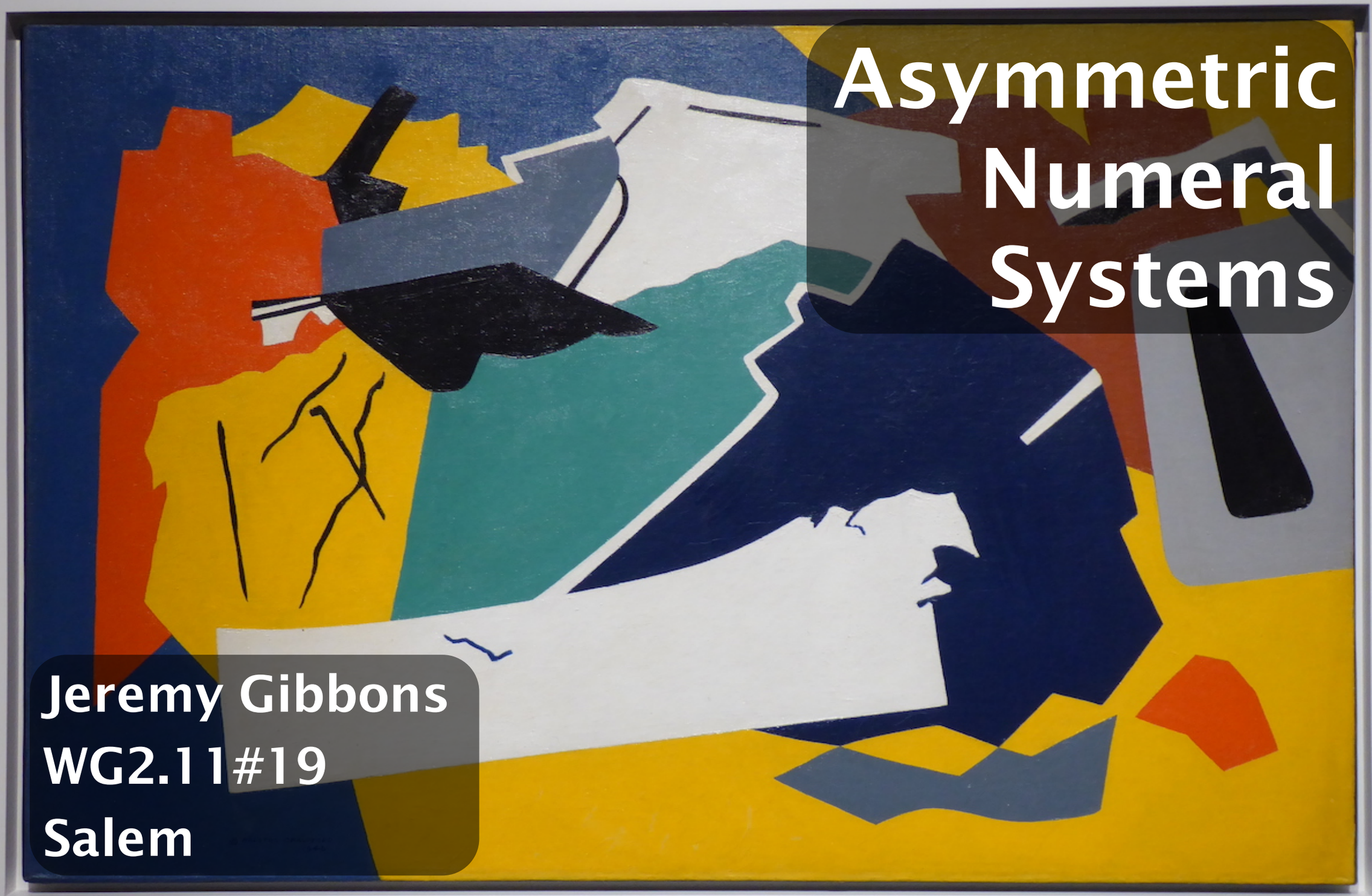
Asymmetric
Numeral
Systems
Jeremy Gibbons
WG2.11#19
Salem

1. Coding

- *Huffman coding* (HC)

  efficient; optimally effective for bit-sequence-per-symbol

- *arithmetic coding* (AC)

  Shannon-optimal (fractional entropy); but computationally expensive

- *asymmetric numeral systems* (ANS)

  efficiency of Huffman, effectiveness of arithmetic coding

- applications of *streaming* (another story)

ANS introduced by Jarek Duda (2006–2013).

