Functional Programming Practicals

0 Getting started

We will be using GHCi for the practicals. To run GHCi, simply open a terminal window and type '`ghci`'. One typically uses a text editor to write or edit a Haskell script, saves that to disk, and loads it into GHCi. To load a script, it is helpful if you run GHCi from the directory containing the script. You can simply give the name of the script file as a parameter to the command `ghci`. Or, within GHCi, you can type '`:l`' followed by the name of the script to load, and '`:r`' with no parameter to reload the file previously loaded.

There are model answers for most of the exercises, and in some cases skeletons of a solution to save you from having to type in what is provided to start with. For instructions on how to find those, see the practical assistant.

> Answers are given in these coloured boxes.

Jeremy Gibbons
(with thanks to Colin Runciman, Jon Hill, Ralf Hinze, Nicolas Wu)
September 2014

1 Basic definitions

1. Define the following numeric functions:

   - a function *square* that squares its argument, then a function *quad* that raises its argument to the fourth power using *square*;

     Function *quad* raises its argument to the fourth power:

     $$quad :: Integer \to Integer$$
     $$quad\ x = square\ (square\ x)$$

     It uses function *square* from the lectures:

     $$square :: Integer \to Integer$$
     $$square\ x = x * x$$

   - a function *larger* that returns the larger of its two arguments;

     Function *larger* returns the larger of its two arguments:

     $$larger :: (Integer, Integer) \to Integer$$
     $$larger\ (x, y) = \textbf{if}\ x > y\ \textbf{then}\ x\ \textbf{else}\ y$$

     or alternatively,

     $$larger :: (Integer, Integer) \to Integer$$
     $$larger\ (x, y)$$
     $$\quad | \ x > y \quad = x$$
     $$\quad | \ otherwise = y$$

     (Actually, the type in both cases is the more general $Ord\ a \Rightarrow (a, a) \to a$, as you can verify by omitting the type declaration and asking for the inferred type with "`:t larger`". This means that it works for any ordered type a, not just for *Integer*s. We will say more about this overloading mechanism later.)

- a function for computing the area of a circle with a given radius (use the type *Double*). (Hint: the formula for calculating the area A of a circle with a radius r is $A = \pi r^2$, where π is called *pi* in Haskell.)

  Function *area* computes the area of a circle, given its radius:

  ```
  area :: Float → Float
  area r = pi * r * r
  ```

2. Here is a script of function definitions:

   ```
   add :: Integer → Integer → Integer
   add x y     = x + y

   double :: Integer → Integer
   double x    = x + x

   first :: Integer → Integer → Integer
   first x y   = x

   cond :: Bool → Integer → Integer → Integer
   cond x y z = if x then y else z

   twice :: (Integer → Integer) → Integer → Integer
   twice f x   = f (f x)

   infinity :: Integer
   infinity    = infinity + 1
   ```

   (The function *twice* is an example of a higher-order function, which takes another function as one of its arguments. Although we haven't studied higher-order functions yet, just follow the reduction rules.)

Give the applicative- and normal-order reduction sequences for the following expressions:

- *first* 42 (*double* (*add* 1 2))

  In applicative order:

  $$\begin{aligned} & \mathit{first}\ (42, \mathit{double}\ (\mathit{add}\ 1\ 2)) \\ = {} & \mathit{first}\ (42, \mathit{double}\ (1 + 2)) \\ = {} & \mathit{first}\ (42, \mathit{double}\ 3) \\ = {} & \mathit{first}\ (42, 3 + 3) \\ = {} & \mathit{first}\ (42, 6) \\ = {} & 42 \end{aligned}$$

  and in normal order:

  $$\begin{aligned} & \mathit{first}\ (42, \mathit{double}\ (\mathit{add}\ 1\ 2)) \\ = {} & 42 \end{aligned}$$

  terminating straight away, rather than in several steps.

- *first* 42 (*double* (*add* 1 *infinity*))

  In applicative order:

  $$\begin{aligned} & \mathit{first}\ (42, \mathit{double}\ (\mathit{add}\ 1\ \mathit{infinity})) \\ = {} & \mathit{first}\ (42, \mathit{double}\ (1 + \mathit{infinity})) \\ = {} & \mathit{first}\ (42, \mathit{double}\ (1 + (1 + \mathit{infinity}))) \\ = {} & \ldots \end{aligned}$$

  which diverges; and in normal order:

  $$\begin{aligned} & \mathit{first}\ (42, \mathit{double}\ (\mathit{add}\ 1\ \mathit{infinity})) \\ = {} & 42 \end{aligned}$$

  terminating straight away, rather than diverging.

- *first infinity* (*double* (*add* 1 2))

  In applicative order:

  $$\begin{aligned} & \mathit{first}\ (\mathit{infinity}, \mathit{double}\ (\mathit{add}\ 1\ 2)) \\ = {} & \mathit{first}\ (1 + \mathit{infinity}, \mathit{double}\ (\mathit{add}\ 1\ 2)) \\ = {} & \ \ldots \end{aligned}$$

  and in normal order:

  $$\begin{aligned} & \mathit{first}\ (\mathit{infinity}, \mathit{double}\ (\mathit{add}\ 1\ 2)) \\ = {} & \mathit{infinity} \\ = {} & 1 + \mathit{infinity} \\ = {} & \ \ldots \end{aligned}$$

  Neither terminates—normal-order isn't magic!

