Functional Programming
Practicals 02: Program Derivation

Shin-Cheng Mu

FLOLAC 2020

1. Let *descend* be defined by:

$$\begin{array}{l} \mathit{descend} :: \mathrm{Nat} \rightarrow \mathrm{List\ Nat} \\ \mathit{descend}\ 0 = [\,] \\ \mathit{descend}\ (\mathbf{1}_{+}\ n) = \mathbf{1}_{+}\ n : \mathit{descend}\ n \enspace . \end{array}$$

(a) Let $\mathit{sumseries} = \mathit{sum} \cdot \mathit{descend}$, synthesise an inductive definition of f.

Solution: It is immediate that $\mathit{sum}\ (\mathit{descend}\ 0) = 0$. For the inductive case we calculate:

$$\begin{array}{cl} & \mathit{sum}\ (\mathit{descend}\ (\mathbf{1}_{+}\ n)) \\ = & \{\ \text{definition of } \mathit{descend}\ \} \\ & \mathit{sum}\ ((\mathbf{1}_{+}\ n) : \mathit{descend}\ n) \\ = & \{\ \text{definition of } \mathit{sum}\ \} \\ & \mathbf{1}_{+}\ n + \mathit{sum}\ (\mathit{descend}\ n)) \\ = & \{\ \text{definition of } \mathit{sum}\ \} \\ & \mathbf{1}_{+}\ n\ + \mathit{sumseries}\ n \enspace . \end{array}$$

Thus we have

$$\begin{array}{lcl} \mathit{sumseries}\ 0 & = & 0 \\ \mathit{sumseries}\ (\mathbf{1}_{+}\ n) & = & \mathbf{1}_{+}\ n\ + \mathit{sumseries}\ n \enspace . \end{array}$$

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

$$\mathit{repeatN}\ (n, x) = \mathit{map}\ (\mathit{const}\ x)\ (\mathit{descend}\ n) \enspace .$$

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

Solution: It is immediate that $\mathit{repeatN}\ (0, x) = [\,]$. For the inductive case we calculate

$$
\begin{aligned}
& \mathit{repeatN}\ (\mathbf{1}_+\ n, x) \\
= {} & \quad \{ \text{ definition of } \mathit{repeatN} \ \} \\
& \mathit{map}\ (\mathit{const}\ x)\ (\mathit{descend}\ (\mathbf{1}_+\ n)) \\
= {} & \quad \{ \text{ definition of } \mathit{descend} \ \} \\
& \mathit{map}\ (\mathit{const}\ x)\ (\mathbf{1}_+\ n : \mathit{descend}\ n) \\
= {} & \quad \{ \text{ definition of } \mathit{map} \text{ and } \mathit{const} \ \} \\
& x : \mathit{map}\ (\mathit{const}\ x)\ (\mathit{descend}\ n) \\
= {} & \quad \{ \text{ definition of } \mathit{repeatN} \ \} \\
& x : \mathit{repeatN}\ (n, x) \quad .
\end{aligned}
$$

Thus we have

$$
\begin{aligned}
\mathit{repeatN}\ (0, \quad\ x) &= [\,] \\
\mathit{repeatN}\ (\mathbf{1}_+\ n, x) &= x : \mathit{repeatN}\ (n, x) \quad .
\end{aligned}
$$

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

$$\mathit{rld} = \mathit{concat} \cdot \mathit{map}\ \mathit{repeatN} \quad .$$

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

Solution: For the base case:

$$
\begin{aligned}
& \mathit{rld}\ [\,] \\
= {} & \quad \{ \text{ definition of } \mathit{rld} \ \} \\
& \mathit{concat}\ (\mathit{map}\ \mathit{repeatN}\ [\,]) \\
= {} & \quad \{ \text{ definitions of } \mathit{map} \text{ and } \mathit{concat} \ \} \\
& [\,]
\end{aligned}
$$

