

## Exercises on Lecture 1

**Exercise 1.** Huffman coding is often claimed to be ‘optimal’, in that it assigns codes to symbols to give the minimal average (weighted by frequency) code length. So how can Huffman be bettered?

**Exercise 2.** Prove the following:

$$x \text{ within } (int_1 \triangleright int_2) \Rightarrow x \text{ within } int_1$$

**Exercise 3.** Show that  $\triangleright$  is associative with identity *unit*.

**Exercise 4.** Define an inverse  $\triangleleft$  (‘widen’) of  $\triangleright$  such that

$$(int_1 \triangleright int_2) \triangleleft int_1 = int_2$$

**Exercise 5.** Prove the fusion theorem for *foldr*: For strict *h*,

$$h \cdot \text{foldr } f \ e = \text{foldr } f' \ e'$$

provided  $h \ e = e'$  and  $h (f \ x \ z) = f' \ x (h \ x)$  for every *x* and *z*.

**Exercise 6.** By defining *map* as an instance of *foldr*, prove *map fusion*:

$$\text{foldr } f \ e \cdot \text{map } g = \text{foldr } (f \cdot g) \ e$$

**Exercise 7.** Why don’t the universal property and the fusion theorem for *foldr* hold for non-strict *h*? What happens to the First Duality Theorem for infinite or partial lists?

**Exercise 8.** ‘Parallel loops’ may also be fused into one:

$$h \ xs = (\text{foldr } f_1 \ e_1 \ xs, \text{foldr } f_2 \ e_2 \ xs)$$

then

$$h = \text{foldr } f \ (e_1, e_2)$$

where  $f \ x \ (z_1, z_2) = (f_1 \ x \ z_1, f_2 \ x \ z_2)$ . For example, *average* = *uncurry div* · *sumlenght*, where *sumlenght* *xs* = (*sum xs*, *length xs*). This is sometimes known as the ‘Banana Split Theorem’ (why?). Prove the theorem, using the universal property of *foldr* again.

**Exercise 9.** *foldl* can be expressed in terms of *foldr*:

$$\text{foldl } f = \text{flip } (\text{foldr } (\text{comp } f) \ \text{id}) \ \textbf{where } \text{comp } f \ x \ u = u \cdot \text{flip } f \ x$$

Verify this claim, and hence (from the universal property of *foldr*) derive the following universal property of *foldl*: for *h* strict in its second argument,

$$h = \text{foldl } f \ \equiv \ h \ e \ [] = e \wedge h \ e \ (x : xs) = h \ (f \ e \ x) \ xs$$

**Exercise 10.** Using the universal property of *foldl*, prove the lemma about encoding as a *foldl*:

$$\text{foldl } (\triangleright) \text{ unit} \cdot \text{encodeSyms } m = \text{snd} \cdot \text{foldl } \text{step } (m, \text{unit})$$

where

$$\text{step } (m, \text{int}) s = (\text{newModel } m s, \text{int} \triangleright \text{encodeSym } m s)$$

**Exercise 11.** Here is a fusion theorem for *foldl*: for strict *h*,

$$h \cdot \text{foldl } f e = \text{foldl } f' (h e) \text{ for any } e$$

iff, for all *e*,

$$h \cdot f e = f' (h e)$$

Prove this theorem, by induction on the list.

**Exercise 12.** Define *begins* :: *Eq*  $\alpha$   $\rightarrow$  [ $\alpha$ ]  $\rightarrow$  [ $\alpha$ ]  $\rightarrow$  *Bool*.

**Exercise 13.** What alternative is there to returning the length of the message to indicate when to stop decoding? What are the advantages and disadvantages of this alternative?

## Exercises on Lecture 2

Exercise 14. Show

$$\begin{aligned}(2 \times l, 2 \times r) &= (0, \frac{1}{2}) \triangleleft (l, r) \\ (2 \times l - 1, 2 \times r - 1) &= (\frac{1}{2}, 1) \triangleleft (l, r)\end{aligned}$$

Exercise 15. *toBits* (*l*, *r*) does not return the shortest binary fraction within [*l*..*r*]; what definition does?

Exercise 16. Show that

$$\text{foldr pack } (\frac{1}{2}) \text{ } bs = \text{foldr pack } 0 \text{ } (bs ++ [1])$$

Exercise 17. Show that

$$\begin{aligned}\text{fromBits } bs &= \text{mean } (\text{foldr pack } 0 \text{ } bs, \text{foldr pack } 1 \text{ } bs) \\ \text{where mean } (x, y) &= (x + y) / 2\end{aligned}$$

Exercise 18. Show that

$$\begin{aligned}(\text{foldr pack } 0 \text{ } bs, \text{foldr pack } 1 \text{ } bs) &= \text{foldl } (\triangleright) \text{ unit } (\text{map encodeBit } bs) \\ \text{where encodeBit } b &= (b/2, (b+1)/2)\end{aligned}$$

Exercise 19. Verify that the hylo definitions of *pick* and *size* are indeed equivalent to the specifications given earlier.

