Typed λ-calculus: Further Questions

P. B. Levy

University of Birmingham

The exam for this course consists of all exercises in Handouts 2–4, if you haven't done them already, and two additional questions below.

1 Answers To "Concepts and Syntax" Exercises

Here are the answers to the exercises in Section 8 of Handout 1.

Question What integer is

$$\begin{array}{l} \mathsf{let}\ 3\ \mathsf{be}\ x. \\ \mathsf{let}\ \mathsf{inl}\ \lambda y_{\mathbb{Z}}.(x+y)\ \mathsf{be}\ u. \\ \mathsf{let}\ 4\ \mathsf{be}\ x. \\ \mathsf{match}\ u\ \mathsf{as}\ \{\mathsf{inl}\ f.f2, \mathsf{inr}\ f.0\} \end{array}$$

?

Correct answer 5

Plausible but incorrect answer 6

Question What integer is

$$\begin{array}{l} \mathsf{let}\ \lambda x_{\mathbb{Z}}.\ \mathsf{inl}\ \lambda y_{\mathbb{Z}}.(x+y)\ \mathsf{be}\ f. \\ \mathsf{let}\ f0\ \mathsf{be}\ u. \\ \mathsf{match}\ u\ \mathsf{as}\ \{ \\ \quad \mathsf{inl}\ g.\ \mathsf{let}\ f1\ \mathsf{be}\ v.\ \mathsf{match}\ v\ \mathsf{as}\ \{\mathsf{inl}\ h.\ g3, \mathsf{inr}\ h.\ 0\}, \\ \quad \mathsf{inr}\ g.\ 0 \\ \} \end{array}$$

?

Correct answer 3

Plausible but incorrect answer 4. Both these exercises illustrate the idea of *static binding*, meaning that bindings cannot be changed. The incorrect answers, 6 and 4, are not in accordance with our definition of the notation. Unfortunately, Emacs Lisp would give you these answers. That is because it uses *dynamic binding*, meaning that a binding of `x` overwrites any previous binding of `x`.

Question (variant record type) For sets A, B, C, D, E, we define $\alpha(A, B, C, D, E)$ to be the set of tuples

$$\{\langle \#\mathsf{left}, x, y\rangle | x \in A, y \in B\} \cup \{\langle \#\mathsf{right}, x, y, z\rangle | x \in C, y \in D, z \in E\}$$

Now think of α as an operation on types. Inventing a reasonable syntax, given 2 introduction rules and 1 elimination rule for $\alpha(A, B, C, D, E)$.

Answer We invent the syntax

$$\langle \#\mathsf{left}, M, N\rangle \qquad\qquad \langle \#\mathsf{right}, M, N, P\rangle$$

for terms of type $\alpha(A, B, C, D, E)$. And we invent the syntax

$$\texttt{case}\ M\ \texttt{of}\ \{\langle \#\mathsf{left}, \mathtt{x}, \mathtt{y}\rangle.\ N, \langle \#\mathsf{right}, \mathtt{x}, \mathtt{y}, \mathtt{z}\rangle.\ N'\}$$

for pattern-matching a term M of type $\alpha(A, B, C, D, E)$.
The introduction rules are

$$\frac{\Gamma \vdash M : A \quad \Gamma \vdash N : B}{\Gamma \vdash \langle \#\mathsf{left}, M, N\rangle : \alpha(A, B, C, D, E)}$$

$$\frac{\Gamma \vdash M : C \quad \Gamma \vdash N : D \quad \Gamma \vdash P : E}{\Gamma \vdash \langle \#\mathsf{right}, M, N, P\rangle : \alpha(A, B, C, D, E)}$$

The elimination rule is

$$\frac{\Gamma \vdash M : \alpha(A, B, C, D, E) \qquad \Gamma, \mathtt{x} : A, \mathtt{y} : B \vdash N : F \quad \Gamma, \mathtt{x} : C, \mathtt{y} : D, \mathtt{z} : E \vdash N' : F}{\Gamma \vdash \texttt{case}\ M\ \texttt{of}\ \{\langle \#\mathsf{left}, \mathtt{x}, \mathtt{y}\rangle.\ N, \langle \#\mathsf{right}, \mathtt{x}, \mathtt{y}, \mathtt{z}\rangle.\ N'\} : F}$$

Question For sets A, B, C, D, E, F, G, we define $\beta(A, B, C, D, E, F, G)$ to be the set of functions that take

- a sequence of arguments $(\#\mathsf{left}, x, y)$, where $x \in A$ and $y \in B$, to an element of C
- a sequence of arguments $(\#\mathsf{right}, x, y, z)$, where $x \in D$ and $y \in E$ and $z \in F$, to an element of G.

