Modeling Prime Factorization with Bijections between Natural Numbers, Finite Sequences and Multisets

Paul Tarau

Department of Computer Science and Engineering
University of North Texas
tarau@cs.unt.edu

Abstract

We describe mechanisms to emulate properties of *prime numbers* by connecting them to computationally simpler representations involving bijections from natural numbers to *sequences* and *multisets* of natural numbers. Through a generic framework, implemented as a Haskell type class, we model interesting properties of canonical representation of factorization and its simpler equivalents, like information lost through factorization, a connection to Catalan families and similarity of behavior of the corresponding concepts of smooth numbers and the Ω and ω functions.

Categories and Subject Descriptors D.3.3 [*PROGRAMMING LANGUAGES*]: Language Constructs and Features—Data types and structures

General Terms Algorithms, Languages, Theory

Keywords experimental mathematics and functional programming, declarative modeling of computational phenomena, sequence and multiset encodings and prime factorization, information loss through factorization, factorization and Catalan families, properties of smooth numbers and the Ω and ω functions

1. Introduction

The study of the prime factorization of natural numbers has been a driving force towards various abstraction layers materialized along the years as new research and application fields ranging from formal logic and foundation of mathematics to algebraic and analytic number theory and cryptography. It is not unusual that significant progress on prime numbers correlates with unexpected paradigm shifts, the prototypical example being Riemann's paper [1] connecting primality and complex analysis, all evolving around the still unsolved *Riemann Hypothesis* [2–4]

Prime numbers exhibit a number of fundamental properties of natural phenomena and human artifacts in an unusually pure form. For instance, *reversibility* is present as the ability to recover the operands of a product of distinct primes. This relates to the information-theoretical view of multiplication [5] and it suggests investigating connections between combinatorial properties of multisets and operations on multisets and multiplicative number theory.

At the same time, the growing number of conjectures [6] and the large number of still unsolved problems involving prime numbers [7] suggest that modeling the underlying generic mechanisms that connect, like factorization, a given natural number to a finite multiset of natural numbers is worth exploring.

As a sequel of [8], with focus on emulating properties of prime factorization, seen as a bijection between $\mathbb{N}$ and its finite multisets, we use this time time two infinite families of such bijections, based on p-adic valuations and a size-proportionate encoding using bijective base-k numbers.

After encapsulating their commonalities as a Haskell type class, we study in this generic framework some interesting properties of factorization and compare them with its simpler emulations.

The paper is organized as follows.

Section 2 introduces two general mechanisms for defining bijections between $\mathbb{N}$ and its finite sequences and multisets that will be used to emulate properties of prime factorization. Subsection 2.1 overviews the encoding of natural numbers in bijective base-k that will be used in subsection 2.2 to define a size-proportionate bijection from $\mathbb{N}$ to finite lists of natural numbers. Subsection 2.3 introduces a family of bijections between $\mathbb{N}$ and sequences of natural numbers using n-adic valuations. Subsection 2.4 describes a bijection connecting sequences and multisets.

Section 3 overviews the encoding of finite multisets with primes.

In section 4 we introduce a generic framework for multiset-based emulations of primality.

Section 5 describes a connection between recursive application of our canonical factorization operation and Catalan families.

Section 6 studies the information loss through factorization for various instances of our framework, including the actual primes.

Section 7 studies emulations of the behavior of the ω function (counting the number of distinct prime factors of a number) and the distribution of *smooth* numbers (numbers that are the product of only "small" prime factors).

Section 8 overviews some related work and section 9 concludes the paper.

The paper is written as a *literate Haskell program*. The code extracted from it is also available as a standalone file at `http://logic.cse.unt.edu/tarau/research/2013/primes.hs`.

2. Bijections between natural numbers and sequences and multisets of natural numbers

We start by introducing two general mechanisms for defining bijections between $\mathbb{N}$ and its finite sequences and multisets. We will use the resulting decomposition of a number to a multiset as an analogue of prime factorization which will model some interesting properties of the primes themselves.

