# 15-814 Homework 3 Solutions

October 25, 2017

**Task 1** *Define the function $f$ in the language* **LNS** *from Homework 2, without using well-founded recursion. You will want to make use of pairs.*

**Solution:** We compute for each $n$ the pair $\langle f(n), f(n+1)\rangle$ and take the first projection. We assume that the usual $+$ has been defined, e.g. see Homework 2's solution notes.

$$\mathtt{f} = \lambda n{:}\mathtt{nat}.\pi_1\mathtt{natrec}(n; \langle \mathtt{z}, \mathtt{s}(\mathtt{z})\rangle; x.y.\langle \pi_2 y, \pi_1 y + \pi_2 y\rangle)$$

**Task 2** *Define the function $f$ in System* **T***, using well-founded recursion.*

**Solution:** We first define the function $F$ for which we want to perform a fixed-point computation. We again assume that $\mathtt{ifz}, \leq, -, +$ have been defined following Homework 2.

$$\mathtt{F} = \lambda f{:}\mathtt{nat} \to \mathtt{nat}.\lambda n{:}\mathtt{nat}.\mathtt{ifz}\ (n \leq \mathtt{s}(\mathtt{z}))\ n\ (f(n - \mathtt{s}(\mathtt{s}(\mathtt{z}))) + f(n - \mathtt{s}(\mathtt{z})))$$

It is useful to note its type $\mathtt{F} : (\mathtt{nat} \to \mathtt{nat}) \to (\mathtt{nat} \to \mathtt{nat})$. A fixed point of $\mathtt{F}$ would be an $x : \mathtt{nat} \to \mathtt{nat}$ such that $\mathtt{F}x = x$. Next, we define the n-fold composition of a function with itself.

$$\mathtt{ncompose} = \lambda f{:}(\mathtt{nat} \to \mathtt{nat}) \to (\mathtt{nat} \to \mathtt{nat}).\lambda n{:}\mathtt{nat}.\mathtt{natrec}(n; \lambda m{:}\mathtt{nat}.m; x.y.f\ y)$$

Finally, we put the two together to define what we want.

$$\mathtt{f} = \lambda n{:}\mathtt{nat}.\mathtt{ncompose}\ \mathtt{F}\ n\ n$$

To get a better understanding of what is going on, let us manually unfold $\mathtt{f}\ 3$. (I write $\equiv$ here because some of these steps are not in correct evaluation order). Notice that composing $\mathtt{F}$ with itself 3 times is sufficient, because we always decrease $n$ by at least 1 in each recursive call.

$$\begin{aligned}
\mathtt{f}\ 3 &\equiv \mathtt{ncompose}\ \mathtt{F}\ 3\ 3 \\
&\equiv \mathtt{natrec}(3; \lambda m{:}\mathtt{nat}.m; x.y.\mathtt{F}\ y)\ 3 \\
&\equiv \mathtt{F}\ (\mathtt{F}\ (\mathtt{F}\ \overbrace{\lambda m{:}\mathtt{nat}.m}^{\mathtt{id}}))\ 3 \\
&\equiv (\mathtt{F}^3\ \mathtt{id})\ 3 \\
&\equiv \mathtt{ifz}\ (3 \leq 1)\ 2\ ((\mathtt{F}^2\ \mathtt{id})\ (3-2) + (\mathtt{F}^2\ \mathtt{id})\ (3-1)) \\
&\equiv (\mathtt{F}^2\ \mathtt{id})\ 1 + (\mathtt{F}^2\ \mathtt{id})\ 2 \\
&\equiv (\mathtt{F}^2\ \mathtt{id})\ 1 + (\mathtt{F}\ \mathtt{id})\ 0 + (\mathtt{F}\ \mathtt{id})\ 1 \\
&\equiv 2
\end{aligned}$$

Essentially, the n-fold composition acts like a "clock" or "fuel" for $F$ that is decremented on each recursive call. This ensures well-foundedness because we cannot decrement the clock forever.

