Recounting the Rationals: Twice!

Roland Backhouse and João F. Ferreira[*]

School of Computer Science

University of Nottingham

Nottingham NG8 1BB, England

{rcb,jff}@cs.nott.ac.uk

April 8, 2008

Abstract

We derive an algorithm that enables the rationals to be efficiently enumerated in two different ways. One way is known and is credited to Moshe Newman; it corresponds to a deforestation of the so-called Calkin-Wilf tree of rationals. The second is new and corresponds to a deforestation of the Stern-Brocot tree of rationals. We show that both enumerations stem from the same simple algorithm. In this way, we construct a Stern-Brocot enumeration algorithm with the same time and space complexity as Newman's algorithm.

Keywords: Calkin-Wilf tree, Stern-Brocot tree, algorithm derivation, enumeration algorithm, rational numbers

Recently, there has been a spate of interest in the construction of bijections between the natural numbers and the (positive) rationals (see [GLB06, KRSS03, CW00] and [AZ04, pages 94–97]). Gibbons *et al* [GLB06] describe as "startling" the observation that the rationals can be efficiently enumerated[1] by "deforesting" the Calkin-Wilf [CW00] tree of rationals. However, they claim that it is "not at all obvious" how to "deforest" the Stern-Brocot tree of rationals. (For information on the Stern-Brocot tree, see [GKP94, pages 116–118].)

In this paper, we derive an efficient algorithm for enumerating the rationals both in Calkin-Wilf and Stern-Brocot order. The algorithm is based on a bijection between the rationals and invertible 2×2 matrices. The key to the algorithm's derivation is the

---

[1]By an *efficient enumeration* we mean a method of generating each rational without duplication with constant cost per rational in terms of arbitrary-precision simple arithmetic operations.

reformulation of Euclid's algorithm in terms of matrices. The enumeration is efficient in the sense that it has the same time and space complexity as the algorithm credited to Moshe Newman in [KRSS03], albeit with a constant-fold increase in the number of variables and number of arithmetic operations needed at each iteration.

Section 1 reviews Euclid's algorithm, whilst section 2 discusses the enumeration algorithms. Section 3 discusses the method used to derive the algorithm. The appendix documents a Haskell implementation of the algorithm.

1   Euclid's Algorithm

A positive rational in so-called "lowest form" is an ordered pair of positive, coprime integers. Every rational $\frac{m}{n}$ has unique lowest-form representation $\frac{m/(m\triangledown n)}{n/(m\triangledown n)}$. (We use "$\triangledown$" to denote "greatest common divisor". We prefer to use an infix notation whenever —as in this case— the operator is symmetric and associative. As we see below, the exploitation of symmetry and associativity is extremely important to effective reasoning.)

Because computing the lowest-form representation involves computing greatest common divisors, it seems sensible to investigate Euclid's algorithm to see whether it gives insight into how to enumerate the rationals. Indeed it does.

Below we present Euclid's algorithm as it might be presented in a modern textbook. (We use Dijkstra's Guarded Command Language [Dij75] to express the algorithm because it allows us to fully express the symmetry between m and n. The "do-od" statement is executed repeatedly. Termination occurs when both of the two guards $y<x$ and $x<y$ are false (i.e. when x and y are equal). When $y<x$ evaluates to true, the assignment $x := x-y$ is executed, and then the do-od is executed again. Similarly, when $x<y$ the assignment $y := y-x$ is executed before repeated execution of the do-od statement.)

$$\{\ 0<m \wedge 0<n\ \}$$

$$x,y := m,n\,;$$

$$\{\ \textbf{Invariant:}\quad 0<x\ \wedge\ 0<y\ \wedge\ x\triangledown y\ =\ m\triangledown n$$

$$\textbf{Bound function: } x+y\ \}$$

$$\text{do } y<x \rightarrow x := x-y$$

$$\square\ \ x<y \rightarrow y := y-x$$

$$\text{od}$$

$$\{\ x=y=x\triangledown y=m\triangledown n\ \}$$

The algorithm below is a somewhat unusual, but very effective, way of rewriting

Euclid's algorithm when the goal is to establish the theorem that the greatest common divisor of two numbers is a linear combination of the numbers.

The algorithm is expressed in matrix terms. The input to the algorithm is a vector $(m\,n)$ of strictly positive integers. The vector $(x\,y)$ is initialised to $(m\,n)$ and, on termination, its value is the vector $(m\nabla n\ m\nabla n)$. In addition to computing the greatest common divisor, it also computes a matrix $\mathbf{C}$. An invariant of the algorithm is that the vector $(x\,y)$ equals $(m\,n)\times\mathbf{C}$. In words, $(x\,y)$ is a "linear combination" of $(m\,n)$. Specifically, $\mathbf{I}$, $\mathbf{A}$ and $\mathbf{B}$ are 2×2 matrices; $\mathbf{I}$ is the identity matrix $\left(\begin{smallmatrix}1&0\\0&1\end{smallmatrix}\right)$, $\mathbf{A}$ is the matrix $\left(\begin{smallmatrix}1&0\\-1&1\end{smallmatrix}\right)$ and $\mathbf{B}$ is the matrix $\left(\begin{smallmatrix}1&-1\\0&1\end{smallmatrix}\right)$. The assignment $(x\,y)\ :=\ (x\,y)\times\mathbf{A}$ is equivalent to $x,y\ :=\ x-y,y$, as can be easily checked.