For the inductive case:

$$
\begin{aligned}
& \mathit{rld}\ ((n, x) : \mathit{xs}) \\
= {} & \quad \{ \text{ definition of } \mathit{rld} \ \} \\
& \mathit{concat}\ (\mathit{map}\ \mathit{repeatN}\ ((n, x) : \mathit{xs})) \\
= {} & \quad \{ \text{ definitions of } \mathit{map} \ \} \\
& \mathit{concat}\ (\mathit{repeatN}\ (n, x) : \mathit{map}\ \mathit{repeatN}\ \mathit{xs}) \\
= {} & \quad \{ \text{ definitions of } \mathit{concat} \ \} \\
& \mathit{repeatN}\ (n, x) \mathbin{+\!\!+} \mathit{concat}\ (\mathit{map}\ \mathit{repeatN}\ \mathit{xs}) \\
= {} & \quad \{ \text{ definition of } \mathit{rld} \ \} \\
& \mathit{repeatN}\ (n, x) \mathbin{+\!\!+} \mathit{rld}\ \mathit{xs} \quad .
\end{aligned}
$$

We have thus derived:

$$\begin{aligned} rld\ [\,] &= [\,] \\ rld\ ((n,x) : xs) &= repeatN\ (n,x) \mathbin{+\!\!+} rld\ xs\ . \end{aligned}$$

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

$$\begin{aligned} pos &:: Eq\ a \Rightarrow a \to List\ a \to Int \\ pos\ x &= length \cdot takeWhile\ (x \neq) \end{aligned}$$

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

Solution: It is immediate that $pos\ x\ [\,] = 0$. For the inductive case we calculate:

$$\begin{aligned} & pos\ x\ (y : xs) \\ = \; & length\ (takeWhile\ (x \neq)\ (y : xs)) \\ = \; & \{\ \text{definition of } takeWhile\ \} \\ & length\ (\textbf{if}\ x \neq y\ \textbf{then}\ y : takeWhile\ (x \neq)\ xs\ \textbf{else}\ [\,]) \\ = \; & \{\ \text{function application distributes into } \textbf{if} \text{ (for total functions)}\ \} \\ & \textbf{if}\ x \neq y\ \textbf{then}\ length\ (y : takeWhile\ (x \neq)\ xs)\ \textbf{else}\ length\ [\,] \\ = \; & \{\ \text{definition of } length\ \} \\ & \textbf{if}\ x \neq y\ \textbf{then}\ \mathbf{1}_{+}\ length\ (takeWhile\ (x \neq)\ xs)\ \textbf{else}\ 0 \\ = \; & \{\ \text{definition of } pos\ \} \\ & \textbf{if}\ x \neq y\ \textbf{then}\ \mathbf{1}_{+}\ pos\ x\ xs\ \textbf{else}\ 0\ . \end{aligned}$$

Thus we have constructed:

$$\begin{aligned} pos\ x\ [\,] &= 0 \\ pos\ x\ (y : xs) &= \textbf{if}\ x \neq y\ \textbf{then}\ \mathbf{1}_{+}\ pos\ x\ xs\ \textbf{else}\ 0\ . \end{aligned}$$

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

Solution: Recall one of the possible definitions of *zip*:

$$\begin{aligned}
&\mathit{zip}\ [\,]\ \mathit{ys} &&= [\,] \\
&\mathit{zip}\ (x : \mathit{xs})\ [\,] &&= [\,] \\
&\mathit{zip}\ (x : \mathit{xs})\ (y : \mathit{ys}) &&= (x, y) : \mathit{zip}\ \mathit{xs}\ \mathit{ys}.
\end{aligned}$$

Following the structure, we prove the proposition by induction on *xs* and *ys*. A tip for equational reasoning: it is usually easier to go from the more complex side to the simpler side, from the side with more structure to the side with less structure. Thus we start from the left-hand side.

Case $\mathit{xs} := [\,]$.

$$\begin{aligned}
& \mathit{map}\ (\mathit{second}\ f)\ (\mathit{zip}\ [\,]\ \mathit{ys}) \\
= & \quad \{\ \text{definition of } \mathit{zip}\ \} \\
& \mathit{map}\ (\mathit{second}\ f)\ [\,] \\
= & \quad \{\ \text{definition of } \mathit{map}\ \} \\
& [\,] \\
= & \quad \{\ \text{definition of } \mathit{zip}\ \} \\
& \mathit{zip}\ [\,]\ (\mathit{map}\ f\ \mathit{ys}).
\end{aligned}$$

Case $\mathit{xs} := x : \mathit{xs}$, $\mathit{ys} := [\,]$.

