Functional Programming
Practicals 0

Shin-Cheng Mu

FLOLAC 2024

Reviews...

1. A practice on curried functions.
   (a) Define a function *poly* such that $poly\ a\ b\ c\ x = a \times x^2 + b \times x + c$. All the inputs and the result are of type *Float*.
   (b) Reuse *poly* to define a function *poly1* such that $poly1\ x = x^2 + 2 \times x + 1$.
   (c) Reuse *poly* to define a function *poly2* such that $poly2\ a\ b\ c = a \times 2^2 + b \times 2 + c$.

Solution:

$$
\begin{array}{l}
poly :: \mathrm{Float} \rightarrow \mathrm{Float} \rightarrow \mathrm{Float} \rightarrow \mathrm{Float} \rightarrow \mathrm{Float} \\
poly\ a\ b\ c\ x = a \times x \times x + b \times x + c \\
poly1 :: \mathrm{Float} \rightarrow \mathrm{Float} \\
poly1 = poly\ 1\ 2\ 1 \\
poly :: \mathrm{Float} \rightarrow \mathrm{Float} \rightarrow \mathrm{Float} \rightarrow \mathrm{Float} \\
poly2\ a\ b\ c = poly\ a\ b\ c\ 2
\end{array}
$$

2. Type in the definition of *square* in your working file.
   (a) Define a function $quad :: \mathrm{Int} \rightarrow \mathrm{Int}$ such that $quad\ x$ computes x^4.

Solution:

$$
\begin{array}{l}
quad :: \mathrm{Int} \rightarrow \mathrm{Int} \\
quad\ x = square\ (square\ x)\ \ .
\end{array}
$$

(b) Type in this definition into your working file. Describe, in words, what this function does.

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

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

Solution:

$$\begin{aligned} &\mathit{quad} :: \mathsf{Int} \to \mathsf{Int} \\ &\mathit{quad} = \mathit{twice}\ \mathit{square} \ . \end{aligned}$$

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

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

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

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

4. Functions as arguments, and a quick practice on sectioning.

(a) Type in the following definition to your working file, without giving the type.

$$\mathit{forktimes}\ f\ g\ x = f\ x \times g\ x \ .$$

Use $: t$ in GHCi to find out the type of *forktimes*. You will end up getting a complex type which, for now, can be seen as equivalent to

$$(t \to \mathit{Int}) \to (t \to \mathit{Int}) \to t \to \mathit{Int} \ .$$

Can you explain this type?

(b) Define a function that, given input x, use *forktimes* to compute $x^2 + 3 \times x + 2$. **Hint**: $x^2 + 3 \times x + 2 = (x + 1) \times (x + 2)$.

Solution:

$$\begin{aligned} &\mathit{compute} :: \mathsf{Int} \to \mathsf{Int} \\ &\mathit{compute} = \mathit{forktimes}\ (+1)\ (+2) \ . \end{aligned}$$

(c) Type in the following definition into your working file: $\mathit{lift2}\ h\ f\ g\ x = h\ (f\ x)\ (g\ x)$. Find out the type of *lift2*. Can you explain its type?

Solution:

$$lift2 :: (a \to b \to c) \to (d \to a) \to (d \to b) \to d \to c \enspace .$$

(d) Use *lift2* to compute $x^2 + 3 \times x + 2$.

Solution:

$$\begin{aligned}&compute :: \mathsf{Int} \to \mathsf{Int}\\ &compute = lift2\ (\times)\ (+1)\ (+2) \enspace .\end{aligned}$$

Definitions and Proofs by Induction

1. Prove that *length* distributes into (++):

$$length\ (xs +\!\!+\ ys) = length\ xs + length\ ys \enspace .$$

Solution: Prove by induction on the structure of *xs*.