Exercise 20. What happens to the proof principle for hylos when  $h \ x = \perp$ ?

Exercise 21. Note that *encode*<sub>1</sub> is an *unfoldr* after a fold (a *foldl* in this instance). This is a kind of dual to hylomorphisms. This pattern hasn't received much study, but one could say that it captures many *changes of data representation*, such as those for dealing with *structure clashes*: the input is read in from one format to some intermediate representation, and written out in another format. Write the functions

$$\begin{aligned}\text{fromBase} &:: \text{Integer} \rightarrow [\text{Integer}] \rightarrow \text{Ratio Integer} \\ \text{toBase} &:: \text{Integer} \rightarrow \text{Ratio Integer} \rightarrow [\text{Integer}]\end{aligned}$$

which, given a base, convert a sequence of digits into a fraction and back again, as a *foldl* and an *unfoldr* respectively. Hence the function that converts a digit sequence from one base to another is an instance of this pattern.

Exercise 22. Write the functions

$$\begin{aligned}\text{writeLines} &:: \text{Int} \rightarrow [\alpha] \rightarrow [[\alpha]] \\ \text{readLines} &:: [[\alpha]] \rightarrow [\alpha]\end{aligned}$$

that break a list into lines of bounded length, and join it back together again, as an *unfoldr* and a *foldl* respectively. (Think of the intermediate format as a queue.) Hence the function that 'repartitions' a list of lines of one width into another width is another instance of this pattern.

**Exercise 23.** *run-length encoding* consists of partitioning a list into runs of equal elements, then encoding each run by its length and its common element. Again, the intermediate format is a queue. Show that this is another instance of the same pattern.

**Exercise 24.** *heapsort* sorts by inserting elements one by one into a *heap* (a node-labelled binary tree in which every label is no smaller than any label below it), then removing them one by one. Show that this is yet another instance of the same pattern.

**Exercise 25.** The Haskell standard prelude defines a function *span*, which takes a predicate  $p$  and a list, and splits the list into a longest initial segment all of whose elements satisfy  $p$ , and a remainder. For example, *span isAlpha* “abc:def” = (“abc”, “:def”). Write *span* as an instance of *evolve*.

**Exercise 26.** Prove that

$$\text{evolve } f \ z = (\text{unfoldr } f \ z, \text{loop } (\text{fmap } \text{snd} \cdot f) \ z)$$

(Hint: write *evolve* as a hylomorphism.)

**Exercise 27.** What happens to the streaming theorem for partial or infinite lists?

**Exercise 28.** Show that streaming condition for *nextBit* and  $\triangleright$  follows from associativity of  $\triangleright$  (Exercise 3) and the fact that  $\text{int}_1 \triangleright \text{int}_2$  is contained in  $\text{int}_1$ .

## Exercises on Lecture 3

**Exercise 29.** Show that if  $(y \oplus w) \ominus y = w$ , then

$$\text{foldl } (\ominus) (\text{foldr } (\oplus) e (ys ++ zs)) ys = zs$$

**Exercise 30.** Give an example of an element of  $[\alpha]$  that does not have  $[]$  as a prefix.

**Exercise 31.** Define *nextBitM*. Why doesn't *nextBitM* appear in the definition of *decode<sub>2</sub>*?

**Exercise 32.** Prove the stream fusion result by filling in the dots:

$$\begin{aligned} & k (\text{stream } f \ g \ z \ xs) \\ = & \quad \{\text{definition of } \text{stream}\} \\ & \dots \\ = & \quad \{\text{distributivity of } k\} \\ & \dots \\ = & \quad \{\text{induction}\} \\ & \dots \\ = & \quad \{\text{fusion condition on } g\} \\ & \dots \\ = & \quad \{\text{fusion condition on } f\} \\ & \dots \end{aligned}$$

**Exercise 33.** Prove that  $\text{contract} \cdot \text{expand} = \text{contract}$ . Why don't we have  $\text{contract} \cdot \text{expand} = \text{id}$ ?

**Exercise 34.** Prove that

$$\begin{aligned} \text{contract } (n, \text{int}_1 \triangleright \text{int}_2) &= \text{contract } (n, \text{int}_1) \triangleright \text{int}_2 \\ \text{contract } (\text{enarrow } ei \ \text{int}) &= \text{contract } ei \triangleright \text{int} \end{aligned}$$

**Exercise 35.** Prove that

$$\begin{aligned} \text{rescale } (n, x) \leq 1/2 &\equiv x \leq 1/2 \\ \text{rescale } (n, x) \geq 1/2 &\equiv x \geq 1/2 \end{aligned}$$

**Exercise 36.** Prove that

$$\text{nextBits } ei = \text{Nothing} \Rightarrow \text{nextBit } (\text{contract } ei) = \text{Nothing}$$

Hence show that, if  $\text{nextBits } ei = \text{Nothing}$ , then

$$\text{evolve } \text{nextBits } ei = (bss, ei_1) \Rightarrow \text{evolve } \text{nextBit } ei = (\text{concat } bss, \text{contract } ei_1)$$

The following exercises establish the above fusion condition in the remaining cases.