$$\begin{aligned}
& \mathit{map}\ (\mathit{second}\ f)\ (\mathit{zip}\ (x : \mathit{xs})\ [\,]) \\
= & \quad \{\ \text{definition of } \mathit{zip}\ \} \\
& \mathit{map}\ (\mathit{second}\ f)\ [\,] \\
= & \quad \{\ \text{definition of } \mathit{map}\ \} \\
& [\,] \\
= & \quad \{\ \text{definition of } \mathit{zip}\ \} \\
& \mathit{zip}\ (x : \mathit{xs})\ [\,] \\
= & \quad \{\ \text{definition of } \mathit{map}\ \} \\
& \mathit{zip}\ (x : \mathit{xs})\ (\mathit{map}\ f\ [\,]).
\end{aligned}$$

Case $\mathit{xs} := x : \mathit{xs}$, $\mathit{ys} := y : \mathit{ys}$.

$$\begin{aligned}
& \mathit{map}\ (\mathit{second}\ f)\ (\mathit{zip}\ (x : \mathit{xs})\ (y : \mathit{ys})) \\
= & \quad \{\ \text{definition of } \mathit{zip}\ \} \\
& \mathit{map}\ (\mathit{second}\ f)\ ((x, y) : \mathit{zip}\ \mathit{xs}\ \mathit{ys}) \\
= & \quad \{\ \text{definition of } \mathit{map}\ \} \\
& \mathit{second}\ f\ (x, y) : \mathit{map}\ (\mathit{second}\ f)\ (\mathit{zip}\ \mathit{xs}\ \mathit{ys})
\end{aligned}$$

$$
\begin{aligned}
&= \quad \{ \text{ definition of } \mathit{second} \ \} \\
&\quad (x, f\ y) : \mathit{map}\ (\mathit{second}\ f)\ (\mathit{zip}\ \mathit{xs}\ \mathit{ys}) \\
&= \quad \{ \text{ induction } \} \\
&\quad (x, f\ y) : \mathit{zip}\ \mathit{xs}\ (\mathit{map}\ f\ \mathit{ys}) \\
&= \quad \{ \text{ definition of } \mathit{zip} \ \} \\
&\quad \mathit{zip}\ (x : \mathit{xs})\ (f\ y : \mathit{map}\ f\ \mathit{ys}) \\
&= \quad \{ \text{ definition of } \mathit{map} \ \} \\
&\quad \mathit{zip}\ (x : \mathit{xs})\ (\mathit{map}\ f\ (y : \mathit{ys})).
\end{aligned}
$$

(b) Consider the following definition

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

such that

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

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

Solution: The base case $[\,]$ is immediate. For the inductive case:

$$
\begin{aligned}
&\quad \mathit{select}\ (x : \mathit{xs}) \\
&= \quad \{ \text{ definition of } \mathit{select} \ \} \\
&\quad \mathit{zip}\ (x : \mathit{xs})\ (\mathit{delete}\ (x : \mathit{xs})) \\
&= \quad \{ \text{ definition of } \mathit{delete} \ \} \\
&\quad \mathit{zip}\ (x : \mathit{xs})\ (\mathit{xs} : \mathit{map}\ (x{:})\ (\mathit{delete}\ \mathit{xs})) \\
&= \quad \{ \text{ definition of } \mathit{zip} \ \} \\
&\quad (x, \mathit{xs}) : \mathit{zip}\ \mathit{xs}\ (\mathit{map}\ (x{:})\ (\mathit{delete}\ \mathit{xs})) \\
&= \quad \{ \text{ property proved above } \} \\
&\quad (x, \mathit{xs}) : \mathit{map}\ (\mathit{second}\ (x{:}))\ (\mathit{zip}\ \mathit{xs}\ (\mathit{delete}\ \mathit{xs})) \\
&= \quad \{ \text{ definition of } \mathit{select} \ \} \\
&\quad (x, \mathit{xs}) : \mathit{map}\ (\mathit{second}\ (x{:}))\ (\mathit{select}\ \mathit{xs}) \quad .
\end{aligned}
$$

We thus have

$$
\begin{aligned}
&\mathit{select}\ [\,] && = [\,] \\
&\mathit{select}\ (x : \mathit{xs}) && = (x, \mathit{xs}) : \mathit{map}\ (\mathit{second}\ (x{:}))\ (\mathit{select}\ \mathit{xs}) \quad .
\end{aligned}
$$

(c) An alternative specification of *delete* is

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

(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 (**??**) given below.

