Functional Programming
Exercise 1: Functions, Values, and Types

Shin-Cheng Mu

2012 Formosan Summer School on Logic, Language, and Computation
Aug 27 – Sep 7, 2012

Functions

1. Define a function that computes the area of a circle with given radius r (you may use 22/7 as an approximation to π).

2. Recall the definition of *curry*:

   $$\# \textbf{ let } \mathit{curry}\ f\ x\ y\ =\ f\ (x, y)$$
   $$\textbf{val } \mathit{curry}\ :\ ('a * 'b \to 'c) \to 'a \to 'b \to 'c\ =\ \langle \mathit{fun} \rangle$$

   Define $\mathit{uncurry} : ('a \to 'b \to 'c) \to ('a * 'b \to c)$. Prove that

   $$\mathit{curry}\ (\mathit{uncurry}\ f) = f$$
   $$\mathit{uncurry}\ (\mathit{curry}\ f) = f$$

Playing Around With Lists

The purpose of this exercise is to familiarise one with list processing and the "combinator" style of programming, in which programs are composed from smaller parts.

1. The list is traditionally an important datatype in functional languages. Start the OCaml interpreter, type in the following expression:

   $$\# [1; 2; 3; 4];;$$
   $$-\ :\ \mathit{int\ list}\ =\ [1; 2; 3; 4]$$

   OCaml says that $[1; 2; 3; 4]$ is has type *int list* — a list whose elements are integers. In OCaml, all elements in a list must be of the same type.

   Guess the type of the following lists, before finding out the answer in OCaml.

   - $[\mathit{true}; \mathit{false}; \mathit{true}]$.
   - $[[1; 2]; [\,]; [6; 7]]$.
   - $[(+); (-); (/)]$.
   - $[\,]$.
   - $[[\,]]$.

2. **Take**. Download the file `"Utils.ml"` from the course website and save it in your current working directory. Load the file by issueing the command

   \# *use* `"Utils.ml"`

   We have defined some functions that might be useful later.

   Try the following expressions:

- *take* 3 [0; 1; 2; 3; 4]
- *take* 0 [0; 1; 2; 3; 4]
- *take* 4 [0; 1; 2]

Describe in words what the function *take* does.

3. **Drop**. Try the following expressions:

   - *drop* 3 [0; 1; 2; 3; 4]
   - *drop* 0 [0; 1; 2; 3; 4]
   - *drop* 4 [0; 1; 2]

   Describe in words what the function *drop* does.

4. **Length**. The function *length* has type $'a\ list \to int$. Try some inputs, and describe in words what this function does.

5. **Append** The operator (@) has type $'a\ list \to 'a\ list \to 'a\ list$.

   1. Try the following expressions:
      - [0; 1; 2]@[3; 4].
      - [0; 1; 0]@[1; 0].

      Describe in words what the operator (@) does.
   2. Which of the following expressions are type correct? For the type-correct expressions, what do they evaluate to?
      - []@[1; 2; 3]@[4]
      - [[ ]]@[1; 2; 3]
      - [[ ]]@[ ]
      - [ ]@[[ ]]
      - [ ]@[ ]
   3. Can you think of a property that relates *take*, *drop* and (@)?

6. Strings and lists of characters are different types in OCaml (unlike in Haskell). We have defined functions *explode* and *implode* in `Utils.ml` that perform the version. Try

   - *explode* `"functional programming"`
   - *implode* [`'f'`; `'u'`; `'n'`; `'c'`; `'t'`; `'i'`; `'o'`; `'n'`]

7. Define function $rotate : int \to 'a\ list \to 'a\ list$ such that $rotates\, n\, xs$, when $0 \leq n \leq length\, xs$, rotates xs leftwards by n positions. For example:

   - $rotate\, 2\, [0; 1; 2; 3; 4; 5] = [2; 3; 4; 5; 0; 1]$
   - *implode* (*rotate* 3 (*explode* `"flolac"`)) = `"lacflo"`

   Hint: use *take*, *drop*, and (@).

