Programming Languages: Functional Programming
Worksheet for 3. Definition and Proof by Induction

Shin-Cheng Mu

Autumn 2023

Finish the definitions.

1 Induction on Natural Numbers

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

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

$$\begin{array}{ll}
exp & :: Nat \to Nat \to Nat \\
exp\; b\; 0 & = \\
exp\; b\; (\mathbf{1}_+\, n) & =
\end{array}$$

2 Induction on Lists

$$\begin{array}{ll}
sum & :: List\; Int \to Int \\
sum\; [\,] & = \\
sum\; (x : xs) & =
\end{array}$$

$$\begin{array}{ll}
map & :: (a \to b) \to List\; a \to List\; b \\
map\; f\; [\,] & = \\
map\; f\; (x : xs) & =
\end{array}$$

$$\begin{array}{ll}
(+\!\!+) & :: List\; a \to List\; a \to List\; a \\
[\,] +\!\!+ ys & = \\
(x : xs) +\!\!+ ys & =
\end{array}$$

Prove: $xs +\!\!+ (ys +\!\!+ zs) = (xs +\!\!+ ys) +\!\!+ zs$.

Proof. Induction on xs.

Case $xs := [\,]$:

Case $xs := x : xs$:

$\square$

- The function *length* defined inductively:

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

- While $(+\!\!+)$ repeatedly applies $(:)$, the function *concat* repeatedly calls $(+\!\!+)$:

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

- $\mathit{filter}\ p\ xs$ keeps only those elements in xs that satisfy p.

$$
\begin{array}{ll}
\mathit{filter} & :: (a \to \mathit{Bool}) \to \mathit{List}\ a \to \mathit{List}\ a \\
\mathit{filter}\ p\ [\,] & = \\
\mathit{filter}\ p\ (x : xs) &
\end{array}
$$

- Recall take and drop, which we used in the previous exercise.

$$
\begin{array}{ll}
\mathit{take} & :: \mathit{Nat} \to \mathit{List}\ a \to \mathit{List}\ a \\
\mathit{take}\ 0\ xs & = \\
\mathit{take}\ (\mathbf{1}_{+}\ n)\ [\,] & = \\
\mathit{take}\ (\mathbf{1}_{+}\ n)\ (x : xs) & =
\end{array}
$$

-

$$
\begin{array}{ll}
\mathit{drop} & :: \mathit{Nat} \to \mathit{List}\ a \to \mathit{List}\ a \\
\mathit{drop}\ 0\ xs & = \\
\mathit{drop}\ (\mathbf{1}_{+}\ n)\ [\,] & = \\
\mathit{drop}\ (\mathbf{1}_{+}\ n)\ (x : xs) & =
\end{array}
$$

- $\mathit{takeWhile}\ p\ xs$ yields the longest prefix of xs such that p holds for each element.

$$
\begin{array}{ll}
\mathit{takeWhile} & :: (a \to \mathit{Bool}) \to \mathit{List}\ a \to \mathit{List}\ a \\
\mathit{takeWhile}\ p\ [\,] & = \\
\mathit{takeWhile}\ p\ (x : xs) &
\end{array}
$$

- $\mathit{dropWhile}\ p\ xs$ drops the prefix from xs.

$$
\begin{array}{ll}
\mathit{dropWhile} & :: (a \to \mathit{Bool}) \to \mathit{List}\ a \to \mathit{List}\ a \\
\mathit{dropWhile}\ p\ [\,] & = \\
\mathit{dropWhile}\ p\ (x : xs) &
\end{array}
$$

- List reversal.

$$
\begin{array}{ll}
\mathit{reverse} & :: \mathit{List}\ a \to \mathit{List}\ a \\
\mathit{reverse}\ [\,] & = \\
\mathit{reverse}\ (x : xs) & =
\end{array}
$$

- $\mathit{inits}\ [1,2,3] = [[\,],[1],[1,2],[1,2,3]]$

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

- $\mathit{tails}\ [1,2,3] = [[1,2,3],[2,3],[3],[\,]]$

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

- Some functions discriminate between several base cases. E.g.

$$\begin{array}{ll} \mathit{fib} & :: \mathit{Nat} \to \mathit{Nat} \\ \mathit{fib}\ 0 & = \\ \mathit{fib}\ 1 & = \\ \mathit{fib}\ (2+n) & = \end{array}$$

- E.g. the function merge merges two sorted lists into one sorted list:

$$\begin{array}{ll} \mathit{merge} & :: \mathit{List}\ \mathit{Int} \to \mathit{List}\ \mathit{Int} \to \mathit{List}\ \mathit{Int} \\ \mathit{merge}\ [\,]\ [\,] & = \\ \mathit{merge}\ [\,]\ (y : \mathit{ys}) & = \\ \mathit{merge}\ (x : \mathit{xs})\ [\,] & = \\ \mathit{merge}\ (x : \mathit{xs})\ (y : \mathit{ys}) & \end{array}$$

-

$$\begin{array}{ll} \mathit{zip} & :: \mathit{List}\ a \to \mathit{List}\ b \to \mathit{List}\ (a,b) \\ \mathit{zip}\ [\,]\ [\,] & = \\ \mathit{zip}\ [\,]\ (y : \mathit{ys}) & = \\ \mathit{zip}\ (x : \mathit{xs})\ [\,] & = \\ \mathit{zip}\ (x : \mathit{xs})\ (y : \mathit{ys}) & = \end{array}$$

- Non-structural induction. Example: merge sort.

$$\begin{array}{ll} \mathit{msort} & :: \mathit{List}\ \mathit{Int} \to \mathit{List}\ \mathit{Int} \\ \mathit{msort}\ [\,] & = \\ \mathit{msort}\ [x] & = \\ \mathit{msort}\ \mathit{xs} & = \end{array}$$

3 User Defined Inductive Datatypes

- This is a possible definition of internally labelled binary trees:

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

- on which we may inductively define functions:

$$
\begin{array}{lll}
sumT & :: & Tree\ Nat \to Nat \\
sumT\ \textsf{Null} & = & \\
sumT\ (\textsf{Node}\ x\ t\ u) & = &
\end{array}
$$