# Functional Programming
# Practicals 0

Shin-Cheng Mu

FLOLAC 2026

## 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:**

$poly$ :: Float → Float → Float → Float → Float
$poly\ a\ b\ c\ x = a \times x \times x + b \times x + c$

$poly1$ :: Float → Float
$poly1 = poly\ 1\ 2\ 1$

$poly$ :: Float → Float → Float → Float
$poly2\ a\ b\ c = poly\ a\ b\ c\ 2$

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

**Solution:**

$quad$ :: Int → Int
$quad\ x = square\ (square\ x)$ .

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

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

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

**Solution:**

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

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

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

(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.

$$forktimes\ f\ g\ x = f\ x \times g\ x\ .$$

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

$$(t \to Int) \to (t \to Int) \to t \to Int\ .$$

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:**

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

(c) Type in the following definition into your working file: $lift2\ h\ f\ g\ x = h\ (f\ x)\ (g\ x)$. 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:**

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

# 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** $xs := [\,]$:
**Goal**: $length\ ([\,] +\!\!+ ys) = length\ [\,] + length\ ys$.
**Proof**:

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

**Case** $xs := x : xs$:
**Assume**: $length\ (xs +\!\!+ ys) = length\ xs + length\ ys$
**Goal**: $length\ ((x : xs) +\!\!+ ys) = length\ (x : xs) + length\ ys$.

**Proof**:

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

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

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

**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** $xss := [\,]$:
**Goal**: $sum\ (concat\ [\,]) = sum\ (map\ sum\ [\,])$
**Proof**:

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

**Case** $xss := xs : xss$:
**Assume**: $sum\ (concat\ xss) = sum\ (map\ sum\ xss)$

**Goal**: $sum\ (concat\ (xs : xss)) = sum\ (map\ sum\ (xs : xss))$
**Proof**:

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

The lemma that $sum$ distributes over $+\!\!+$, that is,

$$sum\ (xs +\!\!+ ys) = sum\ xs + sum\ ys,$$

needs a separate proof by induction. Here it goes:

**Case** $xs := [\,]$:
**Goal**: $sum\ ([\,] +\!\!+ ys) = sum\ [\,] + sum\ ys$.
**Proof**:

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

**Case** $xs := x : xs$:
**Assume**: $sum\ (xs +\!\!+ ys) = sum\ xs + sum\ ys$.
**Goal**: $sum\ ((x : xs) +\!\!+ ys) = sum\ (x : xs) + sum\ ys$.

**Proof**:

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

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*:

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

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

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

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

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

   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** $xs := [\,]$.
**Goal**: $\mathit{filter}\ p\ (\mathit{map}\ f\ [\,]) = \mathit{map}\ f\ (\mathit{filter}\ (p \cdot f)\ [\,])$.
**Proof**:

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

**Case** $xs := x : xs$.
**Assume**: $\mathit{filter}\ p\ (\mathit{map}\ f\ xs) = \mathit{map}\ f\ (\mathit{filter}(p \cdot f)\ xs)$.
**Goal**: $\mathit{filter}\ p\ (\mathit{map}\ f\ (x : xs)) = \mathit{map}\ f\ (\mathit{filter}\ (p \cdot f)\ (x : xs))$.
**Proof**:

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

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

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

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:

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

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** $n := 0$.
**Goal**: $take\ 0\ xs +\!\!+ drop\ 0\ xs = xs$.
**Proof**:

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

**Case** $n := 1_+\ n$ and $xs := [\,]$.
**Goal**: $take\ (1_+\ n)\ [\,] +\!\!+ drop\ (1_+\ n)\ [\,] = [\,]$.
**Proof**:

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

**Case** $n := 1_+\ n$ and $xs := x : xs$.
**Assume**: $take\ n\ xs +\!\!+ drop\ n\ xs = xs$.
**Goal**: $take\ (1_+\ n)\ (x : xs) +\!\!+ drop\ (1_+\ n)\ (x : xs) = x : xs$.
**Proof**:

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

6. Define a function $fan :: a \to List\ a \to List\ (List\ a)$ such that $fan\ x\ xs$ inserts $x$ into the 0th, 1st…$n$th 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:**

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

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** $xs := [\,]$.
**Goal**: $map\ (map\ f)\ (fan\ x\ [\,]) = fan\ (f\ x)\ (map\ f\ [\,])$.

