Programming Languages
Practicals 3. Definition and Proof by Induction

Shin-Cheng Mu

Autumn 2025

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

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

Solution: Prove by induction on the structure of xs.

Case $xs := [\,]$:

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

Case $xs := x : xs$:

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

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 $xss := [\,]$:

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

Case $xss := xs : xss$:

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

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 $xs := [\,]$:

$$sum\ ([\,] +\!\!+ ys)$$

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

Case $xs := x : xs$:

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

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:

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

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

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

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

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

   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:

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

We proceed by induction on xs.

Case $xs := [\,]$:

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

Case $xs := x : xs$:

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

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

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

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 $n := 0$

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

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

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

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

$$take\ (\mathbf{1}_+\ n)\ (x : xs) +\!\!+ drop\ (\mathbf{1}_+\ n)\ (x : xs)$$

$=$ { definitions of $take$ and $drop$ }
$(x : take\ n\ xs) +\!\!+ drop\ n\ xs$
$=$ { definition of $(+\!\!+)$ }
$x : take\ n\ xs +\!\!+ drop\ n\ xs$
$=$ { induction }
$x : xs$.

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 $xs := [\,]$:

$map\ (map\ f)\ (fan\ x\ [\,])$
$=$ { definition of fan }
$map\ (map\ f)\ [[x]]$
$=$ { definition of map }
$[[f\ x]]$
$=$ { definition of fan }
$fan(f\ x)\ [\,]$
$=$ { definition of fan }
$fan\ (f\ x)\ (map\ f\ [\,])$.

Case $xs := y : ys$:

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

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:

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

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 $xs := [\,]$:

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

Case $xs := x : xs$:

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

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:

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

11. Define $\mathit{tails} :: \mathit{List}\ a \to \mathit{List}\ (\mathit{List}\ a)$ 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:

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

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

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

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

Solution:

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

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

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

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:

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

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:

$$\mathbf{data}\ Tree\ a\ =\ \mathsf{Null} \mid \mathsf{Node}\ a\ (Tree\ a)\ (Tree\ a)\ .$$

(a) Given $(\downarrow) :: Nat \to Nat \to Nat$, which yields the smaller one of its arguments, define $minT :: Tree\ Nat \to Nat$, 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:

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

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

Solution:

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

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

Solution:

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

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

Solution: Induction on t.

Case $t :=$ Null. Omitted.

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

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

$$
\begin{aligned}
& n + (x \downarrow minT\ t \downarrow minT\ u) \\
= & \ \{ \text{ definition of } minT \ \} \\
& n + minT\ (\mathsf{Node}\ x\ t\ u) \ .
\end{aligned}
$$

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