2.1 Encoding numbers in bijective base-k

Conventional binary representation of a natural number n describes it as the value of a polynomial $P(x) = \sum_{i=0}^{k} a_i x^i$ with digits $a_i \in \{0, 1\}$ for x=2, i.e. $n = P(2)$. If one rewrites $P(x)$ using Horner's scheme as $Q(x) = a_0+x(a_1+x(a_2+...))$, n can be represented as an iterated application of a function $f(a, x, v) = a+xv$ which can be further specialized as a sequence of applications of two functions $f_0(v) = 2v$ and $f_1(v) = 1 + 2v$, corresponding to the encoding of n as the sequence of binary digits $a_0, a_1, \ldots, a_k$. It is easy to see that this encoding is not bijective as a function to $\{0, 1\}^*$. For instance 1, 10, 100 etc. would all correspond to $n = 1$. The problem is that conventional numbering systems do not provide a *bijection* between arbitrary combinations of digits and natural numbers, given that leading 0s are ignored. As a fix, the functions $o(v) = 1 + 2v$ and $i(v) = 2 + 2v$ can be used instead, resulting in the so called *bijective base-2* representation [9], together with the convention that zero is represented as the empty sequence. With this representation, and denoting the empty sequence ϵ, one obtains $0 = \epsilon, 1 = o\ \epsilon, 2 = i\ \epsilon, 3 = o(o\ \epsilon), 4 = i(o\ \epsilon), 5 = o(i\ \epsilon)$ etc. Note that this representation is *canonical* i.e. a unique sequence of symbols identifies each natural number.

An encoder for *numbers in bijective base-k* [9] that provides such a bijection is implemented as the function `toBijBase` which, like the decoder `fromBijBase`, works one digit a time, using the functions `getBijDigit` and `putBijDigit`. They are both parameterized by the base `b` of the numeration system.

```
type N = Integer

toBijBase :: N→N→[N]
toBijBase _ 0 = []
toBijBase b n | n>0 = d : ds where
  (d,m) = getBijDigit b n
  ds = toBijBase b m

fromBijBase :: N→[N]→N
fromBijBase _ [] = 0
fromBijBase b (d:ds) = n where
  m = fromBijBase b ds
  n = putBijDigit b d m
```

The function `getBijDigit` extracts one bijective base-`b` digit from natural number `n`. It also returns the "left-over" information content as the second component of an ordered pair.

```
getBijDigit :: N→N→(N,N)
getBijDigit b n | n>0 =
  if d == 0 then (b-1,q-1) else (d-1,q) where
    (q,d) = quotRem n b
```

The function `putBijDigit` integrates a bijective base-`b` digit `d` into a natural number `m`:

```
putBijDigit :: N→N→N→N
putBijDigit b d m | 0 ≤ d && d < b = 1+d+b*m
```

The encoding/decoding to bijective base-`b` works as follows:

```
*Primes> toBijBase 4 2013
[0,2,0,2,2,0]
*Primes> fromBijBase 4 it
2013
*Primes> map (toBijBase 2) [0..7]
[[],[0],[1],[0,0],[1,0],[0,1],[1,1],[0,0,0]]
```

This encoding will be used for define a family of bijections between natural numbers and finite lists, in section 2.2.

2.2 A size-proportionate bijection from ℕ to finite lists of natural numbers

DEFINITION 1. *We call* size-proportionate *a bijection f from ℕ to finite lists of natural numbers if the size of the bijective base-2 representation of $n \in \mathbb{N}$ is within a constant factor of the sum of the sizes of the bijective base-2 representations of the members of the list $f(n)$.*

We will define a size-proportionate bijection between ℕ and finite lists of elements of ℕ (denoted [ℕ] from now on) by converting a number n to bijective base $k + 1$ and then using digit k as a separator. The function `nat2nats` implements this operation.

```
nat2nats _ 0 = []
nat2nats _ 1 = [0]
nat2nats k n | n>0 = ns where
  n' = pred n
  k' = succ k
  xs = toBijBase k' n'
  nss = splitWith k xs
  ns = map (fromBijBase k) nss

  splitWith sep xs =  y : f ys where
   (y, ys) = break (==sep) xs

   f [] = []
   f (_:zs) = splitWith sep zs
```