Thus the first argument is always a tag, indicating how many other arguments there are, what their type is, and what the type of the result should be.
Now think of β as an operation on types. Inventing a reasonable syntax, give 1 introduction rule and 2 elimination rules for $\beta(A, B, C, D, E, F, G)$.

Answer We invent the syntax

$$\lambda\{(\#\mathsf{left}, \mathtt{x}, \mathtt{y})\ .M, (\#\mathsf{right}, \mathtt{x}, \mathtt{y}, \mathtt{z}).\ M'\}$$

for something of type $\beta(A, B, C, D, E, F, G)$. And we invent the syntax

$$M(\#\mathsf{left}, N, N') \qquad\qquad M(\#\mathsf{right}, N, N', N'')$$

for applying a term M of type $\beta(A, B, C, D, E, F, G)$.
The introduction rule is

$$\frac{\Gamma, \mathtt{x} : A, \mathtt{y} : B \vdash M : C \quad \Gamma, \mathtt{x} : D, \mathtt{y} : E, \mathtt{z} : F \vdash M' : G}{\Gamma \vdash \lambda\{(\#\mathsf{left}, \mathtt{x}, \mathtt{y})\ .M, (\#\mathsf{right}, \mathtt{x}, \mathtt{y}, \mathtt{z}).\ M'\} : \beta(A, B, C, D, E, F, G)}$$

The elimination rules are

$$\frac{\Gamma \vdash M : \beta(A, B, C, D, E, F, G) \quad \Gamma \vdash N : A \quad \Gamma \vdash N' : B}{\Gamma \vdash M(\#\mathsf{left}, N, N') : C}$$

$$\frac{\Gamma \vdash M : \beta(A, B, C, D, E, F, G) \quad \Gamma \vdash N : D \quad \Gamma \vdash N' : E \quad \Gamma \vdash N'' : F}{\Gamma \vdash M(\#\mathsf{right}, N, N', N'') : G}$$

2 Question on Pure λ-calculus

This question is about the pure (i.e. no imperative features) simply typed λ-calculus.

In this language, a *syntactic isomorphism* from A to B consists of a term $\mathtt{x} : A \vdash M : B$ and a term $\mathtt{y} : B \vdash N : A$ such that the equations

$$\mathtt{x} : A \vdash N[M/\mathtt{x}] = \mathtt{x} : B$$
$$\mathtt{y} : B \vdash M[N/\mathtt{x}] = \mathtt{y} : A$$

are provable in the equational theory. (NB This definition, as it stands, is not suitable for λ-calculus with imperative features.) Construct syntactic isomorphisms

$$(A + B) + C \cong A + (B + C)$$
$$(A \times B) \to C \cong A \to (B \to C)$$
$$(A + B) \to C \cong (A \to C) \times (B \to C)$$

3 Question on $\boldsymbol{\lambda}$-calculus with Imperative Features

CELL is a storage cell (piece of computer memory) that stores an integer.

Consider call-by-value λ-calculus, without divergence or printing, but with the facility to write to and read from CELL.

- `CELL` $:= M.\ N$, where M is an integer expression. To evaluate this, first evaluate M, the put the answer in CELL (overwriting whatever was there previously), then evaluate N.
- `read CELL as x`. N. To evaluate this, define `x` to be whatever is currently in CELL, then evaluate N.

We write $s, M \Downarrow s', T$ to mean that if M (a closed term) is evaluated at a time when CELL contains s (an integer), then it evaluates to T (a terminal term, with the same type as M) with CELL then containing s' (an integer). Give an inductive definition of the relation $\Downarrow$.