Now: Facebook (Zstandard), Apple (LZFSE), Google (Draco), Dropbox (DivANS)...

2. Intervals

Pairs of rationals

$$\textbf{type}\ \mathit{Interval} = (\mathit{Rational}, \mathit{Rational})$$

with operations

$$\begin{aligned}
\mathit{unit} &= (0,1)\\
\mathit{weight}\ (l,r)\ x &= l + (r-l) \times x\\
\mathit{narrow}\ i\ (p,q) &= (\mathit{weight}\ i\ p, \mathit{weight}\ i\ q)\\
\mathit{scale}\ (l,r)\ x &= (x-l)\ /\ (r-l)\\
\mathit{widen}\ i\ (p,q) &= (\mathit{scale}\ i\ p, \mathit{scale}\ i\ q)
\end{aligned}$$

so that *narrow* and *unit* form a monoid, and inverse relationships:

$$\begin{aligned}
\mathit{weight}\ i\ x \in i &\iff x \in \mathit{unit}\\
\mathit{weight}\ i\ x = y &\iff \mathit{scale}\ i\ y = x\\
\mathit{narrow}\ i\ j = k &\iff \mathit{widen}\ i\ k = j
\end{aligned}$$

3. Models

Given

$$counts :: [\,(Symbol, Integer)\,]$$

get

$$encodeSym :: Symbol \to Interval$$
$$decodeSym :: Rational \to Symbol$$

such that

$$decodeSym\ x = s \iff x \in encodeSym\ s$$

Eg alphabet {'a', 'b', 'c'} with counts $2, 3, 5$ encoded as $(0, {}^1/_5)$, $({}^1/_5, {}^1/_2)$, and $({}^1/_2, 1)$.

4. Arithmetic coding

$$
\begin{aligned}
&\mathit{encode}_1 :: [\mathit{Symbol}] \to \mathit{Rational}\\
&\mathit{encode}_1 = \mathit{pick} \circ \mathit{foldl}\ \mathit{estep}_1\ \mathit{unit}\ \textbf{where}\\
&\quad \mathit{estep}_1 :: \mathit{Interval} \to \mathit{Symbol} \to \mathit{Interval}\\
&\quad \mathit{estep}_1\ i\ s = \mathit{narrow}\ i\ (\mathit{encodeSym}\ s)\\
&\mathit{decode}_1 :: \mathit{Rational} \to [\mathit{Symbol}]\\
&\mathit{decode}_1 = \mathit{unfoldr}\ \mathit{dstep}_1\ \textbf{where}\\
&\quad \mathit{dstep}_1 :: \mathit{Rational} \to \mathit{Maybe}\ (\mathit{Symbol}, \mathit{Rational})\\
&\quad \mathit{dstep}_1\ x = \textbf{let}\ s = \mathit{decodeSym}\ x\ \textbf{in}\ \mathit{Just}\ (s, \mathit{scale}\ (\mathit{encodeSym}\ s)\ x)
\end{aligned}
$$

where $\mathit{pick} :: \mathit{Interval} \to \mathit{Rational}$ satisfies $\mathit{pick}\ i \in i$. Eg, with $\mathit{pick} = \mathit{fst}$:

$$(0,1) \xrightarrow{\text{‘a’}} (0, {}^1/_5) \xrightarrow{\text{‘b’}} ({}^1/_{25}, {}^1/_{10}) \xrightarrow{\text{‘c’}} ({}^7/_{100}, {}^1/_{10}) \quad \rightsquigarrow \quad {}^7/_{100}$$

5. Trading in bits

Let $pick = fromBits \circ toBits :: Interval \to Rational$, where

$$\begin{aligned} toBits &:: Interval \to [Bool] \\ fromBits &:: [Bool] \to Rational \end{aligned}$$

Obvious thing: let *toBits i* pick shortest binary fraction in *i*, and *fromBits* evaluate this fraction. But this can't be streamed.