The function `nats2nat k` reverses `nat2nats k` by intercalating the separator `k` between the base k representation of the numbers on a list `ns` and then converts the result to bijective base $k + 1$.

```
nats2nat _ [] = 0
nats2nat _ [0] = 1
nats2nat k ns = n where
  nss = map (toBijBase k) ns
  xs = intercalate [k] nss
  n' = fromBijBase (succ k) xs
  n = succ n'
```

PROPOSITION 1. *The bijection from ℕ to [ℕ] defined by* `nat2nats` *is size-proportionate.*

Proof It follows from the fact that digit k in the bijective base-k+1 is used as a separator turning the bijective base-k representation of $n \in \mathbb{N}$ into a list.

The following example illustrates this property, which is easy to detect visually by noticing similarly-sized decimal representations.

```
*Primes> nats2nat 7 [2,0,1,3,2,0,1,4,9,777,888,0,999]
27662734980522719186436359235
*Primes> nat2nats 7 it
[2,0,1,3,2,0,1,4,9,777,888,0,999]
```

2.3 A bijection between ℕ and [ℕ] using n-adic valuations

Along the lines of [8] we will also describe a mechanism for deriving bijections between ℕ and [ℕ] from one-solution Diophantine equations. Let's observe that

PROPOSITION 2. $\forall z \in \mathbb{N} - \{0\}$ *the Diophantine equation*

$$2^x(2y + 1) = z \tag{1}$$

has exactly one solution $x, y \in \mathbb{N}$.

This follows immediately from the unicity of the decomposition of a natural number as a multiset of prime factors.

We will generalize this observation to obtain a *family of bijections* between ℕ and [ℕ] by choosing an arbitrary base b instead of 2.

DEFINITION 2. *Given a number $n \in \mathbb{N}, n > 1$, the n-adic evaluation of a natural number m is the largest exponent k of n, such that n^k divides m. It is denoted $\nu_n(m)$.*

Note that the solution x of the equation (1) is actually $\nu_2(z)$. This suggests deriving a similar diophantine equation for an arbitrary n-adic valuation.

We implement, for an arbitrary $b \in \mathbb{N}$ the functions `xcons b` and `xdecons b`, defined between $\mathbb{N} \times \mathbb{N}$ and N^+.

```
xcons :: N→(N,N)→N
xcons b (x,y')  | b>1 = (b^x)*y where
  q=y' `div` (b-1)
  y=y'+q+1

xdecons :: N→N→(N,N)
xdecons b z | b>1 && z>0 = (x,y') where
  hd n = if n `mod` b > 0 then 0 else 1+hd (n `div` b)
  x = hd z
  y = z `div` (b^x)
  q = y `div` b
  y' = y-q-1
```

Using `xdecons` we define the head and tail projection functions `xhd` and `xtl`:

```
xhd, xtl :: N→N→N
xhd b = fst . xdecons b
xtl b = snd . xdecons b
```

Intuitively, their correctness follows from the fact that the quotient y, after dividing z with the largest possible power b^x is not divisible by b. We then "rebase" it to base $b-1$, and make z correspond to a unique pair of natural numbers.

More precisely, non-divisibility of y by b, stated as $y = bq + m, b > m > 0$ can be rewritten as $y-q-1 = bq-q+m-1, b > m > 0$, or equivalently $y-q-1 = (b-1)q+(m-1), b > m > 0$ from where it follows that by setting $y' = y-q-1$ and $m' = m-1$, we map z to a pair (y', m') such that $y' = (b-1)q + m'$ and $b-1 > m' \geq 0$. So we can transform (y, m) such that $y = bq+m$ with $b > m > 0$, into a pair (y', m'), such that $y' = q(b-1)+m'$ with $b-1 > m' \geq 0$. Note that the transformation works also in the opposite direction with $y' = y-q-1$ giving $y = y'+q+1$, and with $m' = m-1$ giving $m = m'+1$.