Case $xs := [\,]$:

$$\begin{aligned}&\quad length\ ([\,] +\!\!+\ ys)\\ =&\quad \{\ \text{definition of } (+\!\!+)\ \}\\ &\quad length\ ys\\ =&\quad \{\ \text{definition of } (+)\ \}\\ &\quad 0 + length\ ys\\ =&\quad \{\ \text{definition of } length\ \}\\ &\quad length\ [\,] + length\ ys\end{aligned}$$

Case $xs := x : xs$:

$$
\begin{aligned}
& length\ ((x : xs) +\!\!+\ ys) \\
= & \quad \{ \text{ definition of } (+\!\!+) \ \} \\
& length\ (x : (xs +\!\!+\ ys)) \\
= & \quad \{ \text{ definition of } length \ \} \\
& 1 + length\ (xs +\!\!+\ ys) \\
= & \quad \{ \text{ by induction } \} \\
& 1 + length\ xs + length\ ys \\
= & \quad \{ \text{ definition of } length \ \} \\
& length\ (x : xs) + length\ ys
\end{aligned}
$$

Note that we in fact omitted one step using the associativity of (+).

2. Prove: $sum \cdot concat = sum \cdot map\ sum$.

Solution: By extensional equality, $sum \cdot concat = sum \cdot map\ sum$ if and only if

$$(sum \cdot concat)\ xss = (sum \cdot map\ sum)\ xss,$$

for all xss, which, by definition of $(\cdot)$, is equivalent to

$$sum\ (concat\ xss) = sum\ (map\ sum\ xss),$$

which we will prove by induction on xss.

Case $xss := [\,]$:

$$
\begin{aligned}
& sum\ (concat\ [\,]) \\
= & \quad \{ \text{ definition of } concat \ \} \\
& sum\ [\,] \\
= & \quad \{ \text{ definition of } map \ \} \\
& sum\ (map\ sum\ [\,])
\end{aligned}
$$

Case $xss := xs : xss$:

$$
\begin{aligned}
& sum\ (concat\ (xs : xss)) \\
= & \quad \{ \text{ definition of } concat \ \} \\
& sum\ (xs \mathbin{+\!\!+} (concat\ xss)) \\
= & \quad \{ \text{ lemma: } sum \text{ distributes over } \mathbin{+\!\!+} \ \} \\
& sum\ xs + sum\ (concat\ xss) \\
= & \quad \{ \text{ by induction } \} \\
& sum\ xs + sum\ (map\ sum\ xss) \\
= & \quad \{ \text{ definition of } sum \ \} \\
& sum\ (sum\ xs : map\ sum\ xss) \\
= & \quad \{ \text{ definition of } map \ \} \\
& sum\ (map\ sum\ (xs : xss)).
\end{aligned}
$$

The lemma that *sum* distributes over $\mathbin{+\!\!+}$, that is,

$$sum\ (xs \mathbin{+\!\!+} ys) = sum\ xs + sum\ ys,$$

needs a separate proof by induction. Here it goes:

Case $xs := [\,]$:

$$
\begin{aligned}
& sum\ ([\,] \mathbin{+\!\!+} ys) \\
= & \quad \{ \text{ definition of } (\mathbin{+\!\!+}) \ \} \\
& sum\ ys \\
= & \quad \{ \text{ definition of } (+) \ \} \\
& 0 + sum\ ys \\
= & \quad \{ \text{ definition of } sum \ \} \\
& sum\ [\,] + sum\ ys.
\end{aligned}
$$

Case $xs := x : xs$:

$$
\begin{aligned}
& sum\,((x : xs) +\!\!+ ys) \\
= \quad & \{ \text{ definition of } (+\!\!+) \ \} \\
& sum\,(x : (xs +\!\!+ ys)) \\
= \quad & \{ \text{ definition of } sum \ \} \\
& x + sum\,(xs +\!\!+ ys) \\
= \quad & \{ \text{ induction } \} \\
& x + (sum\ xs + sum\ ys) \\
= \quad & \{ \text{ since } (+) \text{ is associative } \} \\
& (x + sum\ xs) + sum\ ys \\
= \quad & \{ \text{ definition of } sum \ \} \\
& sum\,(x : xs) + sum\ ys.
\end{aligned}
$$