Instead, *toBits i* yields bit sequence *bs* such that $bs +\!\!+ [True]$ is shortest:

$$\begin{aligned} &toBits = unfoldr\ nextBit\ \textbf{where} \\ &\quad nextBit\ (l, r) \mid r \leqslant {}^1/_2 \quad = Just\ (False, (0, {}^1/_2)\ \underline{\text{widen}}\ (l, r)) \\ &\qquad\qquad\quad \mid {}^1/_2 \leqslant l \quad = Just\ (True, ({}^1/_2, 1)\ \underline{\text{widen}}\ (l, r)) \\ &\qquad\qquad\quad \mid otherwise = Nothing \\ &fromBits = foldr\ pack\ ({}^1/_2)\ \textbf{where}\ pack\ b\ x = ((\textbf{if}\ b\ \textbf{then}\ 1\ \textbf{else}\ 0) + x)\ /\ 2 \end{aligned}$$

Now *pick* is a hylomorphism. Also, *toBits* yields a finite sequence.

6. Streaming encoding

Move *fromBits* from encoding to decoding:

$$encode_{Bits} :: [Symbol] \to [Bool]$$
$$encode_{Bits} = toBits \circ foldl\ estep_1\ unit$$

$$decode_{Bits} :: [Bool] \to [Symbol]$$
$$decode_{Bits} = unfoldr\ dstep_1 \circ fromBits$$

Both of these can be *streamed*: alternately producing and consuming.

7. Towards ANS—fission and fusion

$$
\begin{aligned}
& encode_1 \\
= \quad & [\![\text{ definition; go back to } pick = fst \ ]\!] \\
& fst \circ foldl\ estep_1\ unit \\
= \quad & [\![\text{ map fusion for } foldl, \text{ backwards }]\!] \\
& fst \circ foldl\ narrow\ unit \circ map\ encodeSym \\
= \quad & [\![\ narrow \text{ is associative }]\!] \\
& fst \circ foldr\ narrow\ unit \circ map\ encodeSym \\
= \quad & [\![\text{ fusion for } foldr \ ]\!] \\
& foldr\ weight\ 0 \circ map\ encodeSym \\
= \quad & [\![\ map \text{ fusion; let } estep_2\ s\ x = weight\ (encodeSym\ s)\ x \ ]\!] \\
& foldr\ estep_2\ 0
\end{aligned}
$$

so let $encode_2 = foldr\ estep_2\ 0$.

8. Unfoldr–foldr theorem

Inverting a fold:

$$unfoldr\ g\ (foldr\ f\ e\ xs) = xs \quad \Longleftarrow g\ (f\ x\ z) = Just\ (x, z) \wedge g\ e = Nothing$$

Allowing *junk*:

$$(\exists ys\ .\ unfoldr\ g\ (foldr\ f\ e\ xs) = xs +\!\!+ ys) \Longleftarrow g\ (f\ x\ z) = Just\ (x, z)$$

With *invariant*:

$$\begin{aligned} unfoldr\ g\ (foldr\ f\ e\ xs) = xs \quad \Longleftarrow\ & ((g\ (f\ x\ z) = Just\ (x, z)) \Longleftarrow p\ z) \wedge \\ & ((g\ e = Nothing) \Longleftarrow p\ e) \end{aligned}$$

where invariant p of $foldr\ f\ e$ and $unfoldr\ g$ is such that

$$\begin{aligned} p\ (f\ x\ z) &\Longleftarrow p\ z \\ p\ z' &\Longleftarrow p\ z \wedge g\ z = Just\ (x, z') \end{aligned}$$

