Functional Programming
Practicals

Shin-Cheng Mu

1 Functions

1. Define a function $\mathit{even} :: \mathit{Int} \rightarrow \mathit{Bool}$ that determines whether the input is an even number. You may use the following functions:

$$\begin{aligned} \mathit{mod} &:: \mathit{Int} \rightarrow \mathit{Int} \rightarrow \mathit{Int} \quad , \\ (==) &:: \mathit{Int} \rightarrow \mathit{Int} \rightarrow \mathit{Bool} \quad . \end{aligned}$$

(Types of the functions written above are not in their most general form.)

Solution:

$$\begin{aligned} &\mathit{even} && :: \mathit{Int} \rightarrow \mathit{Bool} \\ &\mathit{even}\ n && = \ n \ \text{`}\mathit{mod}\text{`}\ 2 == 0 \quad . \end{aligned}$$

2. Define a function that computes the area of a circle with given radius r (using $22/7$ as an approximation to π). The return type of the function might be *Double*.

Solution:

$$\begin{aligned} &\mathit{area} && :: \mathit{Double} \rightarrow \mathit{Double} \\ &\mathit{area}\ r && = \ r \times r \times (22/7) \quad . \end{aligned}$$

3. Type in the definition of *smaller* into your working file. Then try the following:
   (a) In GHCi, type `:t smaller` to see the type of *smaller*.
   (b) Try applying it to some arguments, e.g. $\mathit{smaller}\ 3\ 4$, $\mathit{smaller}\ 3\ 1$.
   (c) In your working file, define a new function $\mathit{st3} = \mathit{smaller}\ 3$.
   (d) Find out the type of *st3* in GHCi. Try $\mathit{st3}\ 4$, $\mathit{st3}\ 1$. Explain the results you see.

4. Type in the definition of *square* in your working file.
   (a) Define a function $\mathit{quad} :: \mathit{Int} \rightarrow \mathit{Int}$ such that $\mathit{quad}\ x$ computes x^4.
   (b) Type in this definition into your working file. Describe, in words, what this function does.

$$\begin{aligned} &\mathit{twice} && :: (a \rightarrow a) \rightarrow (a \rightarrow a) \\ &\mathit{twice}\ f\ x && = f\ (f\ x) \quad . \end{aligned}$$

(c) Define *quad* using *twice*.

5. Replace the previous *twice* with this definition:

$$\begin{aligned} &twice &&:: (a \to a) \to (a \to a) \\ &twice\ f &&= f \cdot f \quad . \end{aligned}$$

   (a) Does *quad* still behave the same?

   (b) Explain in words what this operator $(\cdot)$ does.

6. Let the following identifiers have type:

$$\begin{aligned} &f &&:: Int \to Char \\ &g &&:: Int \to Char \to Int \\ &h &&:: (Char \to Int) \to Int \to Int \\ &x &&:: Int \\ &y &&:: Int \\ &c &&:: Char \end{aligned}$$

   Which of the following expressions are type correct?

   1. $(g \cdot f)\ x\ c$
   2. $(g\ x \cdot f)\ y$
   3. $(h \cdot g)\ x\ y$
   4. $(h \cdot g\ x)\ c$
   5. $h \cdot g\ x\ c$

   You may type the expressions into Haskell and see whether they type check. To define f, for example, include the following in your working file:

$$\begin{aligned} &f &&:: Int \to Char \\ &f &&= undefined \end{aligned}$$

   However, it is better if you can explain why the answers are as they are.

2 Products and Sums

1. In GHCi, issue the command

```
let x = ((1,'a'), True)
```

   This defines a new symbol x, with value $((1, \texttt{'}a\texttt{'}), \mathsf{True})$.

   (a) Find out the type of x by a GHCi command.

   (b) How do you extract the 1 in x? Type an expression $\ldots\ x$ into GHCi such that the result is 1.

   (c) Try to extract $\texttt{'}a\texttt{'}$ and True from x too.

2. Define a function $swap :: (a, b) \to (b, a)$ that, as the name and type suggests, swaps the components

   (a) Define *swap* using pattern matching: $swap\ (x, y) = \ldots$.

