Programming Languages
Practicals 3. Definition and Proof by Induction

Shin-Cheng Mu

Autumn 2025

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

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

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

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 $filter$:

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

   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.

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

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

   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?

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

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:

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

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?).

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

$$\mathit{perms}\ [1,2,3] = [[1,2,3],[2,1,3],[2,3,1],[1,3,2],[3,1,2],[3,2,1]]\ .$$

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

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 $\mathit{map}\ g \cdot \mathit{concat} = \mathit{concat} \cdot \mathit{map}\ (\mathit{map}\ g)$.

10. Define $\mathit{inits} :: \mathit{List}\ a \to \mathit{List}\ (\mathit{List}\ a)$ 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.

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.

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,

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

    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 $(\lambda\ (\mathit{ys}, \mathit{zs}) \to \ldots)$.

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

$$\begin{aligned}&[[1,2,3,4,5],[1,2,4,3,5],[1,2,4,5,3],[1,4,2,3,5],[1,4,2,5,3],\\ &\ [1,4,5,2,3],[4,1,2,3,5],[4,1,2,5,3],[4,1,5,2,3],[4,5,1,2,3]].\end{aligned}$$

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 $sublist :: List\ a \rightarrow List\ (List\ a)$ 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.

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 $(\downarrow) :: Nat \rightarrow Nat \rightarrow Nat$, 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.)

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

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

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