$$\{\ 0<m\wedge 0<n\ \}$$
$$(x\,y),\mathbf{C}\ :=\ (m\,n),\mathbf{I}\,;$$
$$\{\ \textbf{Invariant:}\quad (x\,y)\ =\ (m\,n)\times\mathbf{C}\ \}$$
$$\textbf{do}\ y<x\ \rightarrow (x\,y),\mathbf{C}\ :=\ (x\,y)\times\mathbf{A}\,,\mathbf{C}\times\mathbf{A}$$
$$\Box\ \ x<y\ \rightarrow (x\,y),\mathbf{C}\ :=\ (x\,y)\times\mathbf{B}\,,\mathbf{C}\times\mathbf{B}$$
$$\textbf{od}$$
$$\{\ (x\,y)\ =\ (m\nabla n\ m\nabla n)\ =\ (m\,n)\times\mathbf{C}\ \}$$

The verification of the supplied invariant is a simple consequence of the associativity of matrix multiplication. It is this form of the algorithm that is the starting point for our enumeration of the rationals.

2   Enumerating the Rationals

Beginning with an arbitrary pair of positive integers m and n, the above algorithm calculates an invertible matrix $\mathbf{C}$ such that

$$(m\nabla n\ m\nabla n)\ =\ (m\,n)\times\mathbf{C}\ .$$

It follows that

(1) $(1\,1)\times\mathbf{C}^{-1}\ =\ \left(\dfrac{m}{(m\nabla n)}\ \dfrac{n}{(m\nabla n)}\right)\ .$

Because the algorithm is deterministic, positive integers m and n uniquely define the matrix $\mathbf{C}$. That is, there is a function from pairs of positive integers to finite products of the matrices $\mathbf{A}$ and $\mathbf{B}$.

Also, because the matrices $\mathbf{A}$ and $\mathbf{B}$ are constant and invertible, $\mathbf{C}^{-1}$ is a finite product of the matrices $\mathbf{A}^{-1}$ and $\mathbf{B}^{-1}$ and (1) uniquely defines a rational $\frac{m}{n}$. We may therefore

conclude that there is a bijection between the rationals and the finite products of the matrices $\mathbf{A}^{-1}$ and $\mathbf{B}^{-1}$ provided that we can show that all such products are different.

The finite products of matrices $\mathbf{A}^{-1}$ and $\mathbf{B}^{-1}$ form a binary tree with root the identity matrix (the empty product). Renaming $\mathbf{A}^{-1}$ as $\mathbf{L}$ and $\mathbf{B}^{-1}$ as $\mathbf{R}$, the tree can be displayed with "L" indicating a left branch and "R" indicating a right branch. Fig. 1 displays the first few levels of the tree.
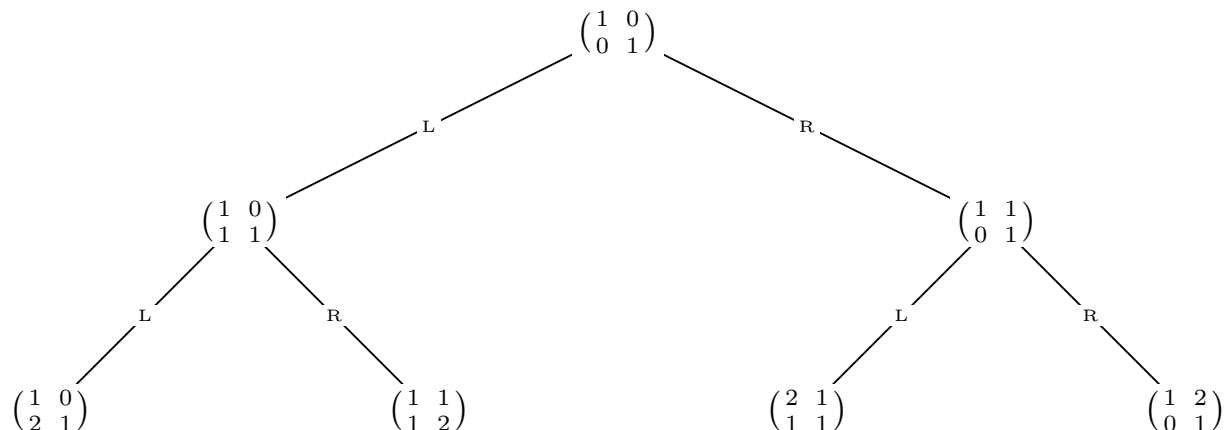

Figure 1: Tree of Products of $\mathbf{L}$ and $\mathbf{R}$

That all matrices in the tree are different is proved by showing that the tree is a binary search tree (as formalised shortly). The key element of the proof[2] is that the determinants of $\mathbf{A}$ and $\mathbf{B}$ are both equal to 1 and, hence, the determinant of any finite product of $\mathbf{L}$s and $\mathbf{R}$s is also 1.