   (b) Define *swap* using *fst* and *snd*: $swap\ x = \ldots$.

   (c) Define *swap* using **case**.

Solution:

$$\begin{aligned}
\mathit{swap}\ (x, y) &= (y, x) \quad ; \\
\mathit{swap}\ x &= (\mathit{snd}\ x, \mathit{fst}\ x) \quad ; \\
\mathit{swap}\ x &= \mathbf{case}\ x\ \mathbf{of}\ (x, y) \rightarrow (y, x) \quad .
\end{aligned}$$

3. Define a function $\mathit{half} :: \mathit{Int} \rightarrow \mathit{Either}\ \mathit{Int}\ \mathit{Int}$ such that
   - if n is even, $\mathit{half}\ n$ returns $\mathsf{Left}\ k$ with $2 \times k = n$;
   - if n is odd, $\mathit{half}\ n$ returns $\mathsf{Right}\ k$ with $2 \times k + 1 = n$.

   You may use the function div. Find out what it does by youself.

Solution:

$$\begin{aligned}
&\mathit{half} \quad :: \mathit{Int} \rightarrow \mathit{Either}\ \mathit{Int}\ \mathit{Int} \\
&\mathit{half}\ n \mid \mathit{even}\ n = \mathsf{Left}\ (n\ \text{`}\mathit{div}\text{`}\ 2) \\
&\qquad\ \ \mid \mathit{odd}\ n \ \ = \mathsf{Right}\ (n\ \text{`}\mathit{div}\text{`}\ 2) \quad .
\end{aligned}$$

4. What are the types of the following expressions?
   (a) $\lambda x \rightarrow (\mathit{snd}\ x, \mathit{fst}\ x)$.
   (b) $\lambda f\ x \rightarrow f\ x\ x$.
   (c) Define:

   $$\begin{aligned}
   \mathit{myEither}\ f\ g\ x = \mathbf{case}\ x\ &\mathbf{of} \\
   &\mathsf{Left}\ y \rightarrow f\ y \\
   &\mathsf{Right}\ z \rightarrow g\ z \quad .
   \end{aligned}$$

   What is the type of $\mathit{myEither}$?[1]
   (d) $\lambda f\ x\ y \rightarrow f\ (\mathit{fst}\ y)\ x$.
   (e) $\lambda f\ x\ y \rightarrow \mathit{fst}\ (f\ y\ x)$.
   (f) $\lambda x\ y \rightarrow x$.
   (g) $\lambda f\ g\ x \rightarrow f\ x\ (g\ x)$.

Solution:

(a) $(a, b) \rightarrow (b, a)$.

(b) $(a \rightarrow a \rightarrow b) \rightarrow a \rightarrow b$.

(c) $(a \rightarrow c) \rightarrow (b \rightarrow c) \rightarrow \mathit{Either}\ a\ b \rightarrow c$.

(d) $(a \rightarrow b \rightarrow c) \rightarrow b \rightarrow (a, d) \rightarrow c$.

(e) $(a \rightarrow b \rightarrow (c, d)) \rightarrow b \rightarrow a \rightarrow c$.

(f) $(a \rightarrow b \rightarrow a)$.

(g) $(a \rightarrow b \rightarrow c) \rightarrow (a \rightarrow b) \rightarrow a \rightarrow c$.

[1] There is such a function called *either*, which is sometimes quite convenient.

3 Inductively Defined Functions on Lists

1. Define a function $\mathit{fstEven} :: [\mathit{Int}] \to \mathit{Int}$ that returns the first even number of the input list.

> **Solution:**
>
> $$\begin{array}{lll} \mathit{fstEven} & :: & [\mathit{Int}] \to \mathit{Int} \\ \mathit{fstEven}\ (x : \mathit{xs}) & \mid \mathit{even}\ x & = \mathsf{True} \\ & \mid \textbf{otherwise} & = \mathit{fstEven}\ \mathit{xs} \quad . \end{array}$$

2. Define a function $\mathit{hasZero} :: [\mathit{Int}] \to \mathit{Bool}$ that returns True if and only if there is a 0 in the input list.

