Enumerating the Elements of the Eisenstein Array

Roland Backhouse rcb@cs.nott.ac.uk

João F. Ferreira joao@joaoff.com

March 24, 2009

Abstract

In 1858 [5], A.M. Stern published a detailed study of a process of constructing an infinite sequence of numbers from a given pair of numbers. Stern attributed the process to Eisenstein, and the sequence of numbers is now known as the Eisenstein array.

In [2], we review Stern's paper and briefly discuss algorithms that enumerate the elements of the Eisenstein Array. In this document we present several Haskell implementations of these algorithms.

1 The Eisenstein Array

Stern [5] describes a process (which he attributes to Eisenstein) of generating an infinite sequence of rows of numbers from a pair of natural numbers m and n. The *zeroth* row in the sequence ("nullte Entwickelungsreihe") is the given pair of numbers:

$$m \quad n \quad .$$

Subsequent rows are obtained by inserting between every pair of numbers the sum of the numbers. Thus the *first* row is

$$m \quad m+n \quad n$$

and the *second* row is

$$m \quad 2\times m+n \quad m+n \quad m+2\times n \quad n \quad .$$

The process of constructing such rows is repeated indefinitely. The sequence of numbers obtained by concatenating the individual rows in order is what is now called the *Eisenstein array* and denoted by $Ei(m,n)$ (see, for example, [4, sequence A064881]) . Stern refers to each occurrence of a number in rows other than the zeroth row as either a *sum element* ("Summenglied") or a *source element* ("Stammglied"). The sum elements are the newly added numbers. For example, in the first row the number $m+n$ is a sum element; in the second row the number $m+n$ is a source element.

2 Newman's Algorithm

In his paper, Stern observes two ways in which the Eisenstein array $Ei(1,1)$ (now known as Stern's diatomic series) enumerates the set of positive rationals. The simplest of the two, and the first in Stern's paper, is the sequence obtained by considering consecutive pairs of elements ("zweigliedrige Gruppen") in each row of the array $Ei(1,1)$. (The second is what has now become known as the Stern-Brocot tree of rationals.) Stern proved that the set of such pairs of numbers is exactly the set of pairs of coprime positive numbers. In other words, the set of all consecutive pairs of numbers in rows of $Ei(1,1)$ is the set of rational numbers in lowest form.

Renewed interest in Stern's work has been sparked by Calkin and Wilf [3]. They give a recurrence relation for the kth element in $Ei(1,1)$, and prove that it equals the number of hyperbinary representations of k.

Moshe Newman is credited with an iterative algorithm for enumerating the rationals. Just as Stern's analysis of the sequence of rows of $Ei(1,1)$ embodies an algorithm for enumerating the rationals, Newman's algorithm for enumerating the rationals embodies an algorithm for enumerating the elements of $Ei(1,1)$ and, indeed, of $Ei(m,n)$ for arbitrary positive numbers m and n. The purpose of the current document is to discuss several Haskell implementations of Newman's algorithm adapted to this purpose.

Newman's algorithm, in the form derived in [1], is implemented in Haskell as follows.

```
cwnEnum :: [Rational]
cwnEnum = iterate nextCW 1/1
  where nextCW :: Rational → Rational
        nextCW r = let (a, b) = (numerator r, denominator r)
                       j      = ⌊a/b⌋
                   in  b/((2×j + 1)×b − a)
```

Since each rational in the sequence $cwnEnum$ is a pair of consecutive numbers in a row of $Ei(1,1)$, Newman's algorithm predicts that each triple of numbers in a given row of $Ei(1,1)$ has the form

$$a \qquad b \qquad (2\left\lfloor \frac{a}{b} \right\rfloor + 1)\times b - a \quad .$$

(See [1, appendix A] for further discussion on the extent to which Stern anticipated this algorithm.)

In order to adapt Newman's algorithm to enumerate $Ei(1,1)$, it suffices to determine when one row is complete and the next begins. This is easy since each row begins and ends with the number 1. Function $newman$ below combines this fact with $cwnEnum$

in order to enumerate the elements of $Ei(1,1)$; specifically, the row is completed when the denominator of the rational is 1.

$$
\begin{aligned}
&newman :: [Integer] \\
&newman = concatMap\ dlevel\ cwnEnum \\
&\quad \textbf{where}\ dlevel\ r \mid (denominator\ r) == 1 = [numerator\ r, 1] \\
&\qquad\qquad\quad \mid otherwise\qquad\qquad\ \ = [numerator\ r]
\end{aligned}
$$

3 Enumerating the Elements of $Ei(m,n)$

In order to enumerate the elements of the array $Ei(m,n)$, the main problem we have to solve is the detection of a change of level (i.e. when one row is complete and the next begins). There are several ways to do this. One is to simultaneously enumerate $Ei(1,1)$ and use this to control the detection process.

