Assignment 2
The Polymorphic λ-Calculus
Sample Solution

15-814: Types and Programming Languages
Frank Pfenning

Due Tue Sep 16, 2025
80 pts

This assignment is due on the above date and it must be submitted electronically on Gradescope. Please use the attached template to typeset your assignment and make sure to include your full name and Andrew ID. For the written problems, you may also submit handwritten answers that have been scanned and are **easily legible**.

Please carefully read the policies on collaboration and credit on the course web pages at `http://www.cs.cmu.edu/~fp/courses/15814-f25/assignments.html`.

You should hand in two files separately:

- `hw02.pdf` with the written answers to the questions.
- `hw02.poly` with the code, where the solutions to the problems are clearly marked and auxiliary code (either from lecture or your own) is included so it passes the LAMBDA checker.

1 Polymorphic Typing

Task 1 (10 pts) Fill in the blanks in the following judgments so that it holds, or indicate there is no way to do so. You do not need to justify your answer or supply a typing derivation, and the types do not need to be "most general" in any sense. As always, feel free to use LAMBDA to check your answers.

(i) $\boxed{\beta\ \mathit{type}} \vdash \forall\alpha.\,\alpha \to \beta\ \mathit{type}$

(ii) $\boxed{\eta\ \mathit{type}, x : \forall\gamma.\,\eta \to \forall\delta.\,\delta,\ y : \eta,\ \beta\ \mathit{type}} \vdash \Lambda\alpha.\,x\,[\alpha \to \alpha]\,y\,[\beta] : \forall\alpha.\,\beta$

(iii) $\cdot \vdash \lambda x.\,x\,[\boxed{u}]\,x\,x : \boxed{u \to u}$ where $u = \forall\alpha.\,\alpha \to \alpha \to \alpha$

(iv) $\alpha\ \mathit{type} \vdash \boxed{\text{not possible}} : \forall\beta.\,\alpha \to \beta$

(v) $x : \forall\alpha.\,(\forall\beta.\,\beta \to \beta) \to \alpha, \gamma\ \mathit{type} \vdash \boxed{\lambda y.\,x\,[\gamma]\,(\Lambda\beta.\lambda z.\,z)} : (\gamma \to \gamma) \to \gamma$

2 Representing Data and Functions

Task 2 (10 pts)

(i) Find a definition of *plus* : $nat \to nat \to nat$ that works in the simply-typed λ-calculus in the sense that we need to instantiate the type $\forall\alpha.\,(\alpha \to \alpha) \to \alpha \to \alpha$ only with a type variable.

(ii) Give a simply-typed definition (in the sense of part (i)) for *times* or conjecture that none exists.

Your answer(s) should be included in the file `hw02.poly`.

> See hw02soln.poly

Task 3 (35 pts) In this exercise we explore the representation of natural numbers in binary form (type *bin*). We can think of them as being generated from the constructors

$$\begin{array}{lcll} \mathsf{b0} & : & \mathit{bin} \to \mathit{bin} & \text{bit 0} \\ \mathsf{b1} & : & \mathit{bin} \to \mathit{bin} & \text{bit 1} \\ \mathsf{e} & : & \mathit{bin} & \text{empty bit string} \end{array}$$

where the least significant bit comes first ("little endian"). We write $\overline{(n)_2}$ for the representation of n in base 2. Here are some examples:

$$\begin{array}{lcl} \overline{(0)_2} & = & \mathsf{e} \\ \overline{(1)_2} & = & \mathsf{b1}\,\mathsf{e} \\ \overline{(2)_2} & = & \mathsf{b0}\,(\mathsf{b1}\,\mathsf{e}) \\ \overline{(6)_2} & = & \mathsf{b0}\,(\mathsf{b1}\,(\mathsf{b1}\,\mathsf{e})) \end{array}$$

To obtain the representation in the *untyped* λ-calculus we *abstract* over all the constructors, so, for example

$$\overline{(6)_2} \;=\; \lambda\mathsf{b0}.\,\lambda\mathsf{b1}.\,\lambda\mathsf{e}.\,\mathsf{b0}\,(\mathsf{b1}\,(\mathsf{b1}\,\mathsf{e}))$$

(i) Give an analogue of the schema of iteration. It should have three clauses, one for each of the constructors.

$$\begin{array}{lcl} f(\mathsf{e}) & = & c \\ f(\mathsf{b0}\ x) & = & g_0\ (f(x)) \\ f(\mathsf{b1}\ x) & = & g_1\ (f(x)) \end{array}$$