Formally, we define the relation $\prec$ on matrices that are finite products of $\mathbf{L}$s and $\mathbf{R}$s by

$$\begin{pmatrix} a & c \\ b & d \end{pmatrix} \prec \begin{pmatrix} a' & c' \\ b' & d' \end{pmatrix} \;\;\equiv\;\; \frac{a+c}{b+d} < \frac{a'+c'}{b'+d'} \;\;.$$

(Note that the denominator in these fractions is strictly positive; this fact is easily proved by induction.) We prove that, for all such matrices $\mathbf{X}$, $\mathbf{Y}$ and $\mathbf{Z}$,

(2) $\mathbf{X} \times \mathbf{L} \times \mathbf{Y} \prec \mathbf{X} \prec \mathbf{X} \times \mathbf{R} \times \mathbf{Z}$.

It immediately follows that there are no duplicates in the tree of matrices because the relation $\prec$ is clearly transitive and a subset of the inequality relation. (Property (2) formalises precisely what we mean by the tree of matrices forming a binary search tree:

---

[2]The proof is an adaptation of the proof in [GKP94, page 117] that the rationals in the Stern-Brocot tree are all different. Our use of determinants corresponds to their use of "the fundamental fact" (4.31). Note that the definitions of $\mathbf{L}$ and $\mathbf{R}$ are swapped around in [GKP94].)

the entries are properly ordered by the relation $\prec$, with matrices in the left branch being "less than" the root matrix which is "less than" matrices in the right branch.)

In order to show that

(3) $\quad \mathbf{X} \times \mathbf{L} \times \mathbf{Y} \prec \mathbf{X} \qquad$,

suppose $\mathbf{X} = \left(\begin{smallmatrix} a & c \\ b & d \end{smallmatrix} \right)$ and $\mathbf{Y} = \left(\begin{smallmatrix} a' & c' \\ b' & d' \end{smallmatrix} \right)$. Then, since $\mathbf{L} = \left(\begin{smallmatrix} 1 & 0 \\ 1 & 1 \end{smallmatrix} \right)$, (3) is easily calculated to be

$$\frac{(a+c) \times a' + (c \times b') + (a+c) \times c' + (c \times d')}{(b+d) \times a' + (d \times b') + (b+d) \times c' + (d \times d')} < \frac{a+c}{b+d} \quad .$$

That this is true is also a simple, albeit longer, calculation (which exploits the monotonicity properties of multiplication and addition); as observed earlier, the key property is that the determinant of $\mathbf{X}$ is 1, i.e. $a \times d - b \times c = 1$. The proof that $\mathbf{X} \prec \mathbf{X} \times \mathbf{R} \times \mathbf{Z}$ is similar.

Of course, we can also express Euclid's algorithm in terms of transpose matrices. Instead of writing assignments to the vector $(x\,y)$, we can write assignments to its transpose $\left(\begin{smallmatrix} x \\ y \end{smallmatrix} \right)$. Noting that $\mathbf{A}$ and $\mathbf{B}$ are each other's transposition, the assignment

$$(x\,y), \mathbf{C} \ := \ (x\,y) \times \mathbf{A} \,, \mathbf{C} \times \mathbf{A}$$

in the body of Euclid's algorithm becomes

$$\begin{pmatrix} x \\ y \end{pmatrix}, \mathbf{C} \ := \ \mathbf{B} \times \begin{pmatrix} x \\ y \end{pmatrix}, \mathbf{B} \times \mathbf{C} \ \ .$$

Similarly, the assignment

$$(x\,y), \mathbf{C} \ := \ (x\,y) \times \mathbf{B} \,, \mathbf{C} \times \mathbf{B}$$

becomes

$$\begin{pmatrix} x \\ y \end{pmatrix}, \mathbf{C} \ := \ \mathbf{A} \times \begin{pmatrix} x \\ y \end{pmatrix}, \mathbf{A} \times \mathbf{C} \ \ .$$

On termination, the matrix $\mathbf{C}$ computed by the revised algorithm will of course be different; the pair $\left(\begin{smallmatrix} m/(m \triangledown n) \\ n/(m \triangledown n) \end{smallmatrix} \right)$ is recovered from it by the identity

$$\mathbf{C}^{-1} \times \begin{pmatrix} 1 \\ 1 \end{pmatrix} \ = \ \begin{pmatrix} m/(m \triangledown n) \\ n/(m \triangledown n) \end{pmatrix} \quad .$$

In this way, we get a second bijection between the rationals and the finite products of the matrices $\mathbf{A}^{-1}$ and $\mathbf{B}^{-1}$. This is the basis for our second method of enumerating the rationals.