> **Solution:**
>
> $$\begin{array}{lll} \mathit{hasZero} & :: & [\mathit{Int}] \to \mathit{Bool} \\ \mathit{hasZero}\ [\,] & = & \mathsf{False} \\ \mathit{hasZero}\ (0 : \mathit{xs}) & = & \mathsf{True} \\ \mathit{hasZero}\ (x : \mathit{xs}) & = & \mathit{hasZero}\ \mathit{xs} \quad . \end{array}$$

3. Define a function *myLast* that takes a list and returns the last (rightmost) element.
   (a) Let the type be $\mathit{myLast} :: [a] \to a$. Define *myLast*.

> **Solution:**
>
> $$\begin{array}{lll} \mathit{myLast} & :: & [a] \to a \\ \mathit{myLast}\ [x] & = & x \\ \mathit{myLast}\ (x : \mathit{xs}) & = & \mathit{myLast}\ \mathit{xs} \quad . \end{array}$$

   (b) What happens in the previous definition of the input list is empty?
   (c) Define $\mathit{myLast} :: [a] \to \mathit{Maybe}\ a$, which returns Nothing if the list is empty.

> **Solution:**
>
> $$\begin{array}{lll} \mathit{myLast} & :: & [a] \to \mathit{Maybe}\ a \\ \mathit{myLast}\ [\,] & = & \mathsf{Nothing} \\ \mathit{myLast}\ [x] & = & \mathsf{Just}\ x \\ \mathit{myLast}\ (x : \mathit{xs}) & = & \mathit{myLast}\ \mathit{xs} \quad . \end{array}$$

4. Define a function *pos* such that *pos x xs* looks for x in *xs* and returns its position. For example, *find* 'a' "abc" yields 0, and *find* 'a' "bac" yields 1.
   (a) Let the type be $\mathit{pos} :: \mathit{Eq}\ a \Rightarrow a \to [a] \to \mathit{Int}$. In your definition, what happens if x is not in the list?

Solution:

$$
\begin{array}{lll}
pos & :: Eq\ a \Rightarrow a \to [a] \to Int & \\
pos\ x\ (y : xs) & \mid x == y & = \ 0 \\
& \mid \textbf{otherwise} & = \ 1 + pos\ x\ xs \quad .
\end{array}
$$

(b) Let the type be $pos :: Eq\ a \Rightarrow a \to [a] \to Maybe\ Int$, such that $pos\ x\ xs$ returns $\mathsf{Nothing}$ if x is not in the list.

Solution:

$$
\begin{array}{lll}
pos & :: Eq\ a \Rightarrow a \to [a] \to Maybe\ Int & \\
pos\ x\ [\,] & = \ \mathsf{Nothing} & \\
pos\ x\ (y : xs) & \mid x == y & = \ \mathsf{Just}\ 0 \\
& \mid \textbf{otherwise} & = \ \textbf{case}\ pos\ x\ xs\ \textbf{of} \\
& & \quad \mathsf{Just}\ i \to \mathsf{Just}(1 + i) \\
& & \quad \mathsf{Nothing} \to \mathsf{Nothing} \quad .
\end{array}
$$

5. Define $myConcat :: [[a]] \to [a]$ such that, for example $myConcat\ [[1, 2, 3], [], [4], [5, 6]] = [1, 2, 3, 4, 5, 6]$. **Hint**: use $(+\!\!+)$.

Solution:

$$
\begin{array}{ll}
myConcat & :: [[a]] \to [a] \\
myConcat\ [\,] & = \ [\,] \\
myConcat\ (xs : xss) & = \ xs +\!\!+ myConcat\ xss \quad .
\end{array}
$$

6. Define $double :: [a] \to [a]$ such that, for example, $double\ [1, 2, 3] = [1, 1, 2, 2, 3, 3]$.

Solution:

$$
\begin{array}{ll}
double & :: [a] \to [a] \\
double\ [\,] & = \ [\,] \\
double\ (x : xs) & = \ x : x : double\ xs \quad .
\end{array}
$$

7. Define $interleave :: [a] \to [a] \to [a]$ such that, for example, $interleave\ [1, 2, 3, 4]\ [5, 6, 7] = [1, 5, 2, 6, 3, 7, 4]$.

