Programming Languages: Functional Programming Practicals 5. Program Calculation

Shin-Cheng Mu

Autumn, 2023

1. Consider the internally labelled binary tree:

$$\mathbf{data}\ \mathsf{ITree}\ a = \mathsf{Null} \mid \mathsf{Node}\ a\ (\mathsf{ITree}\ a)\ (\mathsf{ITree}\ a)\ .$$

   (a) Define $sumT :: \mathsf{ITree\ Int} \to \mathsf{Int}$ that computes the sum of labels in an ITree.

   (b) A *baobab tree* is a kind of tree with very thick trunks. An Itree Int is called a baobab tree if every label in the tree is larger than the sum of the labels in its two subtrees. The following function determines whether a tree is a baobab tree:

$$\begin{aligned} &baobab :: \mathsf{ITree\ Int} \to \mathsf{Bool} \\ &baobab\ \mathsf{Null} &&= \mathsf{True} \\ &baobab\ (\mathsf{Node}\ x\ t\ u) &&= baobab\ t \wedge baobab\ u \wedge \\ &&&\quad x > (sumT\ t + sumT\ u)\ . \end{aligned}$$

   What is the time complexity of $baobab$? Define a variation of $baobab$ that runs in time proportional to the size of the input tree by tupling.

2. Recall the externally labelled binary tree:

$$\mathbf{data}\ \mathsf{Etree}\ a = \mathsf{Tip}\ a \mid \mathsf{Bin}\ (\mathsf{ETree}\ a)\ (\mathsf{ETree}\ a)\ .$$

   The function $size$ computes the size (number of labels) of a tree, while $repl\ \ t\ \ xs$ tries to relabel the tips of t using elements in xs. Note the use of $take$ and $drop$ in $repl$:

$$\begin{aligned} &size\ (\mathsf{Tip}\ \_) &&= 1 \\ &size\ (\mathsf{Bin}\ t\ u) &&= size\ t + size\ u\ . \end{aligned}$$

$$\begin{aligned} &repl :: \mathsf{ETree}\ a \to \mathsf{List}\ b \to \mathsf{ETree}\ b \\ &repl\ (\mathsf{Tip}\ \_)\ \ xs = \mathsf{Tip}\ (head\ xs) \\ &repl\ (\mathsf{Bin}\ t\ u)\ xs = \mathsf{Bin}\ (repl\ t\ (take\ n\ xs))\ (repl\ u\ (drop\ n\ xs)) \\ &\quad \mathbf{where}\ n = size\ t\ . \end{aligned}$$

   The function $repl$ runs in time $O(n^2)$ where n is the size of the input tree. Can we do better? Try discovering a linear-time algorithm that computes $repl$. **Hint**: try calculating the following function:

$$
\begin{aligned}
&repTail :: \mathsf{ETree}\ a \to \mathsf{List}\ b \to (\mathsf{ETree}\ b, \mathsf{List}\ b) \\
&repTail\ s\ xs = (???, ???)\ , \\
&\quad \mathbf{where}\ n = size\ s\ ,
\end{aligned}
$$

where the function $repTail$ returns a tree labelled by some prefix of xs, together with the suffix of xs that is not yet used (how to specify that formally?).

You might need properties including:

$$
\begin{aligned}
take\ m\ (take\ (m+n)\ xs) &= take\ m\ xs\ , \\
drop\ m\ (take\ (m+n)\ xs) &= take\ n\ (drop\ m\ xs)\ , \\
drop\ (m+n)\ xs &= drop\ n\ (drop\ m\ xs)\ .
\end{aligned}
$$

3. The function $tags$ returns all labels of an internally labelled binary tree:

$$
\begin{aligned}
&tags :: \mathsf{ITree}\ a \to \mathsf{List}\ a \\
&tags\ \mathsf{Null} = [\,] \\
&tags\ (\mathsf{Node}\ x\ t\ u) = tags\ t +\!\!+ [x] +\!\!+ tags\ u\ .
\end{aligned}
$$

Try deriving a faster version of $tags$ by calculating

$$
\begin{aligned}
&tagsAcc :: \mathsf{ITree}\ a \to \mathsf{List}\ a \to \mathsf{List}\ a \\
&tagsAcc\ t\ ys = tags\ t +\!\!+ ys\ .
\end{aligned}
$$

4. Recall the standard definition of factorial:

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

This program implicitly uses space linear to n in the call stack.

1. Introduce $factAcc\ n\ m = ...$ where m is an accumulating parameter.
2. Express $fact$ in terms of $factAcc$.
3. Construct a space efficient implementation of $factAcc$.

5. Define the following function $expAcc$:

$$
\begin{aligned}
&expAcc :: \mathsf{Nat} \to \mathsf{Nat} \to \mathsf{Nat} \to \mathsf{Nat} \\
&expAcc\ b\ n\ x = x \times exp\ b\ n\ .
\end{aligned}
$$

(a) Calculate a definition of $expAcc$ that uses only $O(\log n)$ multiplications to compute b^n. You may assume all the usual arithmetic properties about exponentials. **Hint**: consider the cases when n is zero, non-zero even, and odd.

(b) The derived implementation of $expAcc$ shall be tail-recursive. What imperative loop does it correspond to?

6. Recall the standard definition of Fibonacci:

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

Let us try to derive a linear-time, tail-recursive algorithm computing fib.

1. Given the definition $\mathit{ffib}\ n\ x\ y = \mathit{fib}\ n \times x + \mathit{fib}\ (\mathbf{1}_+\ n) \times y$, Express fib using ffib.
2. Derive a linear-time version of ffib.