Part II Types: Exercise Sheet 2

Nathanael Alcock, Dylan McDermott, Ian Orton, Shaun Steenkamp, Domagoj Stolfa, *et al.*

October 2018

Some questions due to Andrew Pitts and Benjamin C. Pierce.

Tips

- When writing proofs, it can sometimes be ambiguous to say that something holds ”by induction” when you mean ”by the inductive hypothesis”. When that is the case, try to be clear about what you are using.
- Try to write proofs in one direction. Avoid writing some steps of the proof next to other ones without numbering them.

Questions

1. Complete Exercises 2, 3 and 4 from Lecture 5:
   - Define a Church encoding for the unit type.
   - Define a Church encoding for the empty type.
   - Define a Church encoding for binary trees, corresponding to the ML datatype `type tree = Leaf | Node of tree * X * tree`.
2. Assume encodings of product and existential types in System F (and associated terms, e.g. $\mathsf{fst}\, e$ and $\mathsf{pack}_{\alpha,B}(A, e)$). The signature BOOL and terms yes, no and choose are given by:

$$\begin{array}{rlr} \mathsf{BOOL} &= (\beta \times \beta) \times \forall\alpha.\, \beta \to \alpha \to \alpha \to \alpha & \beta \vdash \mathsf{BOOL}\ \mathsf{type} \\ \mathsf{yes} &= \mathsf{fst}\,(\mathsf{fst}\, b) & \beta; b : \mathsf{BOOL} \vdash \mathsf{yes} : \beta \\ \mathsf{no} &= \mathsf{snd}\,(\mathsf{fst}\, b) & \beta; b : \mathsf{BOOL} \vdash \mathsf{no} : \beta \\ \mathsf{choose} &= \mathsf{snd}\, b & \beta; b : \mathsf{BOOL} \vdash \mathsf{choose} : \forall\alpha.\, \beta \to \alpha \to \alpha \to \alpha \end{array}$$

   Note: BOOL *can be thought of as an abbreviation for* $(\beta \times \beta) \times \forall\alpha\,.\, \beta \to \alpha \to \alpha \to \alpha$*, and* β *can be thought of as the implementation of the* BOOL *interface.*

   (a) Define a term and such that
$$\beta; b : \mathsf{BOOL} \vdash \mathsf{and} : \beta \to \beta \to \beta$$
   which returns the conjunction of its two arguments.

   (b) Define a term extend such that
$$\cdot; \cdot \vdash \mathsf{extend} : (\exists\beta.\, \mathsf{BOOL}) \to (\exists\beta.\, \mathsf{BOOL} \times (\beta \to \beta \to \beta))$$
   which takes an implementation of BOOL, and adds and to it.

   (c) What should the signature NAT of natural numbers be?
3. Complete Exercises 1, 2 and 3 from Lecture 6.
   - Prove the other direction of the closure property for the $\Theta \vdash \forall\alpha.A$ type case.
   - Prove the other direction of the substitution property for the $\Theta \vdash \forall\alpha.A$ type case.
   - Prove the fundamental lemma for the forall-introduction case $\Theta; \Gamma \vdash \Lambda\alpha.e : \forall\alpha.A$.

4. Complete Exercise 2 from Lecture 7.

   The type safety proof for state would fail if we added a C-like `free()` operation to the reference API.

   - Give a plausible-looking typing rule and operational semantics for free.
   - Find an example of a program that would break.

5. This question concerns the monadic language given in Lecture 8.

   (a) Given types X and Y, define a term:

   $$\cdot;\cdot \vdash \mathsf{fmap} : (X \to Y) \to (T\,X \to T\,Y)$$

   such that for all terms $f : X \to Y$ and values $v : X$ and $v' : Y$ where $f\,v \rightsquigarrow^* v'$ we have

   $$\langle \sigma,\ \mathsf{let}\ y = \mathsf{fmap}\ f\ \{\mathsf{return}\,v\};\ \mathsf{return}\,y\rangle \rightsquigarrow^* \langle \sigma,\ \mathsf{return}\,v'\rangle$$

