Reasoning about Codata — Practicals

The exercises are a mixture of practical exercises (keyboard and computer) and theoretical exercises (pencil and paper). There are more exercises than you can possibly work on within the 45 min time frame. So pick the ones you find most interesting or challenging.

We will be using GHCi for the practical exercises. To run GHCi, simply open a terminal window and type '`ghci`'. One typically uses a text editor to write or edit a Haskell script, saves that to disk, and loads it into GHCi. To load a script, it is helpful if you run GHCi from the directory containing the script. You can simply give the name of the script file as a parameter to the command `ghci`. Or, within GHCi, you can type '`:l`' followed by the name of the script to load, and '`:r`' with no parameter to reload the file previously loaded.

The necessary definitions for the exercises on streams are contained in the script `Stream.lhs`; for the ones on infinite trees load `Tree.lhs`.

1 Streams

1. Try to capture the sequences below using stream equations.

$$\begin{array}{l}
\langle 0, 1, 8, 27, 64, 125, 216, 343, 512, 729, 1000, 1331, \ldots\rangle \\
\langle 1, 3, 9, 27, 81, 243, 729, 2187, 6561, 19683, 59049, \ldots\rangle \\
\langle 0, 0, 1, 1, 2, 4, 3, 9, 4, 16, 5, 25, 6, 36, 7, 49, \ldots\rangle \\
\langle 0, 0, 2, 4, 8, 14, 24, 40, 66, 108, 176, 286, 464, 752, \ldots\rangle \\
\langle 0, 1, 2, 6, 15, 40, 104, 273, 714, 1870, 4895, 12816, \ldots\rangle
\end{array}$$

   Hint: For the latter two puzzles experiment a little with the Fibonacci sequences *fib* and *fib'*. *Hint:* Sloane's On-Line Encyclopedia of Integer Sequences, `http://www.research.att.com/~njas/sequences/`, lists most integer sequences one can think of.

2. Turn the following verbal descriptions into streams.
   (a) The sequence of natural numbers divisible by 3.
   (b) The sequence of natural numbers *not* divisible by 3.
   (c) The sequence of cubes.
   (d) The sequence of all finite binary strings.
   (e) The bit-reversed positive numbers: the order of all bits, except the most significant one, in the binary expansion of n is reversed.

3. The parametric stream *from* is given by

$$\begin{array}{lll}
\mathit{from} & :: & \mathit{Nat} \to \mathit{Stream}\ \mathit{Nat} \\
\mathit{from}\ n & = & n \prec \mathit{from}\ (n+1)\ .
\end{array}$$

   Show that $\mathit{from}\ n + \mathit{pure}\ k = \mathit{from}\ (n + k)$ in at least two different ways.

4. Show $s + 1 = 1 + s$ using solely the idiom laws listed below.

$$\begin{array}{llll}
\mathit{pure}\ \mathit{id} \diamond u & = & u & \text{(identity)} \\
\mathit{pure}\ (\cdot) \diamond u \diamond v \diamond w & = & u \diamond (v \diamond w) & \text{(composition)} \\
\mathit{pure}\ f \diamond \mathit{pure}\ x & = & \mathit{pure}\ (f\ x) & \text{(homomorphism)} \\
u \diamond \mathit{pure}\ x & = & \mathit{pure}\ (\lambda f \to f\ x) \diamond u & \text{(interchange)}
\end{array}$$

   As an aside, can you make sense of the names?

5. Prove the idiom laws using the unique fixed point principle.

6. When are *iterate* $f\ a$ and *iterate* $g\ b$ equal? As a simple example, consider *iterate* $([\texttt{"hi"}]+\!\!+)\ [\,]$ and *iterate* $(+\!\!+[\texttt{"hi"}])\ [\,]$. Can you find sufficient and necessary conditions?

2 Recurrences

1. The naîve implementation of the Fibonacci numbers is horribly inefficient.

   $$\begin{aligned} \mathit{fib}\,0 &= 0 \\ \mathit{fib}\,1 &= 1 \\ \mathit{fib}\,(n+2) &= \mathit{fib}\,n + \mathit{fib}\,(n+1) \end{aligned}$$

   But, can we make this more precise? For instance, how many additions are performed in order to compute $\mathit{fib}\ n$, or, how many recursive calls are made? Express your findings as stream equations. Then try to relate the two streams to examples we covered in the lectures. Can you generalise the results?