3. Prove: $filter\ p \cdot map\ f = map\ f \cdot filter\ (p \cdot f)$.
Hint: for calculation, it might be easier to use this definition of *filter*:

$$
\begin{aligned}
& filter\ p\ [\,] && = [\,] \\
& filter\ p\ (x : xs) && = \textbf{if}\ p\ x\ \textbf{then}\ x : filter\ p\ xs \\
& && \quad\ \textbf{else}\ filter\ p\ xs
\end{aligned}
$$

and use the law that in the world of total functions we have:

$$f\ (\textbf{if}\ q\ \textbf{then}\ e_1\ \textbf{else}\ e_2) = \textbf{if}\ q\ \textbf{then}\ f\ e_1\ \textbf{else}\ f\ e_2$$

You may also carry out the proof using the definition of *filter* using guards:

$$
\begin{aligned}
& \ldots \\
& filter\ p\ (x : xs) \mid p\ x = \ldots \\
& \qquad\qquad\qquad\ \mid \textbf{otherwise} = \ldots
\end{aligned}
$$

You will then have to distinguish between the two cases: $p\ x$ and $\neg\,(p\ x)$, which makes the proof more fragmented. Both proofs are okay, however.

Solution:

$$
\begin{aligned}
& filter\ p \cdot map\ f = map\ f \cdot filter\ (p \cdot f) \\
\equiv \quad & \{ \text{ extensional equality } \} \\
& (\forall xs :: (filter\ p \cdot map\ f)\ xs = (map\ f \cdot filter\ (p \cdot f))\ xs) \\
\equiv \quad & \{ \text{ definition of } (\cdot) \ \} \\
& (\forall xs :: filter\ p\ (map\ f\ xs) = map\ f\ (filter\ (p \cdot f)\ xs)).
\end{aligned}
$$

We proceed by induction on *xs*.

Case $xs := [\,]$:

$$
\begin{array}{ll}
 & \mathit{filter}\ p\ (\mathit{map}\ f\ [\,]) \\
= & \{\ \text{definition of } \mathit{map}\ \} \\
 & \mathit{filter}\ p\ [\,] \\
= & \{\ \text{definition of } \mathit{filter}\ \} \\
 & [\,] \\
= & \{\ \text{definition of } \mathit{map}\ \} \\
 & \mathit{map}\ f\ [\,] \\
= & \{\ \text{definition of } \mathit{filter}\ \} \\
 & \mathit{map}\ f\ (\mathit{filter}\ (p \cdot f)\ [\,])
\end{array}
$$

Case $xs := x : xs$:

$$
\begin{array}{ll}
 & \mathit{filter}\ p\ (\mathit{map}\ f\ (x : xs)) \\
= & \{\ \text{definition of } \mathit{map}\ \} \\
 & \mathit{filter}\ p\ (f\ x : \mathit{map}\ f\ xs) \\
= & \{\ \text{definition of } \mathit{filter}\ \} \\
 & \textbf{if}\ p\ (f\ x)\ \textbf{then}\ f\ x : \mathit{filter}\ p\ (\mathit{map}\ f\ xs)\ \textbf{else}\ \mathit{filter}\ p\ (\mathit{map}\ f\ xs) \\
= & \{\ \text{induction hypothesis}\ \} \\
 & \textbf{if}\ p\ (f\ x)\ \textbf{then}\ f\ x : \mathit{map}\ f\ (\mathit{filter}(p \cdot f)\ xs)\ \textbf{else}\ \mathit{map}\ f\ (\mathit{filter}\ (p \cdot f)\ xs) \\
= & \{\ \text{defintion of } \mathit{map}\ \} \\
 & \textbf{if}\ p\ (f\ x)\ \textbf{then}\ \mathit{map}\ f\ (x : \mathit{filter}\ (p \cdot f)\ xs)\ \textbf{else}\ \mathit{map}\ f\ (\mathit{filter}\ (p \cdot f)\ xs) \\
= & \{\ \text{since } f\ (\textbf{if}\ q\ \textbf{then}\ e_1\ \textbf{else}\ e_2) = \textbf{if}\ q\ \textbf{then}\ f\ e_1\ \textbf{else}\ f\ e_2\ \} \\
 & \mathit{map}\ f\ (\textbf{if}\ p\ (f\ x)\ \textbf{then}\ x : \mathit{filter}\ (p \cdot f)\ xs\ \textbf{else}\ \mathit{filter}\ (p \cdot f)\ xs) \\
= & \{\ \text{definition of } (\cdot)\ \} \\
 & \mathit{map}\ f\ (\textbf{if}\ (p \cdot f)\ x\ \textbf{then}\ x : \mathit{filter}\ (p \cdot f)\ xs\ \textbf{else}\ \mathit{filter}\ (p \cdot f)\ xs) \\
= & \{\ \text{definition of } \mathit{filter}\ \} \\
 & \mathit{map}\ f\ (\mathit{filter}\ (p \cdot f)\ (x : xs))
\end{array}
$$