9. Correctness of decoding

$$
\begin{aligned}
& dstep_1\ (estep_2\ s\ z) \\
= \quad & [\![\ estep_2\ ]\!] \\
& dstep_1\ (weight\ (encodeSym\ s)\ z) \\
= \quad & [\![\ dstep_1; \text{let } s' = decodeSym\ (weight\ (encodeSym\ s)\ z)\ ]\!] \\
& Just\ (s', scale\ (encodeSym\ s')\ (weight\ (encodeSym\ s)\ z)) \\
= \quad & [\![\ s' = s \text{ (see next slide)}\ ]\!] \\
& Just\ (s, scale\ (encodeSym\ s)\ (weight\ (encodeSym\ s)\ z)) \\
= \quad & [\![\ scale\ i \circ weight\ i = id\ ]\!] \\
& Just\ (s, z)
\end{aligned}
$$

9. Correctness of decoding (continued)

Indeed, $s' = s$:

$$
\begin{aligned}
& \mathit{decodeSym}\ (\mathit{weight}\ (\mathit{encodeSym}\ s)\ z) = s \\
\iff & \quad [\![\ \text{central property}\ ]\!] \\
& \mathit{weight}\ (\mathit{encodeSym}\ s)\ z \in \mathit{encodeSym}\ s \\
\iff & \quad [\![\ \text{property of } \mathit{weight}\ ]\!] \\
& z \in \mathit{unit}
\end{aligned}
$$

and $z \in \mathit{unit}$ is an invariant. Therefore

$$\mathit{take}\ (\mathit{length}\ xs)\ (\mathit{decode}_1\ (\mathit{encode}_2\ xs)) = xs$$

for all finite xs.

10. From fractions to integers

AC encodes longer messages as more precise fractions.
In contrast, ANS makes larger integers.

$$
\begin{aligned}
&\mathit{count} :: \mathit{Symbol} \to \mathit{Integer} \\
&\mathit{cumul} :: \mathit{Symbol} \to \mathit{Integer} && \text{-- sum of } \mathit{count}\text{s of earlier symbols} \\
&\mathit{total} :: \mathit{Integer} && \text{-- sum of } \mathit{count}\text{s of all symbols} \\
&\mathit{find} :: \mathit{Integer} \to \mathit{Symbol}
\end{aligned}
$$

such that

$$\mathit{find}\ z = s \iff \mathit{cumul}\ s \leqslant z < \mathit{cumul}\ s + \mathit{count}\ s$$

for $0 \leqslant z < \mathit{total}$.

11. Asymmetric encoding: the idea

- text encoded as integer z, with $\log_2 z$ bits of information
- next symbol s has probability $p = \mathit{count}\ s\ /\ \mathit{total}$, so requires $\log_2 (1/p)$ bits
- so map z, s to $z' \simeq z \times \mathit{total}\ /\ \mathit{count}\ s$—but do so invertibly
- with $z' = (z\ \underline{\mathrm{div}}\ \mathit{count}\ s) \times \mathit{total}$, can undo the known multiplication:

$$z\ \underline{\mathrm{div}}\ \mathit{count}\ s = z'\ \underline{\mathrm{div}}\ \mathit{total}$$

- what about s? headroom! with $z' = (z\ \underline{\mathrm{div}}\ \mathit{count}\ s) \times \mathit{total} + \mathit{cumul}\ s$,

$$s = \mathit{find}\ (\mathit{cumul}\ s) = \mathit{find}\ (z'\ \underline{\mathrm{mod}}\ \mathit{total})$$

- what about z? with $z' = (z\ \underline{\mathrm{div}}\ \mathit{count}\ s) \times \mathit{total} + \mathit{cumul}\ s + z\ \underline{\mathrm{mod}}\ \mathit{count}\ s$,

$$z\ \underline{\mathrm{mod}}\ \mathit{count}\ s = z'\ \underline{\mathrm{mod}}\ \mathit{total} - \mathit{cumul}\ s$$

12. ANS, pictorially

The start of the coding table for alphabet `'a'`, `'b'`, `'c'` with counts $2, 3, 5$:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	⋯
`'a'`	•	•									•	•						
`'b'`			•	•	•								•	•	•			
`'c'`						•	•	•	•	•						•	•	

- integers $0..$ distributed across the alphabet, in proportion to counts
- encoding symbol s with current accumulator x yields the position of the xth blob in row s as the new accumulator x'