In summary, we have:

Theorem 4   Define the matrices $\mathbf{L}$ and $\mathbf{R}$ by

$$\mathbf{L} = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix} \quad \text{and} \quad \mathbf{R} = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} \quad .$$

Then the following algorithm computes a bijection between the (positive) rationals and the finite products of $\mathbf{L}$ and $\mathbf{R}$. Specifically, the bijection is given by the function that maps the rational $\frac{m}{n}$ to the matrix $\mathbf{D}$ constructed by the algorithm together with the function from a finite product, $\mathbf{D}$, of $\mathbf{L}$s and $\mathbf{R}$s to $(1\,1) \times \mathbf{D}$. (The comments added to the algorithm supply the information needed to verify this assertion.)

$$\{\ 0{<}m \wedge 0{<}n\ \}$$

$$(x\,y), \mathbf{D} \ := \ (m\,n), \mathbf{I}\,;$$

$$\{\ \textbf{Invariant:} \quad (m\,n) \ = \ (x\,y) \times \mathbf{D}\ \}$$

$$\text{do } y{<}x \rightarrow (x\,y), \mathbf{D} \ := \ (x\,y) \times \mathbf{L}^{-1}, \mathbf{L} \times \mathbf{D}$$

$$\square \quad x{<}y \rightarrow (x\,y), \mathbf{D} \ := \ (x\,y) \times \mathbf{R}^{-1}, \mathbf{R} \times \mathbf{D}$$

$$\text{od}$$

$$\{\ (x\,y) = (m\nabla n\ \ m\nabla n) \quad \wedge \quad (\,{}^{m}\!/_{(m\nabla n)}\ \ {}^{n}\!/_{(m\nabla n)}) = (1\,1) \times \mathbf{D}\ \}$$

Similarly, by applying the rules of matrix transposition to all expressions in the above, Euclid's algorithm constructs a second bijection between the rationals and finite products of the matrices $\mathbf{L}$ and $\mathbf{R}$. Specifically, the bijection is given by the function that maps the rational $\frac{m}{n}$ to the matrix $\mathbf{D}$ constructed by the revised algorithm together with the function from finite products, $\mathbf{D}$, of $\mathbf{L}$s and $\mathbf{R}$s to $\mathbf{D} \times \binom{1}{1}$.

$\square$

2.1   Enumerating Products of L and R

The problem of enumerating the rationals has been transformed to the problem of enumerating all finite products of the matrices $\mathbf{L}$ and $\mathbf{R}$. As observed earlier, the matrices are naturally visualised as a tree —recall fig. 1— with left branching corresponding to multiplying (on the right) by $\mathbf{L}$ and right branching to multiplying (on the right) by $\mathbf{R}$.

By premultiplying each matrix in the tree by $(1\,1)$, we get a tree of rationals. (Premultiplying by $(1\,1)$ is accomplished by adding the elements in each column.) This tree is called the Calkin-Wilf tree [GLB06, AZ04, CW00]. The first four levels of the tree are shown in fig. 2. In this figure, the vector $(x\,y)$ has been displayed as $\frac{y}{x}$. (Note the order of x and y. This is to aid comparison with existing literature.)
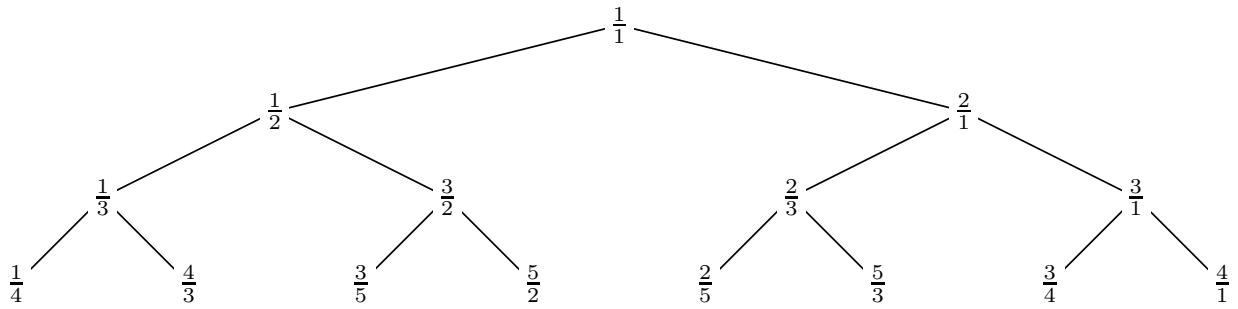
Figure 2: Calkin-Wilf Tree of Rationals