**Proof**:

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

**Case** $xs := y : ys$.
**Assume**: $map\ (map\ f)\ (fan\ x\ ys) = fan\ (f\ x)\ (map\ f\ ys)$.
**Goal**: $map\ (map\ f)\ (fan\ x\ (y : ys)) = fan\ (f\ x)\ (map\ f\ (y : ys))$.
**Proof**:

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

8. Define $perms :: List\ a \to List\ (List\ a)$ 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:**

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

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** $xs := [\,]$.
**Goal**: $map\ (map\ f)\ (perm\ [\,]) = perm\ (map\ f\ [\,])$.
**Proof**:

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

**Case** $xs := x : xs$:
**Assume**: $map\ (map\ f)\ (perm\ xs) = perm\ (map\ f\ xs)$.
**Goal**: $map\ (map\ f)\ (perm\ (x : xs)) = perm\ (map\ f\ (x : xs))$.
**Proof**:

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

10. Define $inits :: List\ a \to List\ (List\ a)$ 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:**

    $$\begin{aligned} inits\ &:: List\ a \to List\ (List\ a) \\ inits\ [\,]\ &= [[\,]] \\ inits\ (x : xs)\ &= [\,] : map\ (x :)\ (inits\ xs)\ . \end{aligned}$$

11. Define $tails :: List\ a \to List\ (List\ a)$ 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:**

    $$\begin{aligned} tails\ &:: List\ a \to List\ (List\ a) \\ tails\ [\,]\ &= [[\,]] \\ tails\ (x : xs)\ &= (x : xs) : tails\ xs\ . \end{aligned}$$

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

    $$\begin{aligned} splits\ [1,2,3,4]\ = [&([\,], [1,2,3,4]), ([1], [2,3,4]), ([1,2], [3,4]), \\ &([1,2,3], [4]), ([1,2,3,4], [\,])]\ . \end{aligned}$$

    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 $(\lambda\ (ys, zs) \to \ldots)$.

    **Solution:**

    $$\begin{aligned} splits\ &:: List\ a \to List\ (List\ a, List\ a) \\ splits\ [\,]\ &= [([\,], [\,])] \\ splits\ (x : xs)\ &= ([\,], x : xs) : map\ cons1\ (splits\ xs)\ , \\ &\quad \text{where } cons1\ (ys, zs) = (x : ys, zs)\ . \end{aligned}$$

If you know how to use $\lambda$ expressions, you may:

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

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 $interleave :: List\ a \to List\ a \to List\ (List\ a)$ 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:**

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

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 $sublist :: List\ a \to List\ (List\ a)$ 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:**

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

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

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

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

(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 $\mathit{min}$ in the standard library. In the lecture we use the symbol $(\downarrow)$ to be brief.)

**Solution:**

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

(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:**

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

(c) Can you define $(\downarrow)$ inductively on $\mathit{Nat}$?

**Solution:**

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

(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** $t :=$ Null.

**Goal**: $\mathit{minT}\ (\mathit{mapT}\ (n+)\ \text{Null}) = n + \mathit{minT}\ \text{Null}$.

**Proof**: Omitted.

**Case** $t$ := Node $t\ x\ u$.
**Assume**: $minT\ (mapT\ (n+)\ t) = n + minT\ t\ \wedge\ minT\ (mapT\ (n+)\ u) = n + minT\ u$.
**Goal**: $minT\ (mapT\ (n+)\ (\text{Node}\ t\ x\ u)) = n + minT\ (\text{Node}\ t\ x\ u)$.
**Proof**:

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

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

16. Consider the Tree defined in question 15. A binary search tree is a Tree Int satisfying the following predicate:

$$
\begin{aligned}
& isBST :: \text{Tree Int} \to \text{Bool} \\
& isBST\ \text{Null} = \text{True} \\
& isBST\ (\text{Node}\ t\ x\ u) = allT\ (<x)\ t\ \wedge\ allT\ (x \leqslant)\ u\ \wedge \\
& \qquad\qquad isBST\ t\ \wedge\ isBST\ u\ ,
\end{aligned}
$$

where $allT :: (a \to \text{Bool}) \to \text{Tree}\ a \to \text{Bool}$ is defined by:

$$
\begin{aligned}
& allT\ p\ \text{Null} = \text{True} \\
& allT\ p\ (\text{Node}\ t\ x\ u) = allT\ p\ t\ \wedge\ p\ x\ \wedge\ allT\ p\ u\ .
\end{aligned}
$$

The function $insert$ is define by:

$$
\begin{aligned}
& insert :: \text{Int} \to \text{Tree Int} \to \text{Tree Int} \\
& insert\ x\ \text{Null} = \text{Node}\ x\ \text{Null Null} \\
& insert\ x\ (\text{Node}\ t\ y\ u) = \text{if}\ x < y\ \text{then Node}\ (insert\ x\ t)\ y\ u \\
& \qquad\qquad\qquad\qquad\qquad\qquad \text{else Node}\ t\ y\ (insert\ x\ u)\ .
\end{aligned}
$$

Prove that $isBST\ t \Rightarrow isBST\ (insert\ x\ t)$, for all $x$ and $t$.