13. Essence of ANS encoding and decoding

$$
\begin{aligned}
&encode_3 :: [Symbol] \to Integer \\
&encode_3 = foldr\ estep_3\ 0 \\
&estep_3 :: Symbol \to Integer \to Integer \\
&estep_3\ s\ z = \mathbf{let}\ (q, r) = z\ \underline{\mathrm{divMod}}\ count\ s\ \mathbf{in}\ q \times total + cumul\ s + r \\
&decode_3 :: Integer \to [Symbol] \\
&decode_3 = unfoldr\ dstep_3 \\
&dstep_3 :: Integer \to Maybe\ (Symbol, Integer) \\
&dstep_3\ z = \mathbf{let}\ (q, r) = z\ \underline{\mathrm{divMod}}\ total \\
&\qquad\qquad\quad s = find\ r \\
&\qquad\qquad \mathbf{in}\ Just\ (s, count\ s \times q + r - cumul\ s)
\end{aligned}
$$

Correctness argument as before. But note that encoding is right-to-left.

14. Variation

Correctness does not depend on starting value: can pick any l instead of 0.

Also, $estep_3$ strictly increasing on $z > 0$, and $dstep_3$ strictly decreasing, so we know when to stop:

$$encode_4 :: [Symbol] \to Integer$$
$$encode_4 = foldr\ estep_3\ l$$

$$decode_4 :: Integer \to [Symbol]$$
$$decode_4 = unfoldr\ dstep_4$$

$$dstep_4 :: Integer \to Maybe\ (Symbol, Integer)$$
$$dstep_4\ z = \textbf{if}\ z == l\ \textbf{then}\ Nothing\ \textbf{else}\ dstep_3\ z$$

and we have

$$decode_4\ (encode_4\ xs) = xs$$

for all finite xs, but this time without junk.

15. Bounded precision

Fix base b and lower bound l. Represent accumulator z as pair (x, ys) such that:

- *remainder* ys is a list of digits in base b
- *window* x satisfies $l \leqslant x < u$ for upper bound $u = l \times b$

under abstraction $z = foldl\ inject\ x\ ys$ where

$$inject\ x\ y = x \times b + y \qquad \text{and} \qquad extract\ x = x\ \underline{\text{divMod}}\ b$$

Eg with $b = 10$ and $l = 100$, pair $(123, [4, 5, 6])$ represents 123456.

$$\textbf{type}\ Number = (Integer, [Integer])$$

Note "you can't miss it" properties:

$$inject\ x\ y < u \iff x < l$$
$$l \leqslant fst\ (extract\ x) \iff u \leqslant x$$

Want b, l powers of two, and u a single word. Also nice if $l\ \underline{\text{mod}}\ total = 0$.

16. Encoding

Maintain window in range.

$$
\begin{aligned}
&econsume_5 :: [Symbol] \to Number \\
&econsume_5 = foldr\ estep_5\ (l, [\,])
\end{aligned}
$$

$$
\begin{aligned}
&estep_5 :: Symbol \to Number \to Number \\
&estep_5\ s\ (x, ys) = \mathbf{let}\ (x', ys') = enorm_5\ s\ (x, ys)\ \mathbf{in}\ (estep_3\ s\ x', ys')
\end{aligned}
$$

$$
\begin{aligned}
&enorm_5 :: Symbol \to Number \to Number \\
&enorm_5\ s\ (x, ys) = \mathbf{if}\quad estep_3\ s\ x < u \\
&\qquad\qquad\quad\ \mathbf{then}\ (x, ys) \\
&\qquad\qquad\quad\ \mathbf{else}\ \ \mathbf{let}\ (q, r) = extract\ x\ \mathbf{in}\ enorm_5\ s\ (q, r : ys)
\end{aligned}
$$

Pre-normalize *before* consuming symbol. Eg with $b = 10, l = 100$:

$$(340, [3]) \xleftarrow{\text{‘a’}} (68, [3]) \xleftarrow{\text{norm}} (683, [\,]) \xleftarrow{\text{‘b’}} (205, [\,]) \xleftarrow{\text{‘c’}} (100, [\,])$$