- *add* (*cond True* 42 (1 + *infinity*)) 4

  In applicative order:

  $$\begin{aligned} & \mathit{add}\ (\mathit{cond}\ (\mathit{True}, 42, 1 + \mathit{infinity}))\ 4 \\ = {} & \mathit{add}\ (\mathit{cond}\ (\mathit{True}, 42, 1 + (1 + \mathit{infinity})))\ 4 \\ = {} & \ \ldots \end{aligned}$$

  which diverges; and in normal order:

  $$\begin{aligned} & \mathit{add}\ (\mathit{cond}\ (\mathit{True}, 42, 1 + \mathit{infinity}))\ 4 \\ = {} & \mathit{cond}\ (\mathit{True}, 42, 1 + \mathit{infinity}) + 4 \\ = {} & (\textbf{if}\ \mathit{True}\ \textbf{then}\ 42\ \textbf{else}\ 1 + \mathit{infinity}) + 4 \\ = {} & 42 + 4 \\ = {} & 46 \end{aligned}$$

  which terminates.

- *twice double* (*add* 1 2)

  In applicative order:

  $$
  \begin{aligned}
  & \mathit{twice}\ (\mathit{double}, \mathit{add}\ 1\ 2) \\
  = {} & \mathit{twice}\ (\mathit{double}, 1 + 2) \\
  = {} & \mathit{twice}\ (\mathit{double}, 3) \\
  = {} & \mathit{double}\ (\mathit{double}\ 3) \\
  = {} & \mathit{double}\ (3 + 3) \\
  = {} & \mathit{double}\ 6 \\
  = {} & 6 + 6 \\
  = {} & 12
  \end{aligned}
  $$

  and in normal order:

  $$
  \begin{aligned}
  & \mathit{twice}\ (\mathit{double}, \mathit{add}\ 1\ 2) \\
  = {} & \mathit{double}\ (\mathit{double}\ (\mathit{add}\ 1\ 2)) \\
  = {} & \mathit{double}\ (\mathit{add}\ 1\ 2) + \mathit{double}\ (\mathit{add}\ 1\ 2) \\
  = {} & (\mathit{add}\ 1\ 2 + \mathit{add}\ 1\ 2) + \mathit{double}\ (\mathit{add}\ 1\ 2) \\
  = {} & ((1 + 2) + \mathit{add}\ 1\ 2) + \mathit{double}\ (\mathit{add}\ 1\ 2) \\
  = {} & (3 + \mathit{add}\ 1\ 2) + \mathit{double}\ (\mathit{add}\ 1\ 2) \\
  = {} & (3 + (1 + 2)) + \mathit{double}\ (\mathit{add}\ 1\ 2) \\
  = {} & (3 + 3) + \mathit{double}\ (\mathit{add}\ 1\ 2) \\
  = {} & 6 + \mathit{double}\ (\mathit{add}\ 1\ 2) \\
  = {} & 6 + (\mathit{add}\ 1\ 2 + \mathit{add}\ 1\ 2) \\
  = {} & \ldots \{\mathit{repeated\ steps}\} \\
  = {} & 6 + 6 \\
  = {} & 12
  \end{aligned}
  $$

  which gives the same result, but takes rather longer.

- $twice\ (add\ 1)\ 0$

  In applicative order:

  $$\begin{aligned} & twice\ (add\ 1, 0) \\ = & \ add\ 1\ (add\ 1\ 0) \\ = & \ add\ 1\ (1 + 0) \\ = & \ add\ 1\ 1 \\ = & \ 1 + 1 \\ = & \ 2 \end{aligned}$$

  and in normal order:

  $$\begin{aligned} & twice\ (add\ 1, 0) \\ = & \ add\ 1\ (add\ 1\ 0) \\ = & \ 1 + add\ 1\ 0 \\ = & \ 1 + (1 + 0) \\ = & \ 1 + 1 \\ = & \ 2 \end{aligned}$$

  which is not very different.

Note: There is not a mistake in the last expression; all you need to know for the time being is that a function application that doesn't have enough arguments is already in normal form. Just follow the rules when reducing the expression.

3. Give a reduction sequence for *fact* 3, where the factorial function *fact* is as defined in the lectures:

$$
\begin{aligned}
&\mathit{fact} :: \mathit{Integer} \to \mathit{Integer} \\
&\mathit{fact}\ 0 = 1 \\
&\mathit{fact}\ n = n * \mathit{fact}\ (n-1)
\end{aligned}
$$

With normal-order reduction, we have:

$$
\begin{aligned}
& \mathit{fact}\ 3 \\
=\ & 3 * \mathit{fact}\ (3-1) \\
=\ & 3 * \mathit{fact}\ 2 \\
=\ & 3 * (2 * \mathit{fact}\ (2-1)) \\
=\ & 3 * (2 * \mathit{fact}\ 1) \\
=\ & 3 * (2 * (1 * \mathit{fact}\ (1-1))) \\
=\ & 3 * (2 * (1 * \mathit{fact}\ 0)) \\
=\ & 3 * (2 * (1 * 1)) \\
=\ & 3 * (2 * 1) \\
=\ & 3 * 2 \\
=\ & 6
\end{aligned}
$$

How would it differ with applicative-order reduction?