By postmultiplying each matrix in the tree by $\left(\begin{smallmatrix} 1 \\ 1 \end{smallmatrix}\right)$, we also get a tree of rationals. (Postmultiplying by $\left(\begin{smallmatrix} 1 \\ 1 \end{smallmatrix}\right)$ is accomplished by adding the elements in each row.) This tree is called the Stern-Brocot tree [GKP94, pages 116–118]. See fig. 3. In this figure, the vector $\left(\begin{smallmatrix} x \\ y \end{smallmatrix}\right)$ has been displayed as $\frac{x}{y}$.
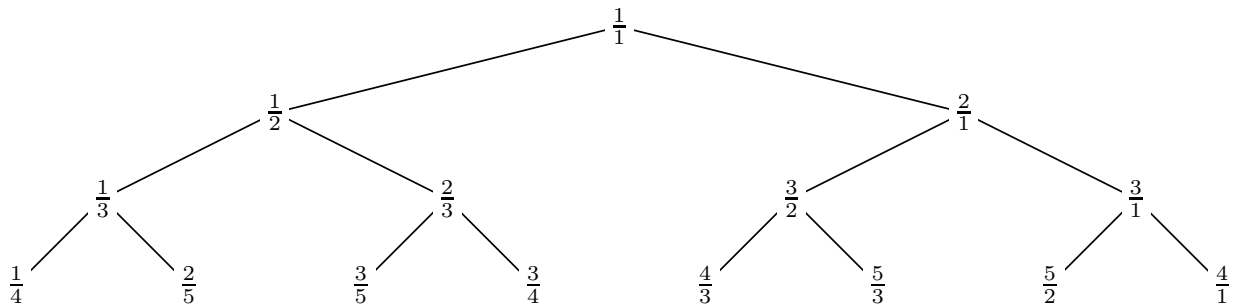

Figure 3: Stern-Brocot Tree of Rationals

Of course, if we can find an efficient way of enumerating the matrices in fig. 1, we immediately get an enumeration of the rationals as displayed in the Calkin-Wilf tree and as displayed in the Stern-Brocot tree — as each matrix is enumerated, simply premultiply by $(1\ 1)$ or postmultiply by $\left(\begin{smallmatrix} 1 \\ 1 \end{smallmatrix}\right)$. Formally, the matrices are enumerated by enumerating all strings of Ls and Rs in lexicographic order, beginning with the empty string; each string is mapped to a matrix by the homomorphism that maps "L" to $\mathbf{L}$, "R" to $\mathbf{R}$, and string concatenation to matrix product. It is easy to enumerate all such strings; as we see shortly, converting strings to matrices is also not difficult, for the simple reason that $\mathbf{L}$ and $\mathbf{R}$ are invertible.

The enumeration proceeds level-by-level. Beginning with the unit matrix (level 0), the matrices on each level are enumerated from left to right. There are 2^k matrices on level k, the first of which is $\mathbf{L}^k$. The problem is to determine for a given matrix, which is the matrix "adjacent" to it. That is, given a matrix $\mathbf{D}$, which is a finite product of $\mathbf{L}$ and $\mathbf{R}$, and is different from $\mathbf{R}^k$ for all k, what is the matrix that is to the immediate

right of $\mathbf{D}$ in fig. 1?

Consider the lexicographic ordering on strings of Ls and Rs of the same length. The string immediately following a string s (that is not the last) is found by identifying the rightmost L in s. Supposing s is the string $t\mathrm{LR}^j$, where R^j is a string of j Rs, its successor is $t\mathrm{RL}^j$.

It's now easy to see how to transform the matrix identified by s to its successor matrix. Simply postmultiply by $\mathbf{R}^{-j}\times\mathbf{L}^{-1}\times\mathbf{R}\times\mathbf{L}^j$. This is because, for all $\mathbf{T}$ and j,

$$(\mathbf{T}\times\mathbf{L}\times\mathbf{R}^j)\times(\mathbf{R}^{-j}\times\mathbf{L}^{-1}\times\mathbf{R}\times\mathbf{L}^j)\ =\ \mathbf{T}\times\mathbf{R}\times\mathbf{L}^j\ \ .$$

Also, it is easy to calculate $\mathbf{R}^{-j}\times\mathbf{L}^{-1}\times\mathbf{R}\times\mathbf{L}^j$. Specifically,

$$\mathbf{R}^{-j}\times\mathbf{L}^{-1}\times\mathbf{R}\times\mathbf{L}^j\ =\ \begin{pmatrix} 2j+1 & 1 \\ -1 & 0 \end{pmatrix}\ \ .$$

(We omit the details. Briefly, by induction, $\mathbf{L}^j$ equals $\begin{pmatrix} 1 & 0 \\ j & 1 \end{pmatrix}$. Also, $\mathbf{R}$ is the transpose of $\mathbf{L}$.)

The final task is to determine, given a matrix $\mathbf{D}$, which is a finite product of Ls and Rs, and is different from $\mathbf{R}^k$ for all k, the unique value j such that $\mathbf{D}=\mathbf{T}\times\mathbf{L}\times\mathbf{R}^j$ for some $\mathbf{T}$. This can be determined by examining Euclid's algorithm once more.