2. Determine the number of binary strings of some given length that do not contain adjacent zeros. Again, first try to come up with a system of recursion equations and then try to relate the streams to known examples. This puzzle calls for generalisation, as well.

3. Turn the Fibonacci sequence

   $$\mathit{fib} = 0 \prec \mathit{fib} + (1 \prec \mathit{fib})$$

   into an iterative form: $\mathit{map}\ g\ (\mathit{iterate}\ f\ a)$. There are, at least, two approaches:

   - Pair *fib* and *fib'*

     $$\mathit{fib} \star \mathit{fib}'\ ,$$

     where $(\star) = \mathit{zip}\ (,)$ turns a pair of streams into a stream of pairs. (The quizzical '$(,)$' is Haskell's pairing constructor.)
   - Use the fact that the tails of *fib* are linear combinations of *fib* and *fib'*.

     $$i * \mathit{fib} + j * \mathit{fib}'$$

   Try to relate the two approaches.

4. Turn the equation

$$x = (a \prec map\ f\ x) + s$$

into an iterative form. *Hint:* You may find the function $tails = iterate\ tail$ useful. As an aside, *tails* is the co-multiplication of the product co-monad *Stream*.

5. Prove that $tabulate\ f = map\ f\ nat$.

6. Show that the sequence given by $a_0 = k$, $a_{2n+1} = f(a_n)$ and $a_{2n+2} = g(a_n)$ corresponds to the stream $a = k \prec map\ f\ a \curlyvee map\ g\ a$. *Hint:* use $nat = bin$ and the previous exercise.

3 Finite Calculus

1. The product rule $\Delta\ (s * t) = s * \Delta\ t + \Delta\ s * \mathit{tail}\ t$ is somewhat asymmetric. Can you find a symmetric variant? Prove it correct.

2. Derive the sum rule $\Sigma\ (s + t) = \Sigma\ s + \Sigma\ t$ from the sum rule $\Delta\ (s + t) = \Delta\ s + \Delta\ t$ using the Fundamental Theorem.

$$t = \Delta\ s \iff \Sigma\ t = s - \mathit{repeat}\ (\mathit{head}\ s)$$

3. Work out $\Sigma\ \mathit{nat}^3$ using the summation laws and the correspondence between powers and falling factorial powers.

4. Here is an alternative definition of Σ

$$\Sigma\ s = 0 \prec \mathit{repeat}\ (\mathit{head}\ s) + \Sigma\ (\mathit{tail}\ s)\ ,$$

   which uses a second-order fixed point. The code implements the naïve way of summing: the ith element is computed using i additions not reusing any previous results. Prove that the two definitions of Σ are equivalent.

5. Generalise the derivation of $\Sigma\ (\mathit{nat} * 2^{\mathit{nat}})$ to $\Sigma\ (\mathit{nat} * c^{\mathit{nat}})$, where c is a constant stream.

$$\begin{aligned}
& \Sigma\ (\mathit{nat} * 2^{\mathit{nat}}) \\
= & \quad \{\ \Delta\ 2^{\mathit{nat}} = 2^{\mathit{nat}}\ \} \\
& \Sigma\ (\mathit{nat} * \Delta\ 2^{\mathit{nat}}) \\
= & \quad \{\ \text{summation by parts}\ \} \\
& \mathit{nat} * 2^{\mathit{nat}} - \Sigma\ (\Delta\ \mathit{nat} * \mathit{tail}\ 2^{\mathit{nat}}) \\
= & \quad \{\ \Delta\ \mathit{nat} = 1, \text{ and definition of } \mathit{nat}\ \} \\
& \mathit{nat} * 2^{\mathit{nat}} - 2 * \Sigma\ 2^{\mathit{nat}} \\
= & \quad \{\ \text{summation law}\ \} \\
& \mathit{nat} * 2^{\mathit{nat}} - 2 * (2^{\mathit{nat}} - 1) \\
= & \quad \{\ \text{arithmetic}\ \} \\
& (\mathit{nat} - 2) * 2^{\mathit{nat}} + 2
\end{aligned}$$

6. Find a closed formula for $\Sigma\ \mathit{fib}^2$. *Hint:* The tail of the sequence is called the sequence of *golden rectangle numbers*.