4. Reflecting on the law we used in the previous exercise:

$$f\ (\textbf{if}\ q\ \textbf{then}\ e_1\ \textbf{else}\ e_2) = \textbf{if}\ q\ \textbf{then}\ f\ e_1\ \textbf{else}\ f\ e_2$$

Can you think of a counterexample to the law above, when we allow the presence of $\bot$? What additional constraint shall we impose on f to make the law true?

Solution: Let $f = const\ 1$ (where $const\ x\ y = x$), and $q = \bot$. We have:

$$
\begin{array}{ll}
& const\ 1\ (\textbf{if}\ \bot\ \textbf{then}\ e_1\ \textbf{else}\ e_2) \\
= & \{\ \text{definition of } const\ \} \\
& 1 \\
\neq & \bot \\
= & \{\ \textbf{if} \text{ is strict on the conditional expression }\} \\
& \textbf{if}\ \bot\ \textbf{then}\ f\ e_1\ \textbf{else}\ f\ e_2
\end{array}
$$

The rule is restored if f is strict, that is, $f\ \bot = \bot$.

5. Prove: $take\ n\ xs +\!\!+\ drop\ n\ xs = xs$, for all n and xs.

Solution: By induction on n, then induction on xs.

Case $n := 0$

$$
\begin{array}{ll}
& take\ 0\ xs +\!\!+\ drop\ 0\ xs \\
= & \{\ \text{definitions of } take \text{ and } drop\ \} \\
& [\,] +\!\!+\ xs \\
= & \{\ \text{definition of } (+\!\!+)\ \} \\
& xs.
\end{array}
$$

Case $n := \mathbf{1}_+\ n$ and $xs := [\,]$

$$
\begin{array}{ll}
& take\ (\mathbf{1}_+\ n)\ [\,] +\!\!+\ drop\ (\mathbf{1}_+\ n)\ [\,] \\
= & \{\ \text{definitions of } take \text{ and } drop\ \} \\
& [\,] +\!\!+ [\,] \\
= & \{\ \text{definition of } (+\!\!+)\ \} \\
& [\,].
\end{array}
$$

Case $n := \mathbf{1}_+\ n$ and $xs := x : xs$

$$
\begin{array}{ll}
& take\ (\mathbf{1}_+\ n)\ (x : xs) +\!\!+\ drop\ (\mathbf{1}_+\ n)\ (x : xs) \\
= & \{\ \text{definitions of } take \text{ and } drop\ \} \\
& (x : take\ n\ xs) +\!\!+\ drop\ n\ xs \\
= & \{\ \text{definition of } (+\!\!+)\ \} \\
& x : take\ n\ xs +\!\!+\ drop\ n\ xs \\
= & \{\ \text{induction}\ \} \\
& x : xs.
\end{array}
$$

6. Define a function $fan :: a \to List\ a \to List\ (List\ a)$ 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:

$$\begin{array}{ll} fan & :: a \to List\ a \to List\ (List\ a) \\ fan\ x\ [\,] & = [[x]] \\ fan\ x\ (y : ys) & = (x : y : ys) : map\ (y\ :)\ (fan\ x ys) \end{array}$$

7. Prove: $map\ (map\ f) \cdot fan\ x = fan\ (f\ x) \cdot map\ f$, for all f and x. **Hint**: you will need the *map*-fusion law, and to spot that $map\ f \cdot (y\ :) = (f\ y\ :) \cdot map\ f$ (why?).