The matrix form of Euclid's algorithm discussed in theorem 4 computes a matrix $\mathbf{D}$ given a pair of positive numbers m and n; it maintains the invariant

$$(m\,n)\ =\ (x\,y)\times\mathbf{D}\ \ .$$

$\mathbf{D}$ is initially the identity matrix and x and y are initialised to m and n, respectively; immediately following the initialisation process, $\mathbf{D}$ is repeatedly premultiplied by $\mathbf{R}$ so long as x is less than y. Simultaneously, y is reduced by x. The number of times that $\mathbf{D}$ is premultiplied by $\mathbf{R}$ is thus the greatest number j such that $j\times m$ is less than n, which is $\left\lfloor\frac{n-1}{m}\right\rfloor$. Now suppose the input values m and n are coprime. Then, on termination of the algorithm, $(1\,1)\times\mathbf{D}$ equals $(m\,n)$. That is, if

$$\mathbf{D}=\begin{pmatrix} \mathbf{D}_{00} & \mathbf{D}_{01} \\ \mathbf{D}_{10} & \mathbf{D}_{11} \end{pmatrix}\ \ ,$$

then,

$$\left\lfloor\frac{n-1}{m}\right\rfloor\ =\ \left\lfloor\frac{\mathbf{D}_{01}+\mathbf{D}_{11}-1}{\mathbf{D}_{00}+\mathbf{D}_{10}}\right\rfloor\ \ .$$

It remains to decide how to keep track of the levels in the tree. For this purpose, it is not necessary to maintain a counter. It suffices to observe that $\mathbf{D}$ is a power of $\mathbf{R}$ exactly when the rationals in the Calkin-Wilf, or Stern-Brocot, tree are integers, and this integer is the number of the next level in the tree (where the root is on level 0). So, it is easy

to test whether the last matrix on the current level has been reached. Equally, the first matrix on the next level is easily calculated. For reasons we discuss in the next section, we choose to test whether the rational in the Calkin-Wilf tree is an integer; that is, we evaluate the boolean $\mathbf{D}_{00} + \mathbf{D}_{10} = 1$. In this way, we get the following (non-terminating) program which computes the successive values of $\mathbf{D}$.

$$\mathbf{D} := \mathbf{I};$$
$$\mathbf{do}\ \mathbf{D}_{00} + \mathbf{D}_{10} = 1 \rightarrow \mathbf{D} := \begin{pmatrix} 1 & 0 \\ \mathbf{D}_{01} + \mathbf{D}_{11} & 1 \end{pmatrix}$$
$$\square\quad \mathbf{D}_{00} + \mathbf{D}_{10} \neq 1 \rightarrow j := \left\lfloor \frac{\mathbf{D}_{01} + \mathbf{D}_{11} - 1}{\mathbf{D}_{00} + \mathbf{D}_{10}} \right\rfloor\ ;\ \mathbf{D} := \mathbf{D} \times \begin{pmatrix} 2j+1 & 1 \\ -1 & 0 \end{pmatrix}$$
$$\mathbf{od}$$

A minor simplification of this algorithm is that the "$- 1$" in the assignment to j can be omitted. This is because $\left\lfloor \frac{n-1}{m} \right\rfloor$ and $\left\lfloor \frac{n}{m} \right\rfloor$ are equal when m and n are coprime and m is different from 1. We return to this shortly.

2.2 The Enumerations

As remarked earlier, we immediately get an enumeration of the rationals as displayed in the Calkin-Wilf tree and as displayed in the Stern-Brocot tree — as each matrix is enumerated, simply premultiply by $(1\ 1)$ or postmultiply by $\binom{1}{1}$, respectively.

In the case of enumerating the Calkin-Wilf tree, several optimisations are possible. First, it is immediate from our derivation that the value assigned to the local variable j is a function of $(1\ 1) \times \mathbf{D}$. In turn, the matrix $\begin{pmatrix} 2j+1 & 1 \\ -1 & 0 \end{pmatrix}$ is also a function of $(1\ 1) \times \mathbf{D}$. Let us name the function J, so that the assignment becomes

$$\mathbf{D} := \mathbf{D} \times J.((1\ 1) \times \mathbf{D})\ .$$

Then, the Calkin-Wilf enumeration iteratively evaluates

$$(1\ 1) \times (\mathbf{D} \times J.((1\ 1) \times \mathbf{D}))\ .$$

Matrix multiplication is associative; so this is

$$((1\ 1) \times \mathbf{D}) \times J.((1\ 1) \times \mathbf{D})\ ,$$

which is also a function of $(1\ 1) \times \mathbf{D}$. Moreover —in anticipation of the current discussion— we have been careful to ensure that the test for a change in the level in the tree is also a function of $(1\ 1) \times \mathbf{D}$. Combined together, this means that, in order to enumerate the rationals in Calkin-Wilf order, it is not necessary to compute $\mathbf{D}$ at each iteration, but only $(1\ 1) \times \mathbf{D}$. Naming the two components of this vector m and n, and simplifying the matrix multiplications, we get[3]