In the following code, the parameters a and b in *eiloop* sequence through the elements of a row of $Ei(1,1)$. The end of the row is detected when $b = 1$. (The second clause is executed only when $b \neq 1$.)

$$
\begin{aligned}
&ei11 :: [Integer] \\
&ei11 = 1 : eiloop\ 1\ 1 \\
&\quad \textbf{where}\ eiloop\ a\ 1 = 1 : 1 : eiloop\ 1\ (a+1) \\
&\qquad\quad eiloop\ a\ b = \textbf{let}\ k = 2 \times \lfloor a/b \rfloor + 1 \\
&\qquad\qquad\qquad\qquad\ \ \textbf{in}\ \ b : eiloop\ b\ (k \times b - a)
\end{aligned}
$$

Generalising to $Ei(m,n)$, we get the following. (Note that the parameters cm and cn of *eiloop* play the same role as global constants in an imperative program; they record the initial values of m and n.) The property exploited by the algorithm is that each element of $Ei(m,n)$ is a linear combination of m and n of which the coefficients are independent of m and n.

$$
\begin{aligned}
&ei :: Integer \to Integer \to [Integer] \\
&ei\ m\ n = m : eiloop\ 1\ 1\ m\ n\ m\ n \\
&\quad \textbf{where}\ eiloop\ a\ 1\ m\ n\ cm\ cn = n : cm : eiloop\ 1\ (a+1)\ cm\ (a \times cm + cn)\ cm\ cn \\
&\qquad\quad eiloop\ a\ b\ m\ n\ cm\ cn = \textbf{let}\ k = 2 \times \lfloor a/b \rfloor + 1 \\
&\qquad\qquad\qquad\qquad\qquad\qquad\ \ \textbf{in}\ \ n : eiloop\ b\ (k \times b - a)\ n\ (k \times n - m)\ cm\ cn
\end{aligned}
$$

We can test if the function *newman* does in fact enumerate the elements of $Ei(1,1)$. Let's compare the first 1000 elements of both enumerations:

```
≫ (take 1000 newman) == (take 1000 (ei 1 1))
True
```

$\gg$ (take 1000 (map (2*) newman)) == (take 1000 (ei 2 2))
True

The part after the prompt, $\gg$, is the Haskell code that `ghci` executes. The result is shown in the subsequent line. The second command shows an instance of the property:

$$map\ (k\times)\ (ei\ 1\ 1) == ei\ k\ k \quad .$$

A simplification of ei is obtained by replacing a and b by m and n in the calculation of k (when n is strictly positive). (The justification for this simplification involves induction on rows. The induction hypothesis is that if (a, b, c) is a triple (a "driegliedrige Gruppe" in Stern's terminology) in a given row of $Ei(1,1)$ and (i, j, k) is a triple in the same position in the same row of $Ei(m,n)$, then $\lfloor a/b \rfloor = \lfloor i/j \rfloor$. (The role of variable c in this hypothesis is simply to exclude the last two elements in a row. Indeed, the equality does not hold for these two elements. Symmetrically, of course, $\lfloor b/c \rfloor = \lfloor j/k \rfloor$, provided m is strictly positive.)

```
ei' :: Integer → Integer → [Integer]
ei' m n
  | n > 0       = m : eiloop 1 1 m n m n
  | otherwise = error "The second argument has to be strictly positive."
  where eiloop a 1 m n cm cn = n : cm : eiloop 1 (a + 1) cm (a×cm + cn) cm cn
        eiloop a b m n cm cn = let k = 2×⌊m/n⌋ + 1
                               in  n : eiloop b (k×b − a) n (k×n − m) cm cn
```

We now define the function *test*, which compares the first 1000 elements of two enumerations of $Ei(m,n)$, with $0 \leqslant m \leqslant x$ and $1 \leqslant n \leqslant x$:

```
test f g x = and [take 1000 (f m n) == take 1000 (g m n) | m ← [0..x], n ← [1..x]]
```

We can use *test* to see if the first 1000 elements of $ei\ m\ n$ and $ei'\ m\ n$, for $0 \leqslant m \leqslant 100$ and $1 \leqslant n \leqslant 100$, are the same (we are testing 10100 pairs).