The following examples illustrate the operations for base 3:

```
*Primes> xcons 3 (10,20)
1830519
*Primes> xhd 3 1830519
10
*Primes> xtl 3 1830519
20
```

Note that `xhd n x` is defines the n-adic valuation of x, $\nu_n(x)$

DEFINITION 3. *We call* `xhd n x` *the n-adic head of $x \in \mathbb{N}^+$,* `xtl n x` *the n-adic tail of $x \in \mathbb{N}^+$ and* `xcons n (x,y)` *the n-adic cons of $x, y \in \mathbb{N}$.*

For each base `b>1` we obtain a pair of functions between natural numbers and lists of natural numbers in terms of `xhd`, `xtl` and `xcons` as follows:

```
nat2xnats :: N→N→[N]
nat2xnats _ 0 = []
nat2xnats k n | n>0 = xhd k n : nat2xnats k (xtl k n)

xnats2nat :: N→[N]→N
xnats2nat _ [] = 0
xnats2nat k (x:xs) = xcons k (x,xnats2nat k xs)
```

PROPOSITION 3. *The function* `nat2xnats` *defines a bijection from $\mathbb{N}$ to $[\mathbb{N}]$ and* `xnats2nat` *is its inverse.*

Proof The follows from the reversibility of the `cons` and `decons` functions.

The following example illustrates how they work:

```
*Primes> nat2xnats 3 2014
[0,0,1,2,0,2]
*Primes> xnats2nat 3 it
2014

*Primes> xnats2nat 2 [0,10,100]
519229685853482762853049632922145
*Primes> nat2xnats 2 it
[0,10,100]
```

As the second example illustrates, this bijection is not size-proportional as values on the $[\mathbb{N}]$ side result in exponentially larger representations on the $\mathbb{N}$ side.

2.4 A bijection between finite multisets and lists of natural numbers

Multisets [10] are collections with repeated elements.

Non-decreasing sequences provide a canonical representation for multisets of natural numbers. While finite multisets and finite lists of elements of $\mathbb{N}$ share a common representation `[N]`, multisets are subject to the implicit constraint that their ordering is immaterial. This suggest that a multiset like $[4, 4, 1, 3, 3, 3]$ could be represented canonically as sequence by first ordering it as $[1, 3, 3, 3, 4, 4]$ and then computing the differences between consecutive elements i.e. $[x_0, x_1 \ldots x_i, x_{i+1} \ldots] \rightarrow [x_0, x_1-x_0, \ldots x_{i+1}-x_i \ldots]$. This gives $[1, 2, 0, 0, 1, 0]$, with the first element 1 followed by the increments $[2, 0, 0, 1, 0]$, as implemented by `mset2list`:

```
mset2list xs = zipWith (-) (xs) (0:xs)
```

It is now clear that incremental sums of the numbers in such a sequence return the original multiset as implemented by `list2mset`:

```
list2mset ns = tail (scanl (+) 0 ns)
```

Note that a canonical representation (i.e. being sorted) is assumed for multisets.

The bijection between finite multisets and finite sequences in $\mathbb{N}$ defined by the functions `mset2list` and `list2mset`, is illustrated by the following example:

```
*Primes> mset2list [1,3,3,3,4,4]
[1,2,0,0,1,0]
*Primes> list2mset [1,2,0,0,1,0]
[1,3,3,3,4,4]
```

3. Encoding finite multisets with primes

The mapping between finite multisets and primes described in this section goes back to Gödel's arithmetic encoding of formulas [11, 12].

A factorization of a natural number is uniquely described as a multiset of primes. Moreover, we can use the fact that each prime number is uniquely associated to its *position* in the infinite stream of primes, to obtain a bijection from multisets of natural numbers to natural numbers. It is provided by the function `to_prime_positions` defined in Appendix. The function `nat2pmset` maps a natural number to the multiset of prime positions in its factorization.

```
nat2pmset 1 = []
nat2pmset n | n>1  = map succ (to_prime_positions n)
```

Clearly the following holds:

PROPOSITION 4. *p is prime if and only if its decomposition into a multiset given by* `nat2pmset` *is a singleton.*

The function `pmset2nat` (relying on `from_pos_in` and `primes` defined in Appendix) maps back a multiset of positions of primes to the result of the product of the corresponding primes.

```
pmset2nat [] = 1
pmset2nat ns = from_prime_positions (map pred ns)
```