**Task 3** *For each of the following pairs of types $\tau_1, \tau_2$, define expressions $f, g$ with types $f : \tau_1 \to \tau_2$ and $g : \tau_2 \to \tau_1$ respectively.*

**Solution:** I write $\lambda a : \tau, b : \tau'.e$ as shorthand for $\lambda a : \tau.\lambda b : \tau'.e$. Note that we can also understand these using the Curry-Howard isomorphism. For example, (Triple Negation) corresponds to the tautology $\neg\neg\neg p \iff \neg p$ that was mentioned in class.

1. (Currying) $\tau_1 \to (\tau_2 \to \tau_3)$, $(\tau_1 \times \tau_2) \to \tau_3$

$$f = \lambda x{:}\tau_1 \to (\tau_2 \to \tau_3), p : \tau_1 \times \tau_2.x\ (\pi_1 p)\ (\pi_2 p)$$

$$g = \lambda x{:}\tau_1 \times \tau_2 \to \tau_3, \lambda p_1{:}\tau_1, p_2{:}\tau_2.x\ \langle p_1, p_2\rangle$$

2. (Permutation) $(\tau_1 + \tau_2) + \tau_3$, $(\tau_2 + \tau_3) + \tau_1$

$$f = \lambda x{:}(\tau_1 + \tau_2) + \tau_3.\texttt{case}\ x\ \{y.\texttt{case}\ y\ \{l.\texttt{inr}(l); r.\texttt{inl}(\texttt{inl}(r))\}; y.\texttt{inl}(\texttt{inr}(y))\}$$

Similarly for $g$.

3. (Distributivity) $\tau \times (\tau_1 + \tau_2)$, $(\tau \times \tau_1) + (\tau \times \tau_2)$

$$f = \lambda x{:}\tau \times (\tau_1 + \tau_2).\texttt{case}\ \pi_2 x\ \{y.\texttt{inl}(\langle \pi_1 x, y\rangle); y.\texttt{inr}(\langle \pi_1 x, y\rangle)\}$$

$$g = \lambda x{:}(\tau \times \tau_1) + (\tau \times \tau_2).\texttt{case}\ x\ \{y.\langle \pi_1 y, \texttt{inl}(\pi_2 y)\rangle; y.\langle \pi_1 y, \texttt{inr}(\pi_2 y)\rangle\}$$

4. (De Morgan) $(\tau_1 + \tau_2) \to \texttt{void}$, $(\tau_1 \to \texttt{void}) \times (\tau_2 \to \texttt{void})$

$$f = \lambda x{:}(\tau_1 + \tau_2) \to \texttt{void}.\langle \lambda y{:}\tau_1.x\ \texttt{inl}(y), \lambda y{:}\tau_2.x\ \texttt{inr}(y)\rangle$$

$$g = \lambda x{:}(\tau_1 \to \texttt{void}) \times (\tau_2 \to \texttt{void}).\lambda y{:}\tau_1 + \tau_2.\texttt{case}\ y\ \{z.\pi_1 x\ z; z.\pi_2 x\ z\}$$

5. (Soundness) $\tau \times (\tau \to \texttt{void})$, $\texttt{void}$

$$f = \lambda x{:}\tau \times (\tau \to \texttt{void}).(\pi_2 x)\ (\pi_1 x)$$

$$g = \lambda x{:}\texttt{void}.\texttt{abort}[\tau \times (\tau \to \texttt{void})](x)$$

6. (Triple Negation) $((\tau \to \texttt{void}) \to \texttt{void}) \to \texttt{void}$, $\tau \to \texttt{void}$

$$f = \lambda x{:}((\tau \to \texttt{void}) \to \texttt{void}) \to \texttt{void}, y{:}\tau.x\ (\lambda v{:}\tau \to \texttt{void}.v\ y)$$

$$g = \lambda x{:}\tau \to \texttt{void}.\lambda y{:}((\tau \to \texttt{void}) \to \texttt{void}).y\ x$$