17. Decoding

$$
\begin{aligned}
&dproduce_5 :: Number \to [\,Symbol\,]\\
&dproduce_5 = unfoldr\ dstep_5\\
&dstep_5 :: Number \to Maybe\ (Symbol, Number)\\
&dstep_5\ (x, ys) = \ \mathbf{let}\ Just\ (s, x') = dstep_3\ x\\
&\qquad\qquad\qquad\qquad (x'', ys'') = dnorm_5\ (x', ys)\\
&\qquad\qquad\qquad \mathbf{in\ if}\ x'' \geqslant l\ \mathbf{then}\ Just\ (s, (x'', ys''))\ \mathbf{else}\ Nothing\\
&dnorm_5 :: Number \to Number \quad \text{-- } dnorm_5\ (enorm_5\ s\ (x, ys)) = (x, ys) \text{ when } l \leqslant x < u\\
&dnorm_5\ (x, y : ys) = \mathbf{if}\ x < l\ \mathbf{then}\ dnorm_5\ (inject\ x\ y, ys)\ \mathbf{else}\ (x, y : ys)\\
&dnorm_5\ (x, [\,]) \quad\ = (x, [\,])
\end{aligned}
$$

Decoding is symmetric to encoding: renormalize *after* emitting a symbol.

$$(340, [\,3\,]) \xrightarrow{\text{‘a’}} (68, [\,3\,]) \xrightarrow{\text{norm}} (683, [\,]) \xrightarrow{\text{‘b’}} (205, [\,]) \xrightarrow{\text{‘c’}} (100, [\,])$$

Correctness again as before (no junk; invariant $l \leqslant x < u$).

18. Trading in sequences

Flush out the last few digits in the *Number* when encoding complete:

$$
\begin{aligned}
&eflush_5 :: Number \to [\,Integer\,] \\
&eflush_5\ (0, ys) = ys \\
&eflush_5\ (x, ys) = \mathbf{let}\ (x', y) = extract\ x\ \mathbf{in}\ eflush_5\ (x', y : ys)
\end{aligned}
$$

$$
\begin{aligned}
&encode_5 :: [\,Symbol\,] \to [\,Integer\,] \\
&encode_5 = eflush_5 \circ econsume_5
\end{aligned}
$$

Conversely, populate initial *Number* from first few digits when decoding:

$$
\begin{aligned}
&dstart_5 :: [\,Integer\,] \to Number \\
&dstart_5\ ys = dnorm_5\ (0, ys)
\end{aligned}
$$

$$
\begin{aligned}
&decode_5 :: [\,Integer\,] \to [\,Symbol\,] \\
&decode_5 = dproduce_5 \circ dstart_5
\end{aligned}
$$

Note that $dstart_5\ (eflush_5\ x) = x \iff l \leqslant x < u$.

19. Fast loops

```
encode :: [Symbol] → [Integer]   -- one tight loop
encode = h1 l ∘ reverse where
  h1 x (s : ss) = let x' = estep3 s x in if x' < u then h1 x' ss else
                  let (q, r) = extract x in r : h1 q (s : ss)
  h1 x [ ]      = h2 x
  h2 x = if x == 0 then [ ] else let (x', y) = extract x in y : h2 x'

decode :: [Integer] → [Symbol]   -- one tight loop
decode = h0 0 ∘ reverse where
  h0 x (y : ys)   | x < l = h0 (inject x y) ys
  h0 x ys                 = h1 x ys
  h1 x ys                 = let Just (s, x') = dstep3 x in h2 s x' ys
  h2 s x (y : ys) | x < l = h2 s (inject x y) ys
  h2 s x ys               = if x ⩾ l then s : h1 x ys else [ ]
```

20. But...

- AC encoding and decoding are both left-to-right
- so AC can be made *adaptive*: adjust model with each symbol
- fixed-precision AC is (I believe) equivalent to a quantizing adaptation

- ANS encoding and decoding go in *opposite* directions
- can't be so easily made adaptive
- then fixed-precision ANS is *a different problem*

Commments welcome! Paper in preparation...