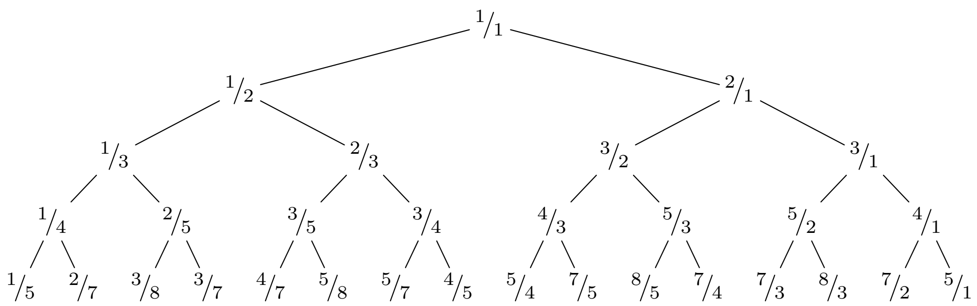

Figure 1: Stern-Brocot tree

4 Infinite trees

The final set of exercises is organised around a common theme: enumerating the positive rationals. Because of that, most of the exercises are inter-dependent. If you get stuck, feel free to continue to work on the previous sets of exercises.

There are many ways to enumerate the positive rationals. Probably the oldest method was discovered in the 1850s by the German mathematician Stern and independently a few years later by the French clockmaker Brocot. It's deceptively simple: Start with the two 'boundary rationals' ${}^{0}/_{1}$ and ${}^{1}/_{0}$, which are not included in the enumeration, and then repeatedly insert the *mediant* ${}^{a+b}/_{c+d}$ between two adjacent rationals ${}^{a}/_{c}$ and ${}^{b}/_{d}$.

Since the number of inserted rationals doubles with every step, the process can be pictured by an infinite binary tree, the so-called Stern-Brocot tree, see Figure 1.

1. *Turn the informal description into a program.*

   If we represent an inserted rational ${}^{a+b}/_{c+d}$ by the matrix $\begin{pmatrix} a & b \\ c & d \end{pmatrix}$, then its left and right descendant can be determined as follows.

   $$\begin{pmatrix} a & a+b \\ c & c+d \end{pmatrix} \mapsfrom \begin{pmatrix} a & b \\ c & d \end{pmatrix} \mapsto \begin{pmatrix} a+b & b \\ c+d & d \end{pmatrix}$$

   Phrase the transformations as matrix multiplications and then define the Stern-Brocot tree as an *unfold*, a *map* after an *iterate*.

2. *Turn the iterative form into a recursive form.*

Show that the iterative formulation is equivalent to the following recursive definition.

$$
\begin{aligned}
&\mathit{stern} :: \mathit{Tree\ Rational} \\
&\mathit{stern} = \mathit{Node}\ 1\ (1\ /\ (1\ /\ \mathit{stern} + 1))\ (\mathit{stern} + 1)
\end{aligned}
$$

The definition makes explicit that the right subtree is the 'successor' of the entire tree, see Figure 1.

3. *Relate the Stern-Brocot tree to Dijkstra's fusc sequence.*

   In EWD570, Dijkstra introduced the following function, also known as Stern's diatomic sequence,

$$
\begin{aligned}
S_1 &= 1 \\
S_{2*n} &= S_n \\
S_{2*n+1} &= S_n + S_{n+1} \quad ,
\end{aligned}
$$

   which is a strange variant of *fib*.

   Tabulate the function: *fusc* = *tabulate* S. *Hint:* You may find it helpful to use the function *chop* that serves as the counterpart of *tail*.

$$
\begin{aligned}
&\mathit{chop} \quad :: \mathit{Tree}\ \alpha \to \mathit{Tree}\ \alpha \\
&\mathit{chop}\ t = \mathit{Node}\ (\mathit{root}\ (\mathit{left}\ t))\ (\mathit{right}\ t)\ (\mathit{chop}\ (\mathit{left}\ t))
\end{aligned}
$$

   Show that *stern* = *fusc* ÷ *fusc'*, where ÷ constructs a rational from two integers and *fusc'* = *chop fusc*.

4. *Turn the recursive form of fusc into an iterative one.*

   Turn the trees

$$
\begin{aligned}
\mathit{num} &= \mathit{Node}\ 1\ \mathit{num} \qquad\qquad (\mathit{num} + \mathit{den}) \\
\mathit{den} &= \mathit{Node}\ 1\ (\mathit{num} + \mathit{den})\ \mathit{den}
\end{aligned}
$$

   into an iterative form (*num* and *den* are more telling names for *fusc* and *fusc'*).

