Programming Languages
Worksheet for 3. Definition and Proof by Induction

Shin-Cheng Mu

Oct. 17, 2018

Finish the definitions.

1 Induction on Natural Numbers

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

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

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

2 Induction on Lists

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

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

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

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

Proof. Induction on xs.

Case $xs := [\,]$:

Case $xs := x : xs$:

$\square$

- The function *length* defined inductively:

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

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

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

- *filter* p xs keeps only those elements in xs that satisfy p.

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

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

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

-

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

- *takeWhile* p xs yields the longest prefix of xs such that p holds for each element.

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

- $dropWhile\ p\ xs$ drops the prefix from xs.

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

- List reversal.

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

- $inits\ [1,2,3] = [[\,],[1],[1,2],[1,2,3]]$

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

- $tails\ [1,2,3] = [[1,2,3],[2,3],[3],[\,]]$

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

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

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

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

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

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

- Non-structural induction. Example: merge sort.

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

3 User Defined Inductive Datatypes

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

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

- on which we may inductively define functions:

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