Solution:

$$\begin{array}{ll} & \mathit{delete}\ (x : \mathit{xs}) \\ = & \quad \{\ \text{definition of } \mathit{delete}\ \} \\ & \mathit{map}\ (\mathit{del}\ (x : \mathit{xs}))\ [0\,..\,\mathit{length}\ (x : \mathit{xs}) - 1] \\ = & \quad \{\ \text{defintion of } \mathit{length}\text{, arithmetics}\ \} \\ & \mathit{map}\ (\mathit{del}\ (x : \mathit{xs}))\ [0\,..\,\mathit{length}\ \mathit{xs}] \\ = & \quad \{\ \mathit{length}\ \mathit{xs} \geqslant 0\text{, by } (\mathbf{??})\ \} \\ & \mathit{map}\ (\mathit{del}\ (x : \mathit{xs}))\ (0 : \mathit{map}\ (\mathbf{1}_{+})\ [0\,..\,\mathit{length}\ \mathit{xs} - 1]) \\ = & \quad \{\ \text{definition of } \mathit{map}\ \} \\ & \mathit{del}\ (x : \mathit{xs})\ 0 : \mathit{map}\ (\mathit{del}\ (x : \mathit{xs}))\ (\mathit{map}\ (\mathbf{1}_{+})\ [0\,..\,\mathit{length}\ \mathit{xs} - 1]) \\ = & \quad \{\ \text{map fusion } (\mathbf{??})\ \} \\ & \mathit{del}\ (x : \mathit{xs})\ 0 : \mathit{map}\ (\mathit{del}\ (x : \mathit{xs}) \cdot (\mathbf{1}_{+}))\ [0\,..\,\mathit{length}\ \mathit{xs} - 1] \end{array}$$

Now we pause for a while to inspect $\mathit{del}\ (x{:}\mathit{xs})$. Apparently, $\mathit{del}\ (x{:}\mathit{xs})\ 0 = \mathit{xs}$. For $\mathit{del}\ (x : \mathit{xs}) \cdot (\mathbf{1}_{+})$ we calculate:

$$\begin{array}{ll} & (\mathit{del}\ (x : \mathit{xs}) \cdot (\mathbf{1}_{+}))\ i \\ = & \quad \{\ \text{definition of } (\cdot)\ \} \\ & \mathit{del}\ (x : \mathit{xs})\ (\mathbf{1}_{+}\ i) \\ = & \quad \{\ \text{definition of } \mathit{del}\ \} \\ & \mathit{take}\ (\mathbf{1}_{+}\ i)\ (x : \mathit{xs}) +\!\!+ \mathit{drop}\ (\mathbf{1}_{+}\ (\mathbf{1}_{+}\ i))\ (x : \mathit{xs}) \\ = & \quad \{\ \text{definitions of } \mathit{take} \text{ and } \mathit{drop}\ \} \\ & x : \mathit{take}\ i\ \mathit{xs} +\!\!+ \mathit{drop}\ (\mathbf{1}_{+}\ i)\ \mathit{xs} \\ = & \quad \{\ \text{definition of } \mathit{del}\ \} \\ & x : \mathit{del}\ \mathit{xs}\ i \\ = & \quad \{\ \text{definition of } (\cdot)\ \} \\ & ((x{:}) \cdot \mathit{del}\ \mathit{xs})\ i\ \ . \end{array}$$

We resume the calculation:

$$
\begin{aligned}
& \quad \mathit{del}\ (x : xs)\ 0 : \mathit{map}\ (\mathit{del}\ (x : xs) \cdot (\mathbf{1}_{+}))\ [0 \mathinner{..} \mathit{length}\ xs - 1] \\
= & \quad \{\ \text{calculation above}\ \} \\
& \quad xs : \mathit{map}\ ((x{:}) \cdot \mathit{del}\ xs)\ [0 \mathinner{..} \mathit{length}\ xs - 1] \\
= & \quad \{\ \text{map fusion } (\mathbf{??})\ \} \\
& \quad xs : \mathit{map}\ (x{:})\ (\mathit{map}\ (\mathit{del}\ xs)\ [0 \mathinner{..} \mathit{length}\ xs - 1]) \\
= & \quad \{\ \text{definition of } \mathit{delete}\ \} \\
& \quad xs : \mathit{map}\ (x{:})\ (\mathit{delete}\ xs) \ .
\end{aligned}
$$

We have thus derived the first, inductive definition of *delete*.

