Functional Programming
Practicals for 2. Definition and Proof by Induction

Shin-Cheng Mu

July 2018

1 Exercises

1. Prove the (very useful) *map*-fusion law: $map\ f \cdot map\ g = map\ (f \cdot g)$.
2. Prove that *length* distributes into $(\mathbin{+\!\!+})$:

$$length\ (xs \mathbin{+\!\!+} ys) = length\ xs + length\ ys \quad .$$

Solution: Prove by induction on the structure of xs.

Case $xs := [\,]$:

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

Case $xs := x : xs$:

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

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

3. Prove: $\mathit{sum} \cdot \mathit{concat} = \mathit{sum} \cdot \mathit{map}\ \mathit{sum}$. **Hint**: you will need a lemma stating that sum distributes over $\mathbin{+\!\!+}$. Write down that lemma and prove it too.

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 $xss := xs : xss$:

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

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

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

needs a separate proof by induction. Here it goes:

Case $xs := [\,]$:

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

Case $xs := x : xs$:

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

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

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

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

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

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

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

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:

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

We proceed by induction on xs.

Case $xs := []$:

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

Case $xs := x : xs$:

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

5. 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:

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

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

6. 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 $n := 0$

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

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

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

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

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

7. 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:

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

Solution:

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

8. 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:

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

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

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

Case $xs := y : ys$:

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

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

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

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

Solution:

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

10. Prove: $map\ (map\ f) \cdot perm = perm \cdot map\ f$. You may need previously proved results as lemmas.

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

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

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:

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

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

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

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

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

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

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

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

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

> **Solution:**
>
> $$\begin{array}{lcl} sublist & :: & List\ a \rightarrow List\ (List\ a) \\ sublist\ [\,] & = & [[\,]] \\ sublist\ (x:xs) & = & xss +\!\!+ map\ (x:)\ xss\ \ , \\ & & \mathbf{where}\ xss = sublist\ xs\ \ . \end{array}$$
>
> The righthand side could be $sublist\ xs +\!\!+ map\ (x:)\ (sublist\ xs)$ (but it could be much slower).

14. 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 $(\downarrow) :: Nat \rightarrow Nat \rightarrow Nat$, which yields the smaller one of its arguments, define $minT :: Tree\ Nat \rightarrow Nat$, 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:

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

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

Solution:

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

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

Solution:

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

(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 $t := \mathsf{Null}$. Omitted.

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

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

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