Solution: This is equivalent to proving that, for all f, x, and xs:

$$map\ (map\ f)\ (fan\ x\ xs) = fan\ (f\ x)\ (map\ f\ xs)\ .$$

Induction on xs.
Case $xs := [\,]$:

$$\begin{array}{ll} & map\ (map\ f)\ (fan\ x\ [\,]) \\ = & \{\ \text{definition of } fan\ \} \\ & map\ (map\ f)\ [[x]] \\ = & \{\ \text{definition of } map\ \} \\ & [[f\ x]] \\ = & \{\ \text{definition of } fan\ \} \\ & fan (f\ x)\ [\,] \\ = & \{\ \text{definition of } fan\ \} \\ & fan\ (f\ x)\ (map\ f\ [\,])\ . \end{array}$$

Case $xs := y : ys$:

$$\begin{array}{ll} & map\ (map\ f)\ (fan\ x\ (y : ys)) \\ = & \{\ \text{definition of } fan\ \} \\ & map\ (map\ f)\ ((x : y : ys) : map\ (y\ :)\ (fan\ x\ ys)) \\ = & \{\ \text{definition of } map\ \} \\ & map\ f\ (x : y : ys) : map\ (map\ f)\ (map\ (y\ :)\ (fan\ x\ ys)) \\ = & \{\ map\text{-fusion}\ \} \\ & map\ f\ (x : y : ys) : map\ (map\ f \cdot (y\ :))\ (fan\ x\ ys) \\ = & \{\ \text{definition of } map\ \} \\ & map\ f\ (x : y : ys) : map\ ((fy\ :) \cdot map\ f)\ (fan\ x\ ys) \\ = & \{\ map\text{-fusion}\ \} \\ & map\ f\ (x : y : ys) : map\ (fy\ :)\ (map\ (map\ f)\ (fan\ x\ ys)) \\ = & \{\ \text{induction}\ \} \\ & map\ f\ (x : y : ys) : map\ (fy\ :)\ (fan\ (f\ x)\ (map\ f\ ys)) \\ = & \{\ \text{definition of } map\ \} \\ & (f\ x : f\ y : map\ f\ ys) : map\ (fy\ :)\ (fan\ (f\ x)\ (map\ f\ ys)) \\ = & \{\ \text{definition of } fan\ \} \\ & fan\ (f\ x)\ (f\ y : map\ f\ ys) \end{array}$$

8. Define $\mathit{perms} :: \mathit{List}\ a \rightarrow \mathit{List}\ (\mathit{List}\ a)$ that returns all permutations of the input list. For example:

$$\mathit{perms}\ [1,2,3] = [[1,2,3],[2,1,3],[2,3,1],[1,3,2],[3,1,2],[3,2,1]]\ .$$

You will need several auxiliary functions defined in the lectures and in the exercises.

Solution:

$$\begin{array}{ll} \mathit{perms} & :: \mathit{List}\ a \rightarrow \mathit{List}\ (\mathit{List}\ a) \\ \mathit{perms}\ [\,] & = [[\,]] \\ \mathit{perms}\ (x : xs) & = \mathit{concat}\ (\mathit{map}\ (\mathit{fan}\ x)\ (\mathit{perms}\ xs)) \end{array}$$

9. Prove: $\mathit{map}\ (\mathit{map}\ f) \cdot \mathit{perm} = \mathit{perm} \cdot \mathit{map}\ f$. You may need previously proved results, as well as a property about *concat* and *map*: for all g, we have $\mathit{map}\ g \cdot \mathit{concat} = \mathit{concat} \cdot \mathit{map}\ (\mathit{map}\ g)$.