Solution:

$$
\begin{array}{ll}
interleave & :: \ [a] \to [a] \to [a] \\
interleave\ [\,]\ ys & = \ ys \\
interleave\ (x : xs)\ ys & = \ x : interleave\ ys\ xs \quad .
\end{array}
$$

8. Define $splitLR :: [Either\ a\ b] \rightarrow ([a],[b])$ such that, for example:

$$splitLR\ [\mathsf{Left}\ 1, \mathsf{Left}\ 3, \mathsf{Right}\ \texttt{'}a\texttt{'}, \mathsf{Left}\ 2, \mathsf{Right}\ \texttt{'}b\texttt{'}]\ =\ ([1,3,2], \texttt{"}\mathrm{ab}\texttt{"})\ .$$

Solution:

$$\begin{array}{lll} splitLR & :: & [Either\ a\ b] \rightarrow ([a],[b]) \\ splitLR\ [\,] & = & ([\,],[\,]) \\ splitLR\ (x:xs) & = & \textbf{case}\ x\ \textbf{of} \\ & & \quad \mathsf{Left}\ y \rightarrow (y:ys, zs) \\ & & \quad \mathsf{Right}\ z \rightarrow (ys, z:zs)\ , \\ \quad \textbf{where}\ (ys,zs) & = & splitLR\ xs\ . \end{array}$$

9. Define a function $fan :: a \rightarrow [a] \rightarrow [[a]]$ such that $fan\ x\ xs$ inserts x into the 0th, 1st… nth positions of xs, where n is the length of xs. For example:

$$fan\ 5\ [1,2,3,4] = [[5,1,2,3,4],[1,5,2,3,4],[1,2,5,3,4],[1,2,3,5,4],[1,2,3,4,5]]\ .$$

Solution:

$$\begin{array}{lll} fan & :: & a \rightarrow [a] \rightarrow [[a]] \\ fan\ x\ [\,] & = & [[x]] \\ fan\ x\ (y:xs) & = & (x:y:xs) : map\ (y:)\ (fan\ x\ xs)\ . \end{array}$$

10. Define $perms :: [a] \rightarrow [[a]]$ that returns all permutations of the input list. For example:

$$perms\ [1,2,3] = [[1,2,3],[2,1,3],[2,3,1],[1,3,2],[3,1,2],[3,2,1]]\ .$$

Solution:

$$\begin{array}{lll} perms & :: & [a] \rightarrow [[a]] \\ perms\ [\,] & = & [[\,]] \\ perns\ (x:xs) & = & concat\ (map\ (fan\ x)\ (perms\ xs))\ . \end{array}$$

11. Try to define functions *inits* and *tails* yourself, and make sure you understand them. Recall that $inits\ [1,2,3] = [[\,],[1],[1,2],[1,2,3]]$, and $tails\ [1,2,3] = [[1,2,3],[2,3],[3],[\,]]$.

4 Inductively Defined Functions on Natural Numbers

1. Define $mul :: \mathbb{N} \rightarrow \mathbb{N} \rightarrow \mathbb{N}$ such that $mul\ m\ n = m \times n$, by induction on natural number, using addition $(+)$.

Solution:

$$
\begin{aligned}
&\mathit{mul} &&::\ \mathbb{N} \to \mathbb{N} \to \mathbb{N}\\
&\mathit{mul}\ 0\ n &&=\ 0\\
&\mathit{mul}\ (\mathbf{1}{+}\ m)\ n &&=\ n + \mathit{mul}\ m\ n \quad .
\end{aligned}
$$

2. Define $\mathit{myMin} :: \mathbb{N} \to \mathbb{N} \to \mathbb{N}$ that returns the smaller of its two arguments. There is a built-in operator (min) for this, but try defining it inductively on natural numbers.

