

Enumerating the Elements of the Eisenstein Array

Roland Backhouse

rcb@cs.nott.ac.uk

João F. Ferreira

joao@joaoff.com

March 16, 2009

Abstract

In [2], we discuss several algorithms that enumerate the elements of the Eisenstein Array [4]. In this document we show and discuss several Haskell implementations of these algorithms.

1 The Eisenstein Array

Given two natural numbers m and n , Stern [4] describes a process (which he attributes to Eisenstein) of generating an infinite sequence of rows of numbers. The *zeroth* row in the sequence (“nullte Entwicklungsreihe”) 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, [3, 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

An interesting question is whether Stern also documents the algorithm currently attributed to Moshe Newman for enumerating the elements of $Ei(1,1)$ [2, section 4.2.3]. Newman's algorithm predicts that each triple of numbers in a given row of $Ei(1,1)$ has the form

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

In [1, appendix A], we have implemented Newman's algorithm as follows.

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

For the purpose of this document, we are interested in the elements of $Ei(1,1)$, i.e., in the sequence of numerators given by *cwnEnum*. Function *newman* enumerates the elements of $Ei(1,1)$, using the infinite list created by *cwnEnum* (we have to detect a change in level).

```
newman :: [Integer]
newman = concatMap dlevel cwnEnum
  where dlevel r | (denominator r) == 1 = [numerator r, 1]
              | otherwise                = [numerator r]
```

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

One way of enumerating the elements of the array $Ei(m,n)$ is:

```
ei :: Integer → Integer → [Integer]
ei m n = m : eiloop 1 1 m n m n
  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 × ⌊a/b⌋ + 1
                                in n : eiloop b (k × b - a) n (k × n - m) cm cn
```

We can test if the function *newman* is in fact enumerating 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
>> (take 1000 (map (2*) newman)) == (take 1000 (ei 2 2))
True

```

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

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

A property that we have not yet proved is that we can replace a and b by m and n in the calculation of k (when m and n are both positive).

```

ei' :: Integer → Integer → [Integer]
ei' m n = m : eiloop 1 1 m n m n
  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 $0 \leq m \leq x$ and $1 \leq n \leq x$:

$$\text{test } f\ g\ x = \text{and } [\text{take } 1000\ (f\ m\ n) == \text{take } 1000\ (g\ m\ n) \mid m \leftarrow [0..x], n \leftarrow [1..x]]$$

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

```

>> test ei ei' 100
True

```

The function *extnewman*, defined below, is the same as *ei*, but it replaces variables a and b by variable r :

```

extnewman :: Integer → Integer → [Integer]
extnewman cm cn = cm : loop 0 cm cn cm cn
  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 on the Online Encyclopedia of Integer Sequences (OEIS) [3].

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

```
numElems :: Int
numElems = 20

list2string :: (Show a) => [a] -> String
list2string = init o tail o show
```

The following function, *oeis*, receives two integer numbers, *m* and *n*, computes the list of the first *numElems* of *ei m n*, transforms it 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 o list2string o (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."
        getDataSeq (Just seq) = return (description seq)
                                     ++ " ( "
                                     ++ (concatMap (++ " ") (catalogNums seq))
                                     ++ ")"
```

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

```
>> 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 )
```

¹To 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>.

References

- [1] Roland Backhouse and João F. Ferreira. Recounting the rationals: Twice! volume 5133, pages 79–91, 2008.
- [2] Roland Backhouse and João F. Ferreira. On Euclid’s algorithm and elementary number theory. 2009.
- [3] Neil J. A. Sloane. The On-Line Encyclopedia of Integer Sequences. <http://www.research.att.com/~njas/sequences/>.
- [4] Moritz A. Stern. Über eine zahlentheoretische Funktion. *Journal für die reine und angewandte Mathematik*, 55:193–220, 1858.