Solution: This is equivalent to proving that, for all f and xs:

$$\mathit{map}\ (\mathit{map}\ f)\ (\mathit{perm}\ xs) = \mathit{perm}\ (\mathit{map}\ f\ xs)\ .$$

Induction on xs.

Case $xs := [\,]$:

$$\begin{array}{ll} & \mathit{map}\ (\mathit{map}\ f)\ (\mathit{perm}\ [\,]) \\ = & \{\ \text{definition of } \mathit{perm}\ \} \\ & \mathit{map}\ (\mathit{map}\ f)\ [[\,]] \\ = & \{\ \text{definition of } \mathit{map}\ \} \\ & [[\,]] \\ = & \{\ \text{definition of } \mathit{perm}\ \} \\ & \mathit{perm}\ [\,] \\ = & \{\ \text{definition of } \mathit{map}\ \} \\ & \mathit{perm}\ (\mathit{map}\ f\ [\,])\ . \end{array}$$

Case $xs := x : xs$:

$$\begin{array}{ll} & \mathit{map}\ (\mathit{map}\ f)\ (\mathit{perm}\ (x : xs)) \\ = & \{\ \text{definition of } \mathit{perm}\ \} \\ & \mathit{map}\ (\mathit{map}\ f)\ (\mathit{concat}\ (\mathit{map}\ (\mathit{fan}\ x)\ (\mathit{perm}\ xs))) \\ = & \{\ \text{since } \mathit{map}\ g \cdot \mathit{concat} = \mathit{concat} \cdot \mathit{map}\ (\mathit{map}\ g)\ \} \\ & \mathit{concat}\ (\mathit{map}\ (\mathit{map}\ (\mathit{map}\ f))(\mathit{map}\ (\mathit{fan}\ x)\ (\mathit{perm}\ xs))) \\ = & \{\ \mathit{map}\text{-fusion}\ \} \\ & \mathit{concat}\ (\mathit{map}\ (\mathit{map}\ (\mathit{map}\ f) \cdot \mathit{fan}\ x)\ (\mathit{perm}\ xs)) \\ = & \{\ \text{previous exercise}\ \} \\ & \mathit{concat}\ (\mathit{map}\ (\mathit{fan}\ (f\ x) \cdot \mathit{map}\ f)\ (\mathit{perm}\ xs)) \\ = & \{\ \mathit{map}\text{-fusion}\ \} \\ & \mathit{concat}\ (\mathit{map}\ (\mathit{fan}\ (f\ x))\ (\mathit{map}\ (\mathit{map}\ f)\ (\mathit{perm}\ xs))) \\ = & \{\ \text{induction}\ \} \\ & \mathit{concat}\ (\mathit{map}\ (\mathit{fan}\ (f\ x))\ (\mathit{perm}\ (\mathit{map}\ f\ xs))) \\ = & \{\ \text{definition of } \mathit{perm}\ \} \\ & \mathit{perm}\ (f\ x : \mathit{map}\ f\ xs) \\ = & \{\ \text{definition of } \mathit{map}\ \} \end{array}$$

10. Define *inits* :: *List a* → *List* (*List a*) that returns all prefixes of the input list.

    inits "abcde" = ["", "a", "ab", "abc", "abcd", "abcde"].

    Hint: the empty list has *one* prefix: the empty list. The solution has been given in the lecture. Please try it again yourself.

    Solution:

    ```
    inits          :: List a → List (List a)
    inits [ ]      = [[ ]]
    inits (x : xs) = [ ] : map (x :) (inits xs)  .
    ```

11. Define *tails* :: *List a* → *List* (*List a*) that returns all suffixes of the input list.

    tails "abcde" = ["abcde", "bcde", "cde", "de", "e", ""].

    Hint: the empty list has *one* suffix: the empty list. The solution has been given in the lecture. Please try it again yourself.

