Programming Languages: Functional Programming Practicals 4. Program Calculation

Shin-Cheng Mu

Autumn 2023

1. Let $descend$ be defined by:

   $$\begin{aligned}&descend :: \mathsf{Nat} \rightarrow \mathsf{List\ Nat}\\ &descend\ 0 = [\,]\\ &descend\ (\mathbf{1}_+\ n) = \mathbf{1}_+\ n : descend\ n \quad .\end{aligned}$$

   (a) Let $sumseries = sum \cdot descend$. Synthesise an inductive definition of $sumseries$.

   (b) The function $repeatN :: (\mathsf{Nat}, a) \rightarrow \mathsf{List}\ a$ is defined by

   $$repeatN\ (n, x) = map\ (const\ x)\ (descend\ n) \quad .$$

   Thus $repeatN\ (n, x)$ produces n copies of x in a list. E.g. $repeatN\ (3, \texttt{'a'}) =$ `"aaa"`. Calculate an inductive definition of $repeatN$.

   (c) The function $rld :: \mathsf{List}\ (\mathsf{Nat}, a) \rightarrow \mathsf{List}\ a$ performs run-length decoding:

   $$rld = concat \cdot map\ repeatN \quad .$$

   For example, $rld\ [(2, \texttt{'a'}), (3, \texttt{'b'}), (1, \texttt{'c'})] =$ `"aabbbc"`. Come up with an inductive defintion of rld.

2. There is another way to define pos such that $pos\ x\ xs$ yields the index of the first occurrence of x in xs:

   $$\begin{aligned}&pos :: \mathsf{Eq}\ a \Rightarrow a \rightarrow \mathsf{List}\ a \rightarrow \mathsf{Int}\\ &pos\ x = length \cdot takeWhile\ (x \not=)\end{aligned}$$

   (This pos behaves differently from the one in the lecture when x does not occur in xs.) Construct an inductive definition of pos.

3. Zipping and mapping.

   (a) Let $second\ f\ (x, y) = (x, f\ y)$. Prove that $zip\ xs\ (map\ f\ ys) = map\ (second\ f)\ (zip\ xs\ ys)$.

(b) Consider the following definition

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

such that

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

That is, each element in the input list is deleted in turns. Let $\mathit{select} :: \mathsf{List}\ a \to \mathsf{List}\ (a, \mathsf{List}\ a)$ be defined by $\mathit{select}\ xs = \mathit{zip}\ xs\ (\mathit{delete}\ xs)$. Come up with an inductive definition of select. **Hint**: you may find second useful.

(c) An alternative specification of delete is

$$\begin{aligned}
&\mathit{delete}\ xs = \mathit{map}\ (\mathit{del}\ xs)\ [0\,..\,\mathit{length}\ xs - 1] \\
&\quad \textbf{where}\ \mathit{del}\ xs\ i = \mathit{take}\ i\ xs \mathbin{+\!\!+} \mathit{drop}\ (1+i)\ xs\ \ ,
\end{aligned}$$

(here we take advantage of the fact that $[0\,..\,n]$ returns $[\,]$ when n is negative). From this specification, derive the inductive definition of delete given above. **Hint**: you may need the following property:

$$[0\,..\,n] = 0 : \mathit{map}\ (\mathbf{1}_+)\ [0\,..\,n-1], \quad \text{if } n \geqslant 0, \tag{1}$$

and the *map-fusion* law (2) given below.

4. Prove the following *map-fusion* law:

$$\mathit{map}\ f \cdot \mathit{map}\ g = \mathit{map}\ (f \cdot g)\ \ . \tag{2}$$

5. Assume that multiplication $(\times)$ is a constant-time operation. One possible definition for $\mathit{exp}\ m\ n = m^n$ could be:

$$\begin{aligned}
&\mathit{exp} :: \mathsf{Nat} \to \mathsf{Nat} \to \mathsf{Nat} \\
&\mathit{exp}\ m\ 0 \qquad\ \ = 1 \\
&\mathit{exp}\ m\ (\mathbf{1}_+\ n) = m \times \mathit{exp}\ m\ n\ \ .
\end{aligned}$$

Therefore, to compute $\mathit{exp}\ m\ n$, multiplication is called n times: $m \times m \ldots m \times 1$. Can we do better? Yet another way to represent a natural number is to use the binary representation.

(a) The function $\mathit{binary} :: \mathsf{Nat} \to \mathsf{List}\ \mathsf{Bool}$ returns the *reversed* binary representation of a natural number. For example:

$$\begin{aligned}
&\mathit{binary}\ 0 = [\,]\ \ , \\
&\mathit{binary}\ 1 = [\mathsf{T}]\ \ , \\
&\mathit{binary}\ 2 = [\mathsf{F}, \mathsf{T}]\ \ ,
\end{aligned}$$

$$binary\ 3 = [\mathsf{T}, \mathsf{T}]\ ,$$
$$binary\ 4 = [\mathsf{F}, \mathsf{F}, \mathsf{T}]\ ,$$

where T and F abbreviates True and False. Given the following functions:

$even :: Nat \to Bool$, returning true iff the input is even,

$odd :: Nat \to Bool$, returning true iff the input is odd, and

$div :: Nat \to Nat \to Nat$, for integral division,

define $binary$. You may just present the code.
Hint One possible implementation discriminates between 3 cases – the input is 0, the input is odd, and the input is even.

(b) Briefly explain in words whether your implementation of $binary$ terminates for all input in Nat, and why.

(c) Define a function $decimal$ :: List Bool $\to$ Nat that takes the reversed binary representation and returns the corresponding natural number. E.g. $decimal\ [\mathsf{T}, \mathsf{T}, \mathsf{F}, \mathsf{T}] = 11$. You may just present the code.

(d) Let $roll\ m = exp\ m \cdot decimal$. Assuming we have proved that $exp\ m\ n$ satisfies all arithmetic laws for m^n. Construct (with algebraic calculation) a definition of $roll$ that does not make calls to exp or $decimal$.

Remark If the fusion succeeds, we have derived a program computing m^n:

$$fastexp\ m = roll\ m \cdot binary.$$

The algorithm runs in time proportional to the length of the list generated by $binary$, which is $O(\log_2 n)$.

6. The following problem concerns calculating the sum $\sum_{i=0}^{n}(x_i \times y^i)$. Let geo be defined by:

$$geo\ y = 1 : map\ (y\times)\ (geo\ y)\ ,$$
$$horner\ y\ xs = sum\ (map\ mul\ (zip\ xs\ (geo\ y)))\ ,$$

where $mul\ (a, b) = a \times b$. Let $xs = [x_0, x_1, x_2 \ldots x_n]$, $horner\ y\ xs$ computes the sum $x_0 + x_1 \times y + x_2 \times y^2 + \cdots + x_n \times y^n$. (**Remark**: for those who familiar with currying, $mul = uncurry\ (\times)$.)

(a) Show that $mul \cdot second\ (y\times) = (y\times) \cdot mul$.

(b) Let $n = length\ xs$. Asymptotically (that is, in terms of the big-O notation), how many multiplications $(\times)$ one must perform to compute $horner\ y\ xs$?

(c) Prove that $sum \cdot map\ (y\times) = (y\times) \cdot sum$.

(d) Construct an inductive definition of $horner$ that uses only $O(n)$ multiplications to compute $horner\ y\ xs$. **Hint**: you will need a number of properties proved in the previous problems in this exercise, and perhaps some more properties.