You may assume the following rule for proving propertie involving if:

$$\text{if } p \text{ then } q \text{ else } r \;\equiv\; (p \Rightarrow q) \wedge (\mathit{not}\ p \Rightarrow r) \;\; .$$

You will need a lemma about $allT$. Try finding out what it is!

**Solution:** Induction on $t$.

**Case** $t :=$ Null.

**Goal**: $isBST$ Null $\Rightarrow$ $isBST$ ($insert\ x$ Null).

**Proof**: the goal simplifies to True $\Rightarrow$ $isBST$ ($insert\ x$ Null). We try to prove $isBST$ ($insert\ x$ Null).

$isBST$ ($insert\ x$ Null)
= { definition of $insert$ }
$isBST$ (Node Null $x$ Null)
= { definition of $isBST$ }
$allT$ ($<x$) Null $\wedge$ $allT$ ($x \leqslant$) Null $\wedge$ $isBST$ Null $\wedge$ $isBST$ Null
= { definitions of $allT$ and $isBST$ }
True $\wedge$ True $\wedge$ True $\wedge$ True
= { definition of ($\wedge$) }
True .

**Case** $t :=$ Node $t\ y\ u$.

**Assume** (1): $isBST\ t \Rightarrow isBST\ (insert\ x\ t)$ and $isBST\ u \Rightarrow isBST\ x\ u$.

**Goal**: $isBST$ (Node $t\ y\ u$) $\Rightarrow$ $isBST$ ($insert\ x$ (Node $t\ y\ u$)).

**Proof**:

**Assume** (2): $isBST$ (Node $t\ y\ u$).
That is, $allT$ ($<y$) $t \wedge allT$ ($y \leqslant$) $u \wedge isBST\ t \wedge isBST\ u$.

**Goal**: $isBST$ ($insert\ x$ (Node $t\ y\ u$)).

**Proof**:

$isBST$ ($insert\ x$ (Node $t\ y\ u$))
= { definition of $insert$ }
$isBST$ (if $x < y$ then Node ($insert\ x\ t$) $y\ u$
else Node $t\ y$ ($insert\ x\ u$))
= { application distributes into if }
if $x < y$ then $isBST$ (Node ($insert\ x\ t$) $y\ u$)
else $isBST$ (Node $t\ y$ ($insert\ x\ u$))

$=$ { definition of $isBST$ }

$\quad$ if $x < y$ then $allT\ (<y)\ (insert\ x\ t) \wedge allT\ (y \leqslant)\ u \wedge$
$\quad\quad isBST\ (insert\ x\ t) \wedge isBST\ u$
$\quad\quad$ else $allT\ (<y)\ t \wedge allT\ (y \leqslant)\ (insert\ x\ u) \wedge$
$\quad\quad isBST\ t \wedge isBST\ (insert\ x\ u)$

$=$ { conditionals }

$\quad (x < y \Rightarrow allT\ (<y)\ (insert\ x\ t) \wedge allT\ (y \leqslant)\ u \wedge$
$\quad\quad isBST\ (insert\ x\ t) \wedge isBST\ u) \wedge$
$\quad (y \leqslant x \Rightarrow allT\ (<y)\ t \wedge allT\ (y \leqslant)\ (insert\ x\ u) \wedge$
$\quad\quad isBST\ t \wedge isBST\ (insert\ x\ u))$

$\Leftarrow$ { induction (assumptions (1)) }

$\quad (x < y \Rightarrow allT\ (<y)\ (insert\ x\ t) \wedge allT\ (y \leqslant)\ u \wedge$
$\quad\quad isBST\ t \wedge isBST\ u) \wedge$
$\quad (y \leqslant x \Rightarrow allT\ (<y)\ t \wedge allT\ (y \leqslant)\ (insert\ x\ u) \wedge$
$\quad\quad isBST\ t \wedge isBST\ u)$

$\Leftarrow$ { predicate logic }

$\quad (x < y \Rightarrow allT\ (<y)\ (insert\ x\ t)) \wedge allT\ (y \leqslant)\ u \wedge$
$\quad (y \leqslant x \Rightarrow allT\ (y \leqslant)\ (insert\ x\ u)) \wedge allT\ (<y)\ t \wedge$
$\quad isBST\ t \wedge isBST\ u$

$\Leftarrow$ { lemma: $allT\ p\ t \wedge p\ x \Rightarrow allT\ p\ (insert\ x\ t)$ for all $p$, $t$, and $x$ }

$\quad allT\ (<y)\ t \wedge allT\ (y \leqslant)\ u \wedge isBST\ t \wedge isBST\ u$

$=$ { Assumptions (2) }

$\quad$ True .

The property we need about $allT$ is that for all $p$, $t$, and $x$,

$$allT\ p\ t \wedge p\ x \Rightarrow allT\ p\ (insert\ x\ t)\ ,$$

which can be proved by induction on $t$.