    Solution:

    ```
    tails          :: List a → List (List a)
    tails [ ]      = [[ ]]
    tails (x : xs) = (x : xs) : tails xs  .
    ```

12. The function *splits* :: *List a* → *List* (*List a*, *List a*) returns all the ways a list can be split into two. For example,

    splits [1, 2, 3, 4] = [([ ], [1, 2, 3, 4]), ([1], [2, 3, 4]), ([1, 2], [3, 4]),
    ([1, 2, 3], [4]), ([1, 2, 3, 4], [ ])] .

    Define *splits* inductively on the input list. **Hint**: you may find it useful to define, in a **where**-clause, an auxiliary function *f* (*ys*, *zs*) = ... that matches pairs. Or you may simply use (λ (*ys*, *zs*) → ...).

    Solution:

    ```
    splits          :: List a → List (List a, List a)
    splits [ ]      = [([ ], [ ])]
    splits (x : xs) = ([ ], x : xs) : map cons1 (splits xs)  ,
          where cons1 (ys, zs) = (x : ys, zs)  .
    ```

If you know how to use λ expressions, you may:

$$
\begin{array}{lcl}
splits & :: & List\ a \rightarrow List\ (List\ a, List\ a) \\
splits\ [\,] & = & [([\,],[\,])] \\
splits\ (x : xs) & = & ([\,], x : xs) : map\ (\lambda\ (ys, zs) \rightarrow (x : ys, zs))\ (splits\ xs)\ \ .
\end{array}
$$

13. An *interleaving* of two lists *xs* and *ys* is a permutation of the elements of both lists such that the members of *xs* appear in their original order, and so does the members of *ys*. Define $interleave :: List\ a \rightarrow List\ a \rightarrow List\ (List\ a)$ such that *interleave xs ys* is the list of interleaving of *xs* and *ys*. For example, *interleave* [1, 2, 3] [4, 5] yields:

    [[1, 2, 3, 4, 5], [1, 2, 4, 3, 5], [1, 2, 4, 5, 3], [1, 4, 2, 3, 5], [1, 4, 2, 5, 3],
    [1, 4, 5, 2, 3], [4, 1, 2, 3, 5], [4, 1, 2, 5, 3], [4, 1, 5, 2, 3], [4, 5, 1, 2, 3]].

Solution:

$$
\begin{array}{lcl}
interleave & :: & List\ a \rightarrow List\ a \rightarrow List\ (List\ a) \\
interleave\ [\,]\ ys & = & [ys] \\
interleave\ xs\ [\,] & = & [xs] \\
interleave\ (x : xs)\ (y : ys) & = & map\ (x\ :)\ (interleave\ xs\ (y : ys)) \mathbin{+\!\!+} \\
 & & \quad map\ (y\ :)\ (interleave\ (x : xs)\ ys)\ \ .
\end{array}
$$

14. A list *ys* is a *sublist* of *xs* if we can obtain *ys* by removing zero or more elements from *xs*. For example, [2, 4] is a sublist of [1, 2, 3, 4], while [3, 2] is *not*. The list of all sublists of [1, 2, 3] is:

    [[], [3], [2], [2, 3], [1], [1, 3], [1, 2], [1, 2, 3]].

    Define a function $sublist ::\ List\ a \rightarrow List\ (List\ a)$ that computes the list of all sublists of the given list. **Hint**: to form a sublist of *xs*, each element of *xs* could either be kept or dropped.

Solution:

$$
\begin{array}{lcl}
sublist & :: & List\ a \rightarrow List\ (List\ a) \\
sublist\ [\,] & = & [[\,]] \\
sublist\ (x : xs) & = & xss \mathbin{+\!\!+} map\ (x\ :)\ xss\ \ , \\
 & & \mathbf{where}\ xss = sublist\ xs\ \ .
\end{array}
$$

The righthand side could be $sublist\ \ xs \mathbin{+\!\!+} map\ \ (x\ \ :)\ \ (sublist\ \ xs)$ (but it could be much slower).