   There are, at least, two approaches:

- Pair *num* and *den*

  $$num \star den \ ,$$

  where $(\star) = zip\ (,)$ turns a pair of trees into a tree of pairs. (The quizzical '$(,)$' is Haskell's pairing constructor.)

- Use the fact that the subtrees of *num* are linear combinations of *num* and *den*.

  $$i * num + j * den$$

  (In EWD578, Dijkstra uses a similar approach to prove that $fusc + fusc' = mirror\ (fusc + fusc')$, where *mirror* swaps the immediate subtrees of a node and all its descendants.)

Try to relate the two approaches.

5. *Show that the rationals are in their lowest common form.*

   In Exercise 3 we have shown that $stern = num \div den$. This fact does *not*, however, imply that $map\ numerator\ stern = num$ and $map\ denominator\ stern = den$. (Why?) In order to prove the latter two equations, we have to show that $num \div den$ are in their lowest common form, that is, the greatest common divisor of *num* and *den* is 1:

   $$num \bigtriangledown den = 1 \ ,$$

   where $\bigtriangledown$ denotes the greatest common divisor lifted to trees.

6. *Show that the Stern-Brocot tree contains every rational at most once.*

   There are, at least, two approaches. One can show that *stern* is a search-tree using the following fact about mediants: if ${}^{a}/_{c} \leqslant {}^{b}/_{d}$, then

   $${}^{a}/_{c} \leqslant {}^{a+b}/_{c+d} \leqslant {}^{b}/_{d} \ .$$

   Alternatively, one can show that *lookup stern* is injective by demonstrating that it has a left-inverse. Now, rational numbers are in a one-to-one correspondence to bit strings. The following instrumented version of the greatest common divisor

$$\begin{aligned} a \blacktriangledown b = \ & \textbf{case}\ \mathit{compare}\ a\ b\ \textbf{of} \\ & LT \to 0 : (a \blacktriangledown (b - a)) \\ & EQ \to [\,] \\ & GT \to 1 : ((a - b) \blacktriangledown b) \ , \end{aligned}$$

maps two positive numbers to a bit string. This defines the required left-inverse. Then

$$\mathit{num} \blacktriangledown \mathit{den} = \mathit{tabulate}\ \mathit{id}$$

establishes the result. (Why?)

7. *Show that the Stern-Brocot tree contains every rational at least once.*

   Show that *lookup stern* is surjective by demonstrating that it has a right-inverse.

8. *Linearise the Stern-Brocot tree.*

   Turn *stream stern* into an iterative form, where the natural transformation *stream*

$$\begin{aligned} & \mathit{stream} \quad :: \mathit{Tree}\ \alpha \to \mathit{Stream}\ \alpha \\ & \mathit{stream}\ t = \mathit{root}\ t \prec \mathit{stream}\ (\mathit{chop}\ t) \end{aligned}$$

   converts an infinite tree to a stream. In other words, enumerate the rationals!

   (a) As a first step, linearise *den*. You have to express *chop den* in terms of *den* and possibly *num*. Show that $\mathit{chop}\ \mathit{den} = \mathit{num} + \mathit{den} - 2 * x$ where x is the unique solution of $x = \mathit{Node}\ 0\ \mathit{num}\ x$.

   (b) Show that the unique solution of $x = \mathit{Node}\ 0\ \mathit{num}\ x$ equals $\mathit{num}\ \textbf{mod}\ \mathit{den}$.

   (c) Using the results of the two previous items, linearise *num* and *den*: $\mathit{sfusc} = \mathit{stream}\ \mathit{num}$ and $\mathit{sfusc}' = \mathit{stream}\ \mathit{den}$.

   (d) Turn $\mathit{sfusc} \star \mathit{sfusc}'$ into an iterative form.

(e) *Polishing up:* Use the formula

$$1 \,/\, (\lfloor n \div d \rfloor + 1 - \{n \div d\}) = d \div (n + d - 2 * (n \mathbin{\mathbf{mod}} d))$$

where $\lfloor r \rfloor$ denotes the integral part of r and $\{r\}$ its fractional part ($r = \lfloor r \rfloor + \{r\}$), to turn the result of the previous item into the following amazingly short program for enumerating the rationals.

$$\begin{aligned}&\mathit{rationals} = \mathit{iterate}\ \mathit{next}\ 1\\&\quad \mathbf{where}\ \mathit{next}\ r = 1 \,/\, (\lfloor r \rfloor + 1 - \{r\})\end{aligned}$$

Ralf Hinze (May 2009)