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

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

Solution:

$$
\begin{aligned}
& \quad \mathit{map}\ f \cdot \mathit{map}\ g = \mathit{map}\ (f \cdot g) \\
\equiv & \quad \{\ \text{extensional equality}\ \} \\
& \quad (\forall xs :: (\mathit{map}\ f \cdot \mathit{map}\ g)\ xs = \mathit{map}\ (f \cdot g)\ xs) \\
\equiv & \quad \{\ \text{definition of } (\cdot)\ \} \\
& \quad (\forall xs :: (\mathit{map}\ f\ (\mathit{map}\ g\ xs) = \mathit{map}\ (f \cdot g)\ xs).
\end{aligned}
$$

We prove the proposition by induction on *xs*.

Case $xs := [\,]$. Omitted.

Case $xs := x : xs$.

$$
\begin{aligned}
& \quad \mathit{map}\ f\ (\mathit{map}\ g\ (x : xs)) \\
= & \quad \{\ \text{definition of } \mathit{map}\text{, twice}\ \} \\
& \quad f\ (g\ x) : \mathit{map}\ f\ (\mathit{map}\ g\ xs) \\
= & \quad \{\ \text{induction}\ \} \\
& \quad f\ (g\ x) : \mathit{map}\ (f \cdot g)\ xs \\
= & \quad \{\ \text{definition of } (\cdot)\ \} \\
& \quad (f \cdot g)\ x : \mathit{map}\ (f \cdot g)\ xs \\
= & \quad \{\ \text{definition of } \mathit{map}\ \} \\
& \quad \mathit{map}\ (f \cdot g)\ (x : xs).
\end{aligned}
$$

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

$$
\begin{aligned}
&exp && :: Nat \to Nat \to Nat\\
&exp\ m\ 0 && = 1\\
&exp\ m\ (1+n) && = m \times exp\ m\ n
\end{aligned}
$$

Therefore, to compute $exp\ m\ n$, multiplication is called n times: $m \times m \times ... \times m \times 1$. Can we do better?

Yet another way to represent a natural number is to use the binary representation.

(a) The function $binary :: Nat \to [Bool]$ returns the *reversed* binary representation of a natural number. For example:

$$
\begin{aligned}
&binary\ 0 = [\,],\\
&binary\ 1 = [T],\\
&binary\ 2 = [F, T],\\
&binary\ 3 = [T, T],\\
&binary\ 4 = [F, F, T],
\end{aligned}
$$

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.

Solution:

$$
\begin{aligned}
&binary && :: Nat \to List\ Bool\\
&binary\ 0 && = [\,]\\
&binary\ n && \mid even\ n = False : binary\ (n\ \text{`}div\text{`}\ 2)\\
& && \mid odd\ n = True : binary\ ((n-1)\ \text{`}div\text{`}\ 2)
\end{aligned}
$$

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

Solution: All non-zero natural numbers strictly decreases when being divided by 2, and thus we eventually reaches the base case for 0.

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

Solution:

$$
\begin{array}{ll}
decimal & :: List\ Bool \rightarrow Nat \\
decimal\ [\,] & = 0 \\
decimal\ (False : xs) & = 2 \times decimal\ xs \\
decimal\ (True : xs) & = 1 + (2 \times decimal\ xs)
\end{array}
$$

(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*.