---

[3]Recall that, to comply with existing literature, the enumerated rational is $\frac{n}{m}$ and not $\frac{m}{n}$.

$$\mathtt{m,n} := 1,1;$$
$$\mathtt{do\ m{=}1} \rightarrow \mathtt{m,n} := \mathtt{n}+1,\mathtt{m}$$
$$\square\ \ \mathtt{m}{\neq}1 \rightarrow \mathtt{m,n} := (2\left\lfloor\frac{\mathtt{n}-1}{\mathtt{m}}\right\rfloor + 1)\times\mathtt{m}-\mathtt{n}\ ,\ \mathtt{m}$$
$$\mathtt{od}$$

At this point, a further simplification is also possible. We remarked earlier that $\left\lfloor\frac{n-1}{m}\right\rfloor$ equals $\left\lfloor\frac{n}{m}\right\rfloor$ when m and n are coprime and m is different from 1. By good fortune, it is also the case that $(2\left\lfloor\frac{n}{m}\right\rfloor + 1)\times m - n$ simplifies to $n+1$ when m is equal to 1. That is, the elimination of "$-\ 1$" in the evaluation of the floor function leads to the elimination of the entire case analysis! This is the algorithm attributed to Newman in [KRSS03].

$$\mathtt{m,n} := 1,1;$$
$$\mathtt{do\ \ m,n} := (2\left\lfloor\frac{\mathtt{n}}{\mathtt{m}}\right\rfloor + 1)\times\mathtt{m}-\mathtt{n}\ ,\ \mathtt{m}$$
$$\mathtt{od}$$

3   Discussion

This paper was motivated by reading two publications, [GKP94, pages 116–118] and [GLB06]. Gibbons, Lester and Bird [GLB06] show how to enumerate the elements of the Calkin-Wilf tree, but claim that "it is not at all obvious how to do this for the Stern-Brocot tree". Specifically, they say:

> However, there is an even better compensation for the loss of the ordering property in moving from the Stern-Brocot to the Calkin-Wilf tree: it becomes possible to deforest the tree altogether, and generate the rationals directly, maintaining no additional state beyond the 'current' rational. This startling observation is due to Moshe Newman (Newman, 2003). In contrast, it is not at all obvious how to do this for the Stern-Brocot tree; the best we can do seems to be to deforest the tree as far as its levels, but this still entails additional state of increasing size.

In this paper, we have shown that it is possible to enumerate the rationals in Stern-Brocot order without incurring "additional state of increasing size". More importantly, we have presented *one* enumeration algorithm with *two* specialisations, one being the Calkin-Wilf enumeration they present, and the other being the Stern-Brocot enumeration that they described as being "not at all obvious".

The optimisation of Calkin-Wilf enumeration which leads to Newman's algorithm is not possible for Stern-Brocot enumeration. Nevertheless, the complexity of Stern-Brocot

enumeration is the same as the complexity of Newman's algorithm, both in time and space. The only disadvantage of Stern-Brocot enumeration is that four variables are needed in place of two; the advantage is the (well-known) advantage of the Stern-Brocot tree over the Calkin-Wilf tree — the rationals on a given level are in ascending order.

Gibbons, Lester and Bird's goal seems to have been to show how the functional programming language Haskell implements the various constructions – the construction of the tree structures and Newman's algorithm. In doing so, they repeat the existing mathematical presentations of the algorithms as given in [GKP94, CW00, KRSS03]. The ingredients for an efficient enumeration of the Stern-Brocot tree are all present in these publications, but the recipe is missing!

The fact that expressing the rationals in "lowest form" is essential to the avoidance of duplication in any enumeration immediately suggests the relevance of Euclid's algorithm. The key to our exposition is that Euclid's algorithm can be expressed in terms of matrix multiplications, where —significantly— the underlying matrices are invertible. Transposition and inversion of the matrices capture the symmetry properties in a precise, calculational framework. As a result, the bijection between the rationals and the tree elements is immediate and we do not need to give separate, inductive proofs for both tree structures. Also, the determination of the next element in an enumeration of the tree elements has been reduced to one unifying construction.

This paper was submitted in April 2007 to the American Mathematical Monthly; it was rejected in November 2007 on the grounds that it was not of sufficient interest to readers of the Monthly. One (of two referees) did, however, recommend publication. The referee made the following general comment.

> Each of the two trees of rationals—the Stern-Brocot tree and the Calkin-Wilf tree—has some history. Since this paper now gives the definitive link between these trees, I encourage the authors, perhaps in their Discussion section, to also give the definitive histories of these trees, something in the same spirit as the Remarks at the end of the Calkin and Wilf paper.

We thank the referee for the detailed comments; unfortunately, we have not been able to obtain copies of the original papers by Stern and Brocot —we would have had difficulty reading the German and French in any case— and are not in a position to fulfill the referee's request. It would, indeed, be interesting for a mathematical historian to pursue this suggestion.

