Functional Programming
Practicals 01: Definition and Proof by Induction

Shin-Cheng Mu

FLOLAC 2020

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

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

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

Case $xs := [\,]$:

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

Case $xs := x : xs$:

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

$$length\ (x : xs) + length\ ys$$

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 $xss := xs : xss$:

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

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 $xs := [\,]$:

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

Case $xs := x : xs$:

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

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*:

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

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:

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

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:

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

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 $xs := x : xs$:

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

$$
\begin{array}{ll}
= & \{\ \text{definition of } (\cdot)\ \} \\
 & \mathit{map}\ f\ (\mathbf{if}\ (p \cdot f)\ x\ \mathbf{then}\ x : \mathit{filter}\ (p \cdot f)\ \mathit{xs}\ \mathbf{else}\ \mathit{filter}\ (p \cdot f)\ \mathit{xs}) \\
= & \{\ \text{definition of } \mathit{filter}\ \} \\
 & \mathit{map}\ f\ (\mathit{filter}\ (p \cdot f)\ (x : \mathit{xs}))
\end{array}
$$

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 $\bot$? What additional constraint shall we impose on f to make the law true?

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

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

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

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

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

Case $n := 0$

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

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

$$\mathit{take}\ (\mathbf{1}_+\ n)\ [\,] +\!\!+ \mathit{drop}\ (\mathbf{1}_+\ n)\ [\,]$$

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

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

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

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

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

Solution:

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

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:

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

Induction on *xs*.

Case $xs := [\,]$:

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

Case $xs := y : ys$:

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

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:

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

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:

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

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

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

Case $\mathit{xs} := x : \mathit{xs}$:

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

10. Define $inits :: List\ a \rightarrow List\ (List\ a)$ that returns all prefixes of the input list.

$$inits\ \texttt{"abcde"} = [\texttt{""}, \texttt{"a"}, \texttt{"ab"}, \texttt{"abc"}, \texttt{"abcd"}, \texttt{"abcde"}].$$

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

Solution:

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

11. Define $tails :: List\ a \rightarrow List\ (List\ a)$ that returns all suffixes of the input list.

$$tails\ \texttt{"abcde"} = [\texttt{"abcde"}, \texttt{"bcde"}, \texttt{"cde"}, \texttt{"de"}, \texttt{"e"}, \texttt{""}].$$

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

Solution:

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

12. The function $splits :: List\ a \rightarrow List\ (List\ a, List\ a)$ 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 $f\ (ys, zs) = ...$ that matches pairs. Or you may simply use $(\lambda\ (ys, zs) \rightarrow ...)$.

Solution:

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

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

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

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:

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

Solution:

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

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:

$$\begin{aligned}
&\mathit{sublist} && :: \mathit{List}\ a \to \mathit{List}\ (\mathit{List}\ a)\\
&\mathit{sublist}\ [\,] && = [[\,]]\\
&\mathit{sublist}\ (x : xs) && = xss +\!\!+\ \mathit{map}\ (x :)\ xss \ \ ,\\
& && \quad \textbf{where}\ xss = \mathit{sublist}\ xs \ \ .
\end{aligned}$$

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

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

$$\textbf{data}\ \mathrm{ETree}\ a = \mathrm{Tip}\ a \mid \mathrm{Bin}\ (\mathrm{ETree}\ a)\ (\mathrm{ETree}\ a)$$

Define a function $\mathit{leaves} :: \mathrm{Tree}\ a \to \mathrm{List}\ a$ such that $\mathit{leaves}\ t$ returns all labels of t in a list. What is its worse case time complexity?

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

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

(a) Given $(\downarrow) :: \mathrm{Nat} \to \mathrm{Nat} \to \mathrm{Nat}$, which yields the smaller one of its arguments, define $\mathit{minT} :: \mathrm{ITree}\ \mathrm{Nat} \to \mathrm{Nat}$, 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:

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

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

Solution:

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

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

Solution:

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

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

Solution: Induction on t.

Case $t :=$ Null. Omitted.

Case $t :=$ Node $x\ t\ u$.

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

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