\,z)$$

in $\lambda 2$ à la Curry and give the typing derivation of your result.

3. Let $x : \top$ and remember that $\top := \forall\alpha{:}*.\alpha{\to}\alpha$.

   (a) Give a type to the term

   $$\lambda y.x\,y\;x(\lambda z.z\,x\,z)$$

   in $\lambda 2$ à la Curry and give the typing derivation of your result.

1	$x : \top$	
2	$\quad y : \bot$	
3	$\quad x : \bot \to \bot$	app, 1
4	$\quad x\,y : \bot$	app, 3, 2
5	$\quad x\,y : \top \to \bot$	app, 4
6	$\quad x\,y\,x : \bot$	app, 4, 1
7	$\quad\quad z : \bot$	
8	$\quad\quad z : \top \to \bot$	app, 7
9	$\quad\quad z\,x : \bot$	app, 8, 1
10	$\quad\quad z\,x : \bot \to \bot$	app, 9
11	$\quad\quad z\,x\,z : \bot$	app, 10, 1
12	$\quad \lambda z.z\,x\,z : \bot \to \bot$	λ-rule, 7, 11
13	$\quad x\,y\,x : (\bot \to \bot) \to \bot$	app, 6
14	$\quad x\,y\,x\,(\lambda z.z\,x\,z) : \bot$	app, 13, 12
15	$\lambda y.x\,y\,x\,(\lambda z.z\,x\,z) : \bot \to \bot$	λ-rule, 2, 14

   (b) Give a type to the term

   $$\lambda y.x\,y\,(x(\lambda z.z\,z))$$

   in $\lambda 2$ à la Curry. Also give the typing derivation of your result.

4. (a) Define $\mathrm{inl} : \sigma \to \sigma + \tau$
   Recall that $\sigma + \tau := \forall\alpha.(\sigma{\to}\alpha){\to}(\tau{\to}\alpha){\to}\alpha$
   Answer:

   $$\lambda x : \sigma.\, \lambda\alpha.\, \lambda f : \sigma{\to}\alpha.\, \lambda g : \tau{\to}\alpha.\, f\,x$$

   (b) Define 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 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 definituion of π_1: $\lambda z : \sigma \times \tau.\, z\,\sigma\,(\lambda x : \sigma.\, \lambda y : \tau.\, x)$)

   (c) Show that the addition function (as defined on the slides) behaves as expected.
   Check that for $\mathrm{Plus} := \lambda n : \mathrm{Nat}.\lambda m : Nat.n\,\mathrm{Nat}\,m\,S$, we have

   $$\begin{aligned} \mathrm{Plus}\,0\,y &= y \\ \mathrm{Plus}\,(S\,x)\,y &= s\,(\mathrm{Plus}\,x\,y) \end{aligned}$$

   where $S := \lambda n : \mathrm{Nat}.\lambda\alpha.\lambda z : \alpha.\lambda f : \alpha{\to}\alpha.f(n\,\alpha\,z\,f)$.

(d) Define leaf : $B \to \text{Tree}_{A,B}$ and join : $\text{Tree}_{A,B}$ and join : $\text{Tree}_{A,B} \to \text{Tree}_{A,B} \to A \to \text{Tree}_{A,B}$
Recall that

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

Now, leaf $:= \lambda b : B.\lambda\alpha.\,\lambda f : B{\to}\alpha.\,\lambda h : A{\to}\alpha{\to}\alpha{\to}\alpha.\,f\,b$ and join is defined as follows:

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

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

$$\lambda\alpha.\,\lambda f : B{\to}\alpha.\,\lambda h : A{\to}\alpha{\to}\alpha{\to}\alpha.?$$

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 $(\text{join}\,a\,t_1\,t_2) =_\beta t_1$ 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 leaf : $B{\to}\text{Tree}_{A,B}$ is the function

$$\lambda b : B.\,\lambda\alpha.\,\lambda f : B{\to}\alpha.\,\lambda h : A{\to}\alpha{\to}\alpha{\to}\alpha.\,f\,b$$

(e) Give the Tree-iteration scheme for $\text{Tree}_{A,B}$ and define $h : \text{Tree}_{A,B} \to \text{Nat}$ that counts the number of leaves of a tree.
The Tree iteration scheme is: given a type D and $f : B{\to}D$, $g : A{\to}D{\to}D{\to}D$, there is a term $k : \text{Tree}_{A,B}{\to}D$ satisfying

$$\begin{array}{rcl}k\,(\text{leaf}\,b) & = & f\,b \\ k\,(\text{join}\,a\,t_1\,t_2) & = & g\,a\,(k\,t_1)\,(k\,t_2)\end{array}$$

as a matter of fact k is just $\lambda t : \text{Tree}_{A,B}.t\,D\,f\,g$.
The function h that counts the number of leaves satisfies

$$\begin{array}{rcl}h\,(\text{leaf}\,b) & = & S\,0 \\ h\,(\text{join}\,a\,t_1\,t_2) & = & \text{Plus}\,(h\,t_1)\,(h\,t_2)\end{array}$$

so we can take $h := \lambda t : \text{Tree}_{A,B}.t\,\text{Nat}\,(\lambda b : B.S\,0)\,(\lambda a : A, \lambda n_t, n_2 : \text{Nat.Plus}\,n_1\,n_2)$.