8. We have also defined a function $fromTo : int \to int \to int\ list$ in `Utils.ml`. Knowing its type, try some inputs, and describe in words what this function does.

9. In the OCaml toplevel, issue the coomand `open List`, to gain access to some more functions on lists. The function *combine* has type $'a\ list \to 'b\ list \to 'a * 'b\ list$. Try

   - *combine* [0; 1; 2; 3] (*explode* `"abcd"`)
   - *combine* [0; 1; 2] (*explode* `"abcd"`)

and describe in words what this function does.

10. Now we will take a look at some *higher-order functions* — functions that takes functions as inputs or returns functions. The first candidate is *filter*, having type $('a \to bool) \to 'a\,list \to 'a\,list$. Try
    - $\mathit{filter}\ \mathit{is\_even}\ [0;1;2;3;4]$
    - $\mathit{filter}\ (\mathbf{fun}\,x \to x\ \mathit{mod}\ 3 = 0)\ [0;1;2;3;4]$

    Describe in words what *filter* does.

11. Try the function *map*:
    - $\mathit{map}\ \mathit{not}\ [\mathit{true};\mathit{true};\mathit{false}]$
    - $\mathit{map}\ (\mathbf{fun}x \to x\ \mathit{mod}\ 4)\ (\mathit{fromTo}\ (-10)\ 10)$

    Answer the questions:
    - What should the type of *map* be?
    - Describe in words what *map* does.

12. Define $\mathit{count}\ :\ ('a \to bool) \to 'a\,list \to int$ such that $\mathit{count}\ p\ xs$ returns the number of elements in xs that satisfies p.

13. Define $\mathit{index}\ :\ 'a\,list \to (int * 'a)\,list$ such that $\mathit{index}\ xs$ labels each element in xs with its index. For example,

    $$\mathit{index}\ (\mathit{explode}\ \texttt{"flolac"})\ =\ [(0,'f');(1,'l');(2,'o');(3,'l');(4,'a');(5,'c')]$$

14. Define $\mathit{positions}\ :\ ('a \to bool) \to 'a\,list \to int\,list$ such that $\mathit{positions}\ p\ xs$ returns the indexes of elements in xs that satisfies p. For example

    $$\mathit{positions}\ \mathit{is\_even}\ [2;4;5;3;6]\ =\ [0;1;4]$$

Types

1. Suppose f and g have the following types:

    $$f : int \to int$$
    $$g : int \to int \to int$$

    Let h be defined by

    $$h\ x\ y =\ f\ (g\ x\ y)$$

    1. What is the type of h?
    2. Which, if any, of the following statements is true?

    $$\begin{aligned} h\ &= f \ll g \\ h\ x\ &= f \ll (g\ x) \\ h\ x\ y &= (f \ll g)\ x\ y \end{aligned}$$

2. Give suitable polymorphic type assignments for the following functions:

    $$\begin{aligned} &\mathbf{let}\ \mathit{const}\ x\ y &&= x \\ &\mathbf{let}\ \mathit{subst}\ f\ g\ x &&= f\ x\ (g\ x) \\ &\mathbf{let}\ \mathit{apply}\ f\ x &&= f\ x \\ &\mathbf{let}\ \mathit{flip}\ f\ x\ y &&= f\ y\ x \end{aligned}$$

3. Define a function *swap* such that:

$$\mathit{flip}\ (\mathit{curry}\ f) = \mathit{curry}\ (f \ll \mathit{swap})$$

   for all $f : {}'a * {}'b \rightarrow {}'c$.
   Hint: there are at least two ways to construct *swap*:

   1. use equational reasoning, construct a definition of *swap* such that both sides simply to the same expression, or
   2. deduce its type, guess a definition using the type, and prove the equality above.

4. Can you find polymorphic type assignments for the following functions?

$$\begin{array}{l} \textbf{let}\ \mathit{strange}\ f\ g = g\ (f\ g) \\ \textbf{let}\ \mathit{stranger}\ f\ \ = f\ f \end{array}$$