**Task 4** *Lists can either be* `Nil` *or* `Cons` *of an element and a list. In ML-style, we write*

$$\tau\ \texttt{list} \triangleq \texttt{Nil} \mid \texttt{Cons of}\ \tau \times \tau\ \texttt{list}$$

*Define the following types and functions in System* $\mathsf{F}$*. As usual, briefly explain (in 1-2 lines) the intuition behind your answer.*

1. *Define $\tau$ `list` in System $\mathsf{F}$.*
2. *Define $\texttt{nil}_\tau : \tau\ \texttt{list}$, the empty list.*
3. *Give the representation of the list $[a_1, a_2, \ldots, a_n]$ where $a_i : \tau$.*
4. *Define $\texttt{cons}_\tau : \tau \to \tau\ \texttt{list} \to \tau\ \texttt{list}$.*

5. *Define* $\texttt{listrec}(l, e_0, x.y.e_1) : \rho$ *(analogous to* $\texttt{natrec}$ *in System* $\mathbf{T}$*). Its static and dynamic semantics are given below. Here,* $x$ *is bound to the head of the list, and* $y$ *is bound to the result of the computation on the tail of the list.*

$$\frac{\Gamma \vdash l : \tau\ \texttt{list} \quad \Gamma \vdash e_0 : \rho \quad \Gamma, x : \tau, y : \rho \vdash e_1 : \rho}{\Gamma \vdash \texttt{listrec}(l, e_0, x.y.e_1) : \rho}\ (\textsc{ListRec})$$

$$\frac{}{\texttt{listrec}(\texttt{nil}, e_0, x.y.e_1) \mapsto e_0}\ (\textsc{ListRec-Nil})$$

$$\frac{}{\texttt{listrec}(\texttt{cons}(h, t), e_0, x.y.e_1) \mapsto [h, \texttt{listrec}(t, e_0, x.y.e_1)/x, y]e_1}\ (\textsc{ListRec-S})$$

6. *Define the function append* : $\tau\ \texttt{list} \to \tau\ \texttt{list} \to \tau\ \texttt{list}$*, which takes two lists and appends the second list at the end of the first. (You can define this by working it out with* $\texttt{listrec}$*, but there is a cleaner, more elegant solution.)*

**Solution:**

1. Note the similarity between this and natural numbers (what if $\tau = \texttt{unit}$?)

$$\tau\ \texttt{list} = \forall \alpha. \alpha \to (\tau \to \alpha \to \alpha) \to \alpha$$

2. 

$$\texttt{nil}_\tau : \tau\ \texttt{list} = \Lambda \alpha. \lambda m{:}\alpha. \lambda c{:}\tau \to \alpha \to \alpha. m$$

3. 

$$[a_1, a_2, \ldots, a_n] = \Lambda \alpha. \lambda m{:}\alpha. \lambda c{:}\tau \to \alpha \to \alpha. c\ a_1 (c\ a_2 (\ \ldots\ (c\ a_n\ m)))$$

4. 

$$\texttt{cons}_\tau : \tau \to \tau\ \texttt{list} \to \tau\ \texttt{list} = \lambda h{:}\tau. \lambda t{:}\tau\ \texttt{list}. \Lambda \alpha. \lambda m{:}\alpha. \lambda c{:}\tau \to \alpha \to \alpha. c\ h\ (t[\alpha]\ m\ c)$$

5. 

$$\texttt{listrec}(l : \tau\ \texttt{list}, e_0, x.y.e_1) : \rho = l[\rho]\ e_0\ (\lambda x{:}\tau. \lambda y{:}\rho. e_1)$$

6. 

$$\texttt{append} : \tau\ \texttt{list} \to \tau\ \texttt{list} \to \tau\ \texttt{list} = \lambda x{:}\tau\ \texttt{list}. \lambda y{:}\tau\ \texttt{list}. \Lambda \alpha. \lambda m{:}\alpha. \lambda c{:}\tau \to \alpha \to \alpha. x[\alpha]\ (y[\alpha]\ m\ c)\ c$$