Functional Programming
Practicals 0

Shin-Cheng Mu

FLOLAC 2022

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:

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

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

Solution:

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

(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:

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

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

Solution:

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

1 Definitions and Proofs by Induction

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

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

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 $xs := x : xs$:

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

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

2. Prove: $\mathit{sum} \cdot \mathit{concat} = \mathit{sum} \cdot \mathit{map}\ \mathit{sum}$.

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

$$(\mathit{sum} \cdot \mathit{concat})\ xss = (\mathit{sum} \cdot \mathit{map}\ \mathit{sum})\ xss,$$

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

$$\mathit{sum}\,(\mathit{concat}\ xss) = \mathit{sum}\,(\mathit{map}\ \mathit{sum}\ xss),$$

which we will prove by induction on xss.

Case $xss := [\,]$:

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

Case $xss := xs : xss$:

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

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 $xs := [\,]$:

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

Case $xs := x : xs$:

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

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

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

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:

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

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:

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

We proceed by induction on *xs*.

Case $xs := [\,]$:

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

Case $xs := x : xs$:

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

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

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

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

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

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

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

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

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

Case $n := 0$

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

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

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

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

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

6. Define a function $\mathit{fan} :: a \to \mathit{List}\ a \to \mathit{List}\ (\mathit{List}\ a)$ such that $\mathit{fan}\ x\ \mathit{xs}$ inserts x into the 0th, 1st…nth positions of xs, where n is the length of xs. For example:

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

Solution:

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

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

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

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

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

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

Case $\mathit{xs} := y : \mathit{ys}$:

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

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

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

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

Solution:

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

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

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

$$map\ (map\ f)\ (perm\ xs) = perm\ (map\ f\ xs)\ .$$

Induction on xs.

Case $xs := [\,]$:

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

Case $xs := x : xs$:

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

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:

    $$\begin{array}{lcl} \mathit{inits} & :: & \mathit{List}\ a \rightarrow \mathit{List}\ (\mathit{List}\ a) \\ \mathit{inits}\ [\,] & = & [[\,]] \\ \mathit{inits}\ (x : \mathit{xs}) & = & [\,] : \mathit{map}\ (x\,:)\ (\mathit{inits}\ \mathit{xs}) \quad . \end{array}$$

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:

    $$\begin{array}{lcl} \mathit{tails} & :: & \mathit{List}\ a \rightarrow \mathit{List}\ (\mathit{List}\ a) \\ \mathit{tails}\ [\,] & = & [[\,]] \\ \mathit{tails}\ (x : \mathit{xs}) & = & (x : \mathit{xs}) : \mathit{tails}\ \mathit{xs} \quad . \end{array}$$

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

    $$\begin{array}{lcl} \mathit{splits}\ [1,2,3,4] & = & [([\,],[1,2,3,4]),([1],[2,3,4]),([1,2],[3,4]), \\ & & \ ([1,2,3],[4]),([1,2,3,4],[\,])] \quad . \end{array}$$

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

    Solution:

    $$\begin{array}{lcl} \mathit{splits} & :: & \mathit{List}\ a \rightarrow \mathit{List}\ (\mathit{List}\ a, \mathit{List}\ a) \\ \mathit{splits}\ [\,] & = & [([\,],[\,])] \\ \mathit{splits}\ (x : \mathit{xs}) & = & ([\,], x : \mathit{xs}) : \mathit{map}\ \mathit{cons1}\ (\mathit{splits}\ \mathit{xs}) \quad , \\ \quad \mathbf{where}\ \mathit{cons1}\ (\mathit{ys}, \mathit{zs}) = (x : \mathit{ys}, \mathit{zs}) \quad . & & \end{array}$$

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

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

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 $\mathit{interleave} :: \mathit{List}\ a \rightarrow \mathit{List}\ a \rightarrow \mathit{List}\ (\mathit{List}\ a)$ such that *interleave xs ys* is the list of interleaving of *xs* and *ys*. For example, *interleave* $[1, 2, 3]$ $[4, 5]$ yields:

$$\begin{array}{l} [[1,2,3,4,5],[1,2,4,3,5],[1,2,4,5,3],[1,4,2,3,5],[1,4,2,5,3], \\ \ [1,4,5,2,3],[4,1,2,3,5],[4,1,2,5,3],[4,1,5,2,3],[4,5,1,2,3]]. \end{array}$$

Solution:

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

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 $\mathit{sublist} :: \mathit{List}\ a \rightarrow \mathit{List}\ (\mathit{List}\ a)$ 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:

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

The righthand side could be $\mathit{sublist}\ \mathit{xs} \mathbin{+\!\!+} \mathit{map}\ (x\ \ :)\ (\mathit{sublist}\ \mathit{xs})$ (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 $\mathit{minT} :: \mathit{Tree\ Nat} \to \mathit{Nat}$, 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:**
>
> $$\begin{array}{lcl} \mathit{minT} & :: & \mathit{Tree\ Nat} \to \mathit{Nat} \\ \mathit{minT}\ \mathrm{Null} & = & \mathit{maxBound} \\ \mathit{minT}\ (\mathrm{Node}\ x\ t\ u) & = & x \downarrow \mathit{minT}\ t \downarrow \mathit{minT}\ u\ \ . \end{array}$$

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

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

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