**Exercise 37.** Suppose  $r \leq 1/2$ . Show that

$$\begin{aligned} \text{nextBits } (n, (l, r)) &= \text{Just } (0 : \text{replicate } n \ 1, (0, (2 \times l, 2 \times r))) \\ \text{nextBit } (\text{contract } (n, (l, r))) &= \text{Just } (0, (2 \times \text{rescale } (n, l), 2 \times \text{rescale } (n, r))) \end{aligned}$$

Hence it is sufficient to show that

$$\begin{aligned} &\text{evolve nextBit } (2 \times \text{rescale } (n, l), 2 \times \text{rescale } (n, r)) \\ &= (\text{replicate } n \ 1 \ ++ \ \text{bs}, \text{int}) \\ &\text{where } (\text{bs}, \text{int}) = \text{evolve } (2 \times l, 2 \times r) \end{aligned}$$

What is the corresponding proof obligation in the case  $1/2 \leq l$ ?

**Exercise 38.** Prove that

$$\begin{aligned} 2 \times \text{rescale } (n + 1, x) &= \text{rescale } (n, x) + 1/2 \\ 2 \times \text{rescale } (n + 1, x) - 1 &= \text{rescale } (n, x) - 1/2 \end{aligned}$$

Recall that if  $(n, (l, r))$  is an expanded interval, then

$$\text{contract } (n, (l, r)) = (\text{rescale } (n, l), \text{rescale } (n, r))$$

so  $0 \leq \text{rescale } (n, l) < \text{rescale } (n, r) \leq 1$ . Hence show that

$$\begin{aligned} 2 \times \text{rescale } (n + 1, l) &\geq 1/2 \\ 2 \times \text{rescale } (n + 1, r) - 1 &\leq 1/2 \end{aligned}$$

**Exercise 39.** Prove that if  $r \leq 1/2$ , then

$$\begin{aligned} &\text{evolve nextBit } (2 \times \text{rescale } (n + 1, l), 2 \times \text{rescale } (n + 1, r)) \\ &= (\text{replicate } n \ 1 \ ++ \ \text{bs}, \text{int}) \end{aligned}$$

where  $(\text{bs}, \text{int}) = \text{evolve } (2 \times l, 2 \times r)$ .

Similarly, prove that if  $1/2 \leq l$ , then

$$\begin{aligned} &\text{evolve nextBit } (2 \times \text{rescale } (n + 1, l) - 1, 2 \times \text{rescale } (n + 1, r) - 1) \\ &= (\text{replicate } n \ 1 \ ++ \ \text{bs}, \text{int}) \end{aligned}$$

where  $(\text{bs}, \text{int}) = \text{evolve } (2 \times l - 1, 2 \times r - 1)$ .

Hence complete the proof that

$$\text{evolve nextBits } ei = (\text{bss}, ei_1) \Rightarrow \text{evolve nextBit } ei = (\text{concat } \text{bss}, \text{contract } ei_1)$$

## Exercises on Lecture 4

**Exercise 40.** Prove that

$$(\forall p, q : 0 \leq p < q \leq d : \lfloor (r - l) \times p/d \rfloor < \lfloor (r - l) \times q/d \rfloor)$$

if and only if  $d \leq r - l$ .

**Exercise 41.** According to the Haskell Report, the finite-precision type *Int* covers at least the range  $[-2^{29}, 2^{29} - 1]$ . What are suitable choices for  $w$  and  $d$ ?

**Exercise 42.** Instead of using semi-open intervals  $[l..r)$  we could use a closed interval  $[l..r - 1]$ . What modifications are required to the definitions of *encode* and *decode* and why should such a representation have an advantage over the semi-open one?

**Exercise 43.** Imagine a static model of three equiprobable symbols A, B and C, so that B is assigned the range  $[1/3..2/3)$ . Suppose a message of a billion B's is to be encoded. What is the output? How big does  $n$  get in the definition of *expand*?

**Exercise 44.** Defining

$$\begin{aligned} \text{step } b \ (x, ds) &= (y, d : ds) \\ &\quad \textbf{where } (y, d) = (x + b \times w) \ \underline{\text{divMod}} \ 2 \end{aligned}$$

Prove that

$$\text{foldr } \text{step } (w/2, []) \cdot \text{concat} = \text{foldr } (\oplus) (w/2, [])$$

where  $bs \oplus (x, ds) = \text{foldr } \text{step } (x, ds) \ bs$ .

**Exercise 45.** In the definition of *decode* we find the local definitions

$$\begin{aligned} (x', ds') &= \text{foldl } (\ominus) (x, ds) \ bss \\ (bss, ei') &= \text{evolve } \text{nextBits } ei \end{aligned}$$

Define a function *absorb* so that these two definitions can be replaced by

$$((x', ds'), ei') = \text{absorb } ((x, ds), ei)$$