   (b) For every type X, define terms:

   $$\cdot;\cdot \vdash \eta_X : X \to T\,X$$
   $$\cdot;\cdot \vdash \mu_X : T\,(T\,X) \to T\,X$$

   such that for all values $v : T\,X$ and $v' : X$ where $\langle \sigma,\ \mathsf{let}\ x = v;\ \mathsf{return}\,x\rangle \rightsquigarrow^* \langle \sigma',\ \mathsf{return}\,v'\rangle$ we have

   $$\langle \sigma,\ \mathsf{let}\ y = \mu_X\,(\eta_{(T\,X)}\,v);\ \mathsf{return}\,y\rangle \rightsquigarrow^* \langle \sigma',\ \mathsf{return}\,v'\rangle$$

Extension

6. Suppose that $\Theta, \beta \vdash F$ type, and fmap is a term such that:

   $$\Theta; \cdot \vdash \mathsf{fmap} : \forall \beta_1.\,\forall \beta_2.\,(\beta_1 \to \beta_2) \to ([\beta_1/\beta]F \to [\beta_2/\beta]F)$$

   The type μF is defined to be $\forall \beta.\,(F \to \beta) \to \beta$.

   (a) Show that $\Theta \vdash \mu F$ type and $\Theta \vdash [\mu F/\beta]F$ type.

   (b) Define terms fold and intro such that:

   $$\Theta; \cdot \vdash \mathsf{fold} : \forall \beta.\,(F \to \beta) \to \mu F \to \beta$$
   $$\Theta; \cdot \vdash \mathsf{intro} : [\mu F/\beta]F \to \mu F$$

   (Hint: you will need to use fold *in the definition of* intro*.)*

7. Let β be a type variable. For this question, Θ ranges over type contexts that do not contain β. The judgements $\Theta \vdash A\ \mathsf{type}^+$ and $\Theta \vdash A\ \mathsf{type}^-$ are defined by:

   $$\frac{}{\Theta \vdash \beta\ \mathsf{type}^+} \qquad \frac{\alpha \in \Theta}{\Theta \vdash \alpha\ \mathsf{type}^+} \qquad \frac{\Theta \vdash A\ \mathsf{type}^- \quad \Theta \vdash B\ \mathsf{type}^+}{\Theta \vdash A \to B\ \mathsf{type}^+} \qquad \frac{\Theta, \alpha \vdash A\ \mathsf{type}^+ \quad \alpha \neq \beta}{\Theta \vdash \forall \alpha.\,A\ \mathsf{type}^+}$$

   $$\frac{\alpha \in \Theta}{\Theta \vdash \alpha\ \mathsf{type}^-} \qquad \frac{\Theta \vdash A\ \mathsf{type}^+ \quad \Theta \vdash B\ \mathsf{type}^-}{\Theta \vdash A \to B\ \mathsf{type}^-} \qquad \frac{\Theta, \alpha \vdash A\ \mathsf{type}^- \quad \alpha \neq \beta}{\Theta \vdash \forall \alpha.\,A\ \mathsf{type}^-}$$

   (a) Show how to define a term $\mathsf{fmap}^+_{\Theta,A}$ for each type A such that $\Theta \vdash A\ \mathsf{type}^+$, and a term $\mathsf{fmap}^-_{\Theta,B}$ for each type B such that $\Theta \vdash B\ \mathsf{type}^-$, that satisfy:

   $$\Theta; \cdot \vdash \mathsf{fmap}^+_{\Theta,A} : \forall \beta_1.\,\forall \beta_2.\,(\beta_1 \to \beta_2) \to ([\beta_1/\beta]A \to [\beta_2/\beta]A)$$
   $$\Theta; \cdot \vdash \mathsf{fmap}^-_{\Theta,B} : \forall \beta_1.\,\forall \beta_2.\,(\beta_1 \to \beta_2) \to ([\beta_2/\beta]B \to [\beta_1/\beta]B)$$

   (Hint: the terms should be defined by induction on the derivations of $\Theta \vdash A\ \mathsf{type}^+$ *and* $\Theta \vdash B\ \mathsf{type}^-$*.)*

   (b) By considering $[\![\Theta \vdash \forall \alpha.\,\alpha\ \mathsf{type}]\!]$, show that there is no closed term of type $\forall \alpha.\alpha$. Deduce that adding the rule

   $$\frac{}{\Theta \vdash \beta\ \mathsf{type}^-}$$

   would make part (a) impossible to answer.