(ii) Give a representation of *bin* as a closed type in the polymorphic lambda-calculus.

$$\mathit{bin} = \forall\alpha.\,\alpha \to (\alpha \to \alpha) \to (\alpha \to \alpha) \to \alpha$$

(iii) Give well-typed representations of the constructor functions

$$\begin{array}{lcll} \mathsf{b0} & : & \mathit{bin} \to \mathit{bin} & (\mathsf{b0}\ \overline{(n)_2} = \overline{(2n)_2}) \\ \mathsf{b1} & : & \mathit{bin} \to \mathit{bin} & (\mathsf{b1}\ \overline{(n)_2} = \overline{(2n+1)_2}) \\ \mathsf{e} & : & \mathit{bin} & (\mathsf{e} = \overline{(0)_2}) \end{array}$$

See hw02soln.poly

(iv) Give a well-typed representation of

$$\mathsf{inc} \quad : \quad \mathit{bin} \to \mathit{bin} \quad (\mathsf{inc}\ \overline{(n)_2} = \overline{(n+1)_2})$$

See hw02soln.poly

(v) Provide several tests cases for the increment function.

See hw02soln.poly

Include the answers to parts (ii)–(v) in the file `hw02.poly`.

3 Typing Self-Application

Task 4 (25 pts) We write F for a (mathematical) function from types to types (which is not expressible in the polymorphic λ-calculus but requires system F^ω). A more general family of types (one for each F) for self-application is given by

$$\begin{array}{lcl} u_F & = & \forall\alpha.\,\alpha \to F(\alpha) \\ \omega_F & : & u_F \to F(u_F) \\ \omega_F & = & \lambda x.\,x\ [u_F]\ x \end{array}$$

We recover the type from this lecture with $F = \Lambda\alpha.\,\alpha$. You may want to verify the general typing derivation in preparation for the following questions, but you do not need to show it.

(i) Consider $F = \Lambda\alpha.\,\alpha \to \alpha$. In this case $u_F = \mathit{bool}$. Calculate the type and characterize the behavior of ω_F as a function on Booleans.

(ii) Consider $F = \Lambda\alpha.\,(\alpha{\to}\alpha){\to}\alpha$. Calculate u_F, the type of ω_F, and characterize the the behavior of ω_F. Can you relate u_F and ω_F to the types and functions we have considered in the course so far?

(i) Here $u_F = \forall\alpha.\,\alpha \to \alpha \to \alpha$, which as expected is the type of Booleans. The type of ω_F is then $u_F \to u_F \to u_F = (\forall\alpha.\,\alpha \to \alpha \to \alpha) \to (\forall\alpha.\,\alpha \to \alpha \to \alpha) \to (\forall\alpha.\,\alpha \to \alpha \to \alpha)$

- $\omega_F\, \mathit{true} = \lambda x.\, \mathit{true} = \mathit{or\ true}$
- $\omega_F\, \mathit{false} = \lambda x.\, x = \mathit{or\ false}$

So ω_F is like *or* on Booleans (where *or* is the logical disjunction for Booleans, $\lambda x.\, \lambda y.\, x\, [\mathit{bool}]\, \mathit{true}\, y$).

(ii) Here $u_F = \forall\alpha.\alpha \to (\alpha \to \alpha) \to \alpha$, which is essentially the type of *nat*, except with its two arguments swapped in order. The type of ω_F is then $u_F \to (u_F \to u_F) \to u_F = (\forall\alpha.\alpha \to (\alpha \to \alpha) \to \alpha) \to ((\forall\alpha.\alpha \to (\alpha \to \alpha) \to \alpha) \to (\forall\alpha.\alpha \to (\alpha \to \alpha) \to \alpha)) \to (\forall\alpha.\alpha \to (\alpha \to \alpha) \to \alpha)$.

We write $\overline{n}$ below to mean this new natural number representation, $\Lambda\alpha.\, \lambda z.\, \lambda s.\, s^n\, z$.

- $\omega_F\, \overline{0} = \lambda s.\, \overline{0}$
- $\omega_F\, \overline{1} = \lambda s.\, s\, \overline{1}$
- $\omega_F\, \overline{2} = \lambda s.\, s\, s\, \overline{2}$
- $\omega_F\, \overline{n} = \lambda s.\, s^n\, \overline{n}$

So applying ω_F to $\overline{n}$ yields a function which takes a function s over these new naturals, and n times applies s to $\overline{n}$.

See `hw02soln.poly` for the implementations.