Solution: Let's calculate $roll\ m\ xs = exp\ m\ (decimal\ xs)$ by distinguishing between the three cases of n: **Case** $xs := [\,]$:

$$
\begin{array}{ll}
 & roll\ m\ [\,] \\
= & exp\ m\ (decimal\ [\,]) \\
= & \{\ \text{definition of } decimal\ \} \\
 & exp\ m\ 0 \\
= & \{\ \text{definition of } exp\ \} \\
 & 1
\end{array}
$$

Case $xs = False : xs$:

$$
\begin{array}{ll}
 & roll\ m\ (False : xs) \\
= & \{\ \text{definition of } roll\ \} \\
 & exp\ m\ (decimal\ (False : xs)) \\
= & \{\ \text{definition of } decimal\ \} \\
 & exp\ m\ (2 \times decimal\ xs) \\
= & \{\ \text{arithmetic: } m^{2n} = (m^2)^n\ \} \\
 & exp\ (m \times m)\ (decimal\ xs) \\
= & \{\ \text{definition of } roll\ \} \\
 & roll\ (m \times m)\ xs
\end{array}
$$

Case $xs = True : xs$:

$$
roll\ m\ (True : xs)
$$

$$
\begin{aligned}
&= \quad \{ \text{ definition of } \mathit{roll} \ \} \\
&\quad\ \mathit{exp}\ m\ (\mathit{decimal}\ (\mathit{True} : \mathit{xs})) \\
&= \quad \{ \text{ definition of } \mathit{decimal} \ \} \\
&\quad\ \mathit{exp}\ m\ (1 + 2 \times \mathit{decimal}\ \mathit{xs}) \\
&= \quad \{ \text{ definition of } \mathit{exp} \ \} \\
&\quad\ m \times \mathit{exp}\ m\ (2 \times \mathit{decimal}\ \mathit{xs}) \\
&= \quad \{ \text{ arithmetic: } m^{2n} = (m^2)^n \ \} \\
&\quad\ m \times \mathit{exp}\ (m \times m)\ (\mathit{decimal}\ \mathit{xs}) \\
&= \quad \{ \text{ definition of } \mathit{roll} \ \} \\
&\quad\ m \times \mathit{roll}\ (m \times m)\ \mathit{xs}
\end{aligned}
$$

We have thus constructed:

$$
\begin{aligned}
&\mathit{roll}\ m\ [\,] &&= 1 \\
&\mathit{roll}\ m\ (\mathit{False} : \mathit{xs}) &&= \mathit{roll}\ (m \times m)\ \mathit{xs} \\
&\mathit{roll}\ m\ (\mathit{True} : \mathit{xs}) &&= m \times \mathit{roll}\ (m \times m)\ \mathit{xs}
\end{aligned}
$$

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

$$\mathit{fastexp}\ m = \mathit{roll}\ m \cdot \mathit{binary}\,.$$

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

6. Recall the internally labelled binary tree:

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

   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 (where $\mathit{sumT} :: \mathrm{ITree\ Int} \rightarrow \mathrm{Int}$ computes the sum of labels in a tree):

   $$
   \begin{aligned}
   &\mathit{baobab} :: \mathrm{ITree\ Int} \rightarrow \mathrm{Bool} \\
   &\mathit{baobab}\ \mathrm{Null} &&= \mathrm{True} \\
   &\mathit{baobab}\ (\mathrm{Node}\ x\ t\ u) &&= \mathit{baobab}\ t \wedge \mathit{baobab}\ u \wedge \\
   & &&\qquad x > (\mathit{sumT}\ \ t + \mathit{sumT}\ \ u)\ \ .
   \end{aligned}
   $$

   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.

7. Recall the externally labelled binary tree:

$$\textbf{data}\ \mathrm{Etree}\ a = \mathrm{Tip}\ a \mid \mathrm{Bin}\ (\mathrm{ETree}\ a)\ (\mathrm{ETree}\ a)\ \ .$$

   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*:

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

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

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

$$\begin{aligned}
&\mathit{repTail} :: \mathrm{ETree}\ a \to \mathrm{List}\ b \to (\mathrm{ETree}\ b \times \mathrm{List}\ b)\\
&\mathit{repTail}\ s\ xs = (\mathit{repl}\ s\ (\mathit{take}\ n\ xs), \mathit{drop}\ n\ xs)\ \ ,\\
&\quad \textbf{where}\ n = \mathit{size}\ s\ \ .
\end{aligned}$$

   You might need properties including:

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

8. The function *tags* returns all labels of an internally labelled binary tree:

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

   Try deriving a faster version of *tags* by Calculating

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

9. Define the following function *expAcc*:

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

   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.