(c) Suppose that X and X' are semantic types and $X \subseteq X'$. Show that if $\Theta \vdash A\ \text{type}^+$ and $\Theta \vdash B\ \text{type}^-$ then for all θ,

$$\begin{aligned}
[\![\beta, \Theta \vdash A\ \text{type}]\!](X/\beta, \theta) &\subseteq [\![\beta, \Theta \vdash A\ \text{type}]\!](X'/\beta, \theta) \\
[\![\beta, \Theta \vdash B\ \text{type}]\!](X/\beta, \theta) &\supseteq [\![\beta, \Theta \vdash B\ \text{type}]\!](X'/\beta, \theta)
\end{aligned}$$

(Only do a few cases of the proof.)

8. Suppose we are given a preorder $(P, \sqsubseteq)$ that has a greatest element $(\top)$, greatest lower bounds $(\wedge)$, an implication operation $(\Longrightarrow)$ and a function Θ assigning a meaning to each base type τ as an element $\Theta(\tau) \in P$. We can then assign meanings to each type T and typing context Γ as element $\Theta[\![T]\!], \Theta[\![\Gamma]\!] \in P$ via recursion on their structure:

$$\begin{aligned}
\Theta[\![\tau]\!] &= \Theta(\tau) && \\
\Theta[\![1]\!] &= \top && \text{Greatest element} \\
\Theta[\![\tau_1 \times \tau_2]\!] &= \Theta[\![\tau_1]\!] \wedge \Theta[\![\tau_2]\!] && \text{Binary meet} \\
\Theta[\![\tau_1 \to \tau_2]\!] &= \Theta[\![\tau_1]\!] \Longrightarrow \Theta[\![\tau_2]\!] && \text{Implication} \\
\Theta[\![\cdot]\!] &= \top && \text{Greatest element} \\
\Theta[\![\Gamma, \tau]\!] &= \Theta[\![\Gamma]\!] \wedge \Theta[\![\tau]\!] && \text{Binary meet}
\end{aligned}$$

We say that if $\forall \Gamma, x, \tau. \Gamma \vdash x : \tau$ holds, then $\Theta[\![\Gamma]\!] \sqsubseteq \Theta[\![\tau]\!]$ holds in any such preorder.

Under these semantics for our types, why is Peirce's law $\cdot \vdash ((\tau \to T) \to \tau) \to \tau$ not provable, whereas in classical logic it is a tautology?

NB: the greatest element, binary meets and implication are defined as follows:

$$\begin{gathered}
\top : \forall \tau \in P. \tau \sqsubseteq \top \\
\tau_1 \wedge \tau_2 : \forall t \in P. t \sqsubseteq \tau_1 \wedge \tau_2 \iff t \sqsubseteq \tau_1 \wedge t \sqsubseteq \tau_2 \\
\tau_1 \Longrightarrow \tau_2 : \forall t \in P. t \sqsubseteq \tau_1 \Longrightarrow \tau_2 \iff t \wedge \tau_1 \sqsubseteq \tau_2
\end{gathered}$$

9. Write a term `and` that takes two Church booleans and returns their conjunction. (How do you change this for `or`?)

10. Write a function `equal` that takes two Church numerals and returns a Church boolean.

11. (a) Write a function `isnil` that takes a Church-encoded list and returns a Church boolean.
    (b) Write a function `head` : $X \to \texttt{CList}_X \to X$ which returns the first argument if the list is empty, and the head of the list otherwise.
    (c) Write a function `tail` : $\texttt{CList}_X \to \texttt{CList}_X$ which returns the empty list when applied to an empty list, and the tail of the list otherwise.

12. (a) Define a predecessor function `pred` : $\mathbf{N} \to \mathbf{N} \to \mathbf{N}$ in PLC on Church numerals. (Mapping 0 to 0 and n to n - 1). *(Hint: use a pair $(n, n-1)$. Is there any other way?).*
    (b) Using your predecessor function defined above, define a subtraction function on Church numerals.
    (c) Approximately how many steps of evaluation (as a function of n) are required to calculate `pred` c_n?