$\gg$ test ei ei' 100
True

Building on ei', a further transformation is to replace the test for the end of a row in the sequence. This has been done in the function *extnewman*, defined below. It is the same as *ei*, but it replaces variables a and b by variable r which counts the rows. The last two elements in row r of $Ei(cm,cn)$ are $cm + r\times cn$ and cn, and the first two elements in row $r + 1$ are cm and $(r + 1)\times cm + cn$.

```
extnewman :: Integer → Integer → [Integer]
extnewman cm cn
  | cn > 0 =      cm : loop 0 cm cn cm cn
  | otherwise = error "The second argument has to be strictly positive."
  where loop r m n cm cn | ((m == (cm + r×cn)) ∧ (n == cn)) =
                             n : cm : loop (r + 1) cm ((r + 1)×cm + cn) cm cn
                         | otherwise = let k = 2×⌊m/n⌋ + 1
                                       in  n : loop r n (k×n − m) cm cn
```

We can do a similar test for *extnewman* as we did for *ei'*:

```
≫ test ei extnewman 100
True
```

4 The Online Encyclopedia of Integer Sequences

In this section, we show how we can use the functions from the Haskell module *Math.OEIS*[1] to search for occurrences of the Eisenstein array in the Online Encyclopedia of Integer Sequences (OEIS) [4].

We start by defining the number of elements, *numElems*, that we want to send to the OEIS, and a function that converts a list $[x_1, \cdots, x_n]$ to the string $"x_1, \cdots, x_n"$:

```
numElems :: Int
numElems = 20


list2string :: (Show a) ⇒ [a] → String
list2string = init ∘ tail ∘ show
```

The following function, *oeis*, inputs two integer numbers, m and n, computes the list of the first *numElems* of *ei m n*, transforms the list into a string and checks if it exists in the OEIS. It prints the description of the sequence, together with its reference.

```
oeis      :: Integer → Integer → IO ()
oeis m n = do s ← searchSequence_IO ∘ list2string ∘ (take numElems) ei m n
              r ← getDataSeq s
              putStrLn "Ei(" ++ show m ++ "," ++ show n ++ "):\n\t" ++ r
  where getDataSeq            :: Maybe OEISSequence → IO String
        getDataSeq Nothing    = return "Sequence not found."
```

[1] To run this literate Haskell file, you need to have the module *Math.OEIS* installed. You can download it from `http://hackage.haskell.org/cgi-bin/hackage-scripts/package/oeis`.

$$
\begin{aligned}
getDataSeq\ (Just\ seq) = return\ (&description\ seq) \\
&+\!\!+ "\ (\ " \\
&+\!\!+ (concatMap\ (+\!\!+"\ ")\ (catalogNums\ seq)) \\
&+\!\!+ ")"
\end{aligned}
$$

As an example, here is the output for the sequence $ei\ 1\ 1$:

$\gg$ oeis 1 1

```
Ei(1,1):
        Triangle T(n,k) = denominator of fraction in k-th term of n-th row of
variant of Farey series. This is also Stern's diatomic array read by
rows (version 1). ( A049456 )
```

Finally, using the function *oeis*, we have searched which instances of $Ei(m,n)$, with $0 \leqslant m < 100$ and $0 < n < 100$, are listed in the OEIS. The occurrences found are listed below:

$Ei(0,1)$: Triangle read by rows: T(n,k) = numerator of fraction in k-th term of n-th row of variant of Farey series. (A049455)

$Ei(1,0)$: Stern's diatomic array read by rows (version 2). (A070878)

$Ei(1,1)$: Triangle T(n,k) = denominator of fraction in k-th term of n-th row of variant of Farey series. This isalso Stern's diatomic array read by rows (version 1). (A049456)

$Ei(1,2)$: Eisenstein array $Ei(1,2)$. (A064881)

$Ei(1,3)$: Eisenstein array $Ei(1,3)$. (A064883)

$Ei(2,1)$: Eisenstein array $Ei(2,1)$. (A064882)

$Ei(2,3)$: Eisenstein array $Ei(2,3)$. (A064886)

$Ei(3,1)$: Eisenstein array $Ei(3,1)$. (A064884)

$Ei(3,2)$: Eisenstein array $Ei(3,2)$. (A064885)

References

[1] Roland Backhouse and João F. Ferreira. Recounting the rationals: Twice! In *Mathematics of Program Construction*, volume 5133 of *LNCS*, pages 79–91, 2008.

[2] Roland Backhouse and João F. Ferreira. On Euclid's algorithm and elementary number theory. Submitted for publication. Available at http://joaoff.com/publications/2009/euclid-alg/, 2009.

[3] Neil Calkin and Herbert S. Wilf. Recounting the rationals. *The American Mathematical Monthly*, 107(4):360–363, 2000.

[4] Neil J. A. Sloane. The On-Line Encyclopedia of Integer Sequences. `http://www.research.att.com/~njas/sequences/`.

[5] Moritz A. Stern. Über eine zahlentheoretische Funktion. *Journal für die reine und angewandte Mathematik*, 55:193–220, 1858.