Solution:

$$
\begin{aligned}
&\mathit{myMin} &&::\ \mathbb{N} \to \mathbb{N} \to \mathbb{N}\\
&\mathit{myMin}\ 0\ n &&=\ 0\\
&\mathit{myMin}\ m\ 0 &&=\ 0\\
&\mathit{myMin}\ (\mathbf{1}{+}\ m)\ (\mathbf{1}{+}\ n) &&=\ \mathbf{1}{+}\ (\mathit{myMin}\ m\ n) \quad .
\end{aligned}
$$

3. Define a function $\mathit{elemAt} :: \mathbb{N} \to [a] \to a$ such that $\mathit{elemAt}\ n\ \mathit{xs}$ yields the nth element of xs.[2]

Solution:

$$
\begin{aligned}
&\mathit{elemAt} &&::\ \mathbb{N} \to [a] \to a\\
&\mathit{elemAt}\ 0\ (x : \mathit{xs}) &&=\ x\\
&\mathit{elemAt}\ (\mathbf{1}{+}\ n)\ (x : \mathit{xs}) &&=\ \mathit{elemAt}\ n\ \mathit{xs} \quad .
\end{aligned}
$$

4. Define a function $\mathit{insertAt} :: \mathbb{N} \to a \to [a] \to [a]$ such that $\mathit{insertAt}\ n\ x\ \mathit{xs}$ inserts x into xs such that the nth element of the new list is x.

Solution:

$$
\begin{aligned}
&\mathit{insertAt} &&::\ \mathbb{N} \to a \to [a] \to a\\
&\mathit{insertAt}\ 0\ x\ \mathit{xs} &&=\ x : \mathit{xs}\\
&\mathit{insertAt}\ (\mathbf{1}{+}\ n)\ x\ (y : \mathit{xs}) &&=\ y : \mathit{insertAt}\ n\ \mathit{xs} \quad .
\end{aligned}
$$

5 User-Defined Inductive Datatypes

1. Consider the type

$$\mathbf{data}\ \mathit{ETree}\ a\ =\ \mathsf{Tip}\ a\ \mid\ \mathsf{Bin}\ (\mathit{ETree}\ a)\ (\mathit{ETree}\ a) \quad .$$

   (a) How is it different from the type *Tree* in the lecture note?

   (b) Define Define $\mathit{minT} :: \mathit{ETree}\ \mathit{Int} \to \mathit{Int}$, which computes the minimal element in a tree. The operator for binary minimum in Haskell is $\mathit{min} :: \mathit{Ord}\ a \Rightarrow a \to a \to a$.

[2]This function is denoted (!!) in the standard library.

2. Define $minT :: Tree\ Int \to Int$, which computes the minimal element in a tree. The operator for binary minimum in Haskell is $min :: Ord\ a \to a \to a \to a$. And the largest Int in Haskell is denoted by $maxBound$.

3. Define $mapT :: (a \to b) \to Tree\ a \to Tree\ b$, which applies the functional argument to each element in a tree.

4. Define $flatten :: Tree\ a \to [a]$ that traverses a tree and collects all the labels, in-order, in a list. For example,

$$
\begin{array}{llll}
\mathit{flatten} & (\mathsf{Node}\ 4 & (\mathsf{Node}\ 2 & (\mathsf{Node}\ 1\ \mathsf{Null}\ \mathsf{Null}) \\
 & & & (\mathsf{Node}\ 3\ \mathsf{Null}\ \mathsf{Null})) \\
 & & (\mathsf{Node}\ 6 & (\mathsf{Node}\ 5\ \mathsf{Null}\ \mathsf{Null}) \\
 & & & (\mathsf{Node}\ 7\ \mathsf{Null}\ \mathsf{Null})))
\end{array}
$$

   yields $[1, 2, 3, 4, 5, 6, 7]$. **Hint**: use $(++)$.

5. A *binary search tree* is a tree of type $Tree\ a$, with $Ord\ a$, defined by:

   1. Null is a binary search tree, and
   2. Node $x\ t\ u$ is a binary search tree if:
      - every label in t is less than x,
      - every label in u is greater than x, and
      - t and u are also binary search trees.

   Define (assuming that t is a binary search tree):

   (a) $memberT :: Ord\ a \Rightarrow a \to Tree\ a \to Bool$, such that $memberT\ x\ t$ determines whether x occurs in t, and

   (b) $insertT :: Ord\ a \Rightarrow a \to Tree\ a \to Tree\ a$, such that $insertT\ x\ t$ inserts x into t and still returns a binary tree, if x does not appear in t, and returns t if x is in t.