References

[AZ04]   Martin Aigner and Günter Ziegler. *Proofs From The Book, 3rd Edition.* Springer-Verlag, 2004.

[CW00]   Neil Calkin and Herbert S. Wilf. Recounting the rationals. *The American Mathematical Monthly*, 107(4):360–363, 2000.

[Dij75]   Edsger W. Dijkstra. Guarded commands, nondeterminacy and formal derivation of programs. *Communications of the ACM*, 18(8):453–457, 1975.

[GKP94]   Ronald L. Graham, Donald E. Knuth, and Oren Patashnik. *Concrete Mathematics : a Foundation for Computer Science.* Addison-Wesley Publishing Company, second edition, 1994.

[GLB06]   Jeremy Gibbons, David Lester, and Richard Bird. Enumerating the rationals. *Journal of Functional Programming*, 16(3):281–291, 2006.

[KRSS03]   Donald E. Knuth, C.P. Rupert, Alex Smith, and Richard Stong. Recounting the rationals, continued. *American Mathematical Monthly*, 110(7):642–643, 2003.

A   Appendix: Haskell Implementation

This appendix contains an encoding of the enumeration algorithms in Haskell. The file from which this printed version was compiled is a so-called "lhs2TEX" file[1] which can be used directly as input to a Haskell compiler; this safeguards against typographical errors in the printed paper.

The implementation encodes a matrix as a list of *columns*.

```
type Entry  = Integer
type Column = [Entry]
type Matrix = [Column]
```

We define a type of non-empty trees, with associated map, fold and unfold functions.

```
data Tree a              = Node (a, Tree a, Tree a)
mapt f (Node (a, l, r)) = Node (f a, mapt f l, mapt f r)
foldt f (Node (a, l, r)) = f (a, foldt f l, foldt f r)
unfoldt f x              = let (a, y, z) = f x in Node (a, unfoldt f y, unfoldt f z)
```

With matrices $matId$, $matL$ and $matR$ defined to be the identity matrix, the matrix **L** and the matrix **R**, respectively, the tree of matrices is generated as follows.

```
mTree :: Tree Matrix
mTree = unfoldt level matId
   where level m = (m, m × matL, m × matR)
```

The Calkin-Wilf tree can be obtained by pre-multiplying the matrices by the vector $(1 \ 1)$.

```
cwTree :: Tree Rational
cwTree = mapt (mkCWRat ∘ ([[1], [1]]×)) mTree


mkCWRat :: Matrix → Rational
mkCWRat [[m], [n]] = n/m
```

Similarly, the Stern-Brocot tree can be obtained by post-multiplying the matrices by the transpose of the vector $(1 \ 1)$, i.e., $\binom{1}{1}$.

```
sbTree :: Tree Rational
sbTree = mapt (mkSBRat ∘ (×[[1, 1]])) mTree
```

---

[1] The lhs2TEX system has been implemented by Ralf Hinze and Andres Löh.

$$mkSBRat :: Matrix \rightarrow Rational$$
$$mkSBRat\ [[m, n]] = m/n$$

We enumerate the matrices using the *iterate* function, computing each matrix from the previous one.

$$nextM :: Matrix \rightarrow Matrix$$
$$nextM\ [[1, 0], [n, 1]] = [[1, n + 1], [0, 1]]$$
$$nextM\ [\,c0, c1\,] = \textbf{let}\ j = \lfloor((sum\ c1) - 1)/(sum\ c0)\rfloor$$
$$k = 2{\times}j + 1$$
$$ck = map\ (k{\times})\ c0$$
$$\textbf{in}\ [\,zipWith\ (-)\ ck\ c1\,, c0\,]$$

$$mats :: [\,Matrix\,]$$
$$mats = iterate\ nextM\ matId$$

The (non-optimised) implementation of the Calkin-Wilf enumeration is then a matter of premultiplying by (1 1).

$$cwEnum :: [\,Rational\,]$$
$$cwEnum = map\ mkCW\ mats$$
$$\textbf{where}\ mkCW = mkCWRat \circ ([[1], [1]]{\times})$$

The Stern-Brocot enumeration can be defined in a similar way, but instead of premultiplying, we postmultiply by $\binom{1}{1}$:

$$sbEnum :: [\,Rational\,]$$
$$sbEnum = map\ mkSB\ mats$$
$$\textbf{where}\ mkSB = mkSBRat \circ ({\times}[[1, 1]])$$

Incorporating the optimisations discussed above, the Calkin-Wilf enumeration is transformed to the algorithm attributed to Newman.

$$cwnEnum :: [\,Rational\,]$$
$$cwnEnum = iterate\ nextCW\ 1/1$$

$$nextCW :: Rational \rightarrow Rational$$
$$nextCW\ r = \textbf{let}\ (n, m) = (numerator\ r, denominator\ r)$$
$$j \qquad = \lfloor n/m \rfloor$$
$$\textbf{in}\ m/((2{\times}j + 1){\times}m - n)$$