15. Consider the following datatype for internally labelled binary trees:

$$\textbf{data}\ \mathit{Tree}\ a\ =\ \mathrm{Null} \mid \mathrm{Node}\ a\ (\mathit{Tree}\ a)\ (\mathit{Tree}\ a)\ .$$

(a) Given $(\downarrow) :: \mathit{Nat} \to \mathit{Nat} \to \mathit{Nat}$, which yields the smaller one of its arguments, define $\mathit{minT} :: \mathit{Tree}\ \mathit{Nat} \to \mathit{Nat}$, which computes the minimal element in a tree. (Note: $(\downarrow)$ is actually called *min* in the standard library. In the lecture we use the symbol $(\downarrow)$ to be brief.)

Solution:

$$\begin{array}{lcl} \mathit{minT} & :: & \mathit{Tree}\ \mathit{Nat} \to \mathit{Nat} \\ \mathit{minT}\ \mathrm{Null} & = & \mathit{maxBound} \\ \mathit{minT}\ (\mathrm{Node}\ x\ t\ u) & = & x \downarrow \mathit{minT}\ t \downarrow \mathit{minT}\ u\ . \end{array}$$

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

Solution:

$$\begin{array}{lcl} \mathit{mapT} & :: & (a \to b) \to \mathit{Tree}\ a \to \mathit{Tree}\ b \\ \mathit{mapT}\ f\ \mathrm{Null} & = & \mathrm{Null} \\ \mathit{mapT}\ f\ (\mathrm{Node}\ x\ t\ u) & = & \mathrm{Node}\ (f\ x)\ (\mathit{mapT}\ f\ t)\ (\mathit{mapT}\ f\ u)\ . \end{array}$$

(c) Can you define $(\downarrow)$ inductively on *Nat*?

Solution:

$$\begin{array}{lcl} (\downarrow) & :: & \mathit{Nat} \to \mathit{Nat} \to \mathit{Nat} \\ 0 \downarrow n & = & 0 \\ (\mathbf{1}_{+} m) \downarrow 0 & = & 0 \\ (\mathbf{1}_{+} m) \downarrow (\mathbf{1}_{+} n) & = & \mathbf{1}_{+}\ (m \downarrow n)\ . \end{array}$$

(d) Prove that for all n and t, $\mathit{minT}\ (\mathit{mapT}\ (n+)\ t) = n + \mathit{minT}\ t$. That is, $\mathit{minT} \cdot \mathit{mapT}\ (n+) = (n+) \cdot \mathit{minT}$.

Solution: Induction on t.

Case $t := \mathrm{Null}$. Omitted.

Case $t := \text{Node}\ x\ t\ u$.

$$
\begin{aligned}
& \quad \mathit{minT}\ (\mathit{mapT}\ (n+)\ (\text{Node}\ x\ t\ u)) \\
= & \quad \{\ \text{definition of } \mathit{mapT}\ \} \\
& \quad \mathit{minT}\ (\text{Node}\ (n+x)\ (\mathit{mapT}\ (n+)\ t)\ (\mathit{mapT}\ (n+)\ u)) \\
= & \quad \{\ \text{definition of } \mathit{minT}\ \} \\
& \quad (n+x) \downarrow \mathit{minT}\ (\mathit{mapT}\ (n+)\ t)) \downarrow \mathit{minT}\ (\mathit{mapT}\ (n+)\ u) \\
= & \quad \{\ \text{by induction}\ \} \\
& \quad (n+x) \downarrow (n + \mathit{minT}\ t) \downarrow (n + \mathit{minT}\ u) \\
= & \quad \{\ \text{lemma: } (n+x) \downarrow (n+y) = n + (x \downarrow y)\ \} \\
& \quad n + (x \downarrow \mathit{minT}\ t \downarrow \mathit{minT}\ u) \\
= & \quad \{\ \text{definition of } \mathit{minT}\ \} \\
& \quad n + \mathit{minT}\ (\text{Node}\ x\ t\ u)\ \ .
\end{aligned}
$$

The lemma $(n + x) \downarrow (n + y) = n + (x \downarrow y)$ can be proved by induction on n, using inductive definitions of $(+)$ and $(\downarrow)$.