By using the bijection between lists and multisets we obtain:

```
nat2plist = mset2list . map pred. nat2pmset . succ
plist2nat = pred . pmset2nat . map succ . list2mset
```

The operations `nat2pmset` and `pmset2nat` form a bijection between $\mathbb{N}$ and $[\mathbb{N}^+]$ working as follows:

```
*Primes> nat2pmset 2014
[1,8,16]
*Primes> pmset2nat it
2014

*Primes> nat2plist 2014
[2,3,5]
*Primes> plist2nat it
2014
```

For instance, as the factorization of `2014` is `2*19*53`, the list `[2,8,16]` contains the *positions* of the factors, starting from position `1` of the factor `2`, in the sequence of primes.

Note that the encoding of the prime factors of a number as a multiset of prime positions in the sequence of primes rather than the multiset of the primes themselves is significantly more compact, and the list encoding is even more so. In section 6 we will study in detail optimal information-theoretical encodings derived from factorization, as well as from alternative bijective representations.

4. A generic framework for studying primality through $\mathbb{N} \to [N]$ bijections

To facilitate the comparison of properties of derived from bijections between factors of prime numbers and alternative bijections between $\mathbb{N}$ and finite multisets and sequences of elements of $\mathbb{N}$, we will use a Haskell type class for sharing their common structure.

4.1 The type class defining our framework

Parameterized by the bijections `mix`: $[\mathbb{N}] \to \mathbb{N}$ and `unmix`: $\mathbb{N} \to [\mathbb{N}]$, as well as the bookkeeping operations (`lift` and `unlift`), we design a generic framework that expresses, in terms of these operations, isomorphisms between operations on natural numbers and multisets of natural numbers. We implement the framework as the Haskell type class `Converter`, whose instances will provide a playground for exploring emulations of interesting properties of prime numbers.

```
class Converter m where
  lift :: N→m N
  unlift :: m N → N

  mix :: [m N]→m N
  unmix :: m N→[m N]
```

The polymorphic type parameter `m` will be instantiated to Haskell data types providing alternative implementations of the framework. The following functions will be used to "lift" and "unlift" bijections from `N` and `[N]` as needed.

First we define two generic map functions converting between lifted and unlifted lists of natural numbers.

```
  unlifts :: [m N] → [N]
  unlifts = map unlift

  lifts :: [N] → [m N]
  lifts = map lift
```

Next we define combinators that lift *mixing* and *unmixing* functions (transforming between $\mathbb{N}$ and $[\mathbb{N}]$) to our data types.

```
  mixWith :: ([N]→N)→([m N]→m N)
  mixWith f = lift . f . unlifts

  unmixWith :: (N→[N])→(m N→[m N])
  unmixWith f = lifts . f . unlift
```

The combinators `liftFun` and `liftFuns` apply operations on $\mathbb{N}$ and $[\mathbb{N}]$ to their lifted counterparts.

```
  liftFun:: (N→N) → m N→ m N
  liftFun f = lift.f.unlift

  liftFuns:: ([N]→[N]) → [m N]→ [m N]
  liftFuns f = lifts.f.unlifts
```

After defining a generic list to multiset converter (note that a multiset decomposition is based on elements of $\mathbb{N}+$ rather than N).

```
  to_xmset :: m N → [m N]
  to_xmset = liftFuns (map succ . list2mset) . unmix

  from_xmset :: [m N] → m N
  from_xmset = mix . liftFuns (mset2list . map pred)
```

we can provide a generic "multiset prime" *recognizer* (`is_xprime`) and a *generator* operation (xprimes_from). They use the combinator `liftFun` to apply the successor and predecessor operations `succ` and `pred` to shift natural numbers between $\mathbb{N}$ and $\mathbb{N}^+$.

```
  is_xprime :: m N → Bool
  is_xprime x = f xs where
    xs = to_xmset (liftFun pred x)
    f [p] = True
    f _ = False

  xprimes_from :: m N→[m N]
  xprimes_from x =  filter is_xprime
    (iterate (liftFun succ) x)
```

We can also define a generic bijection from *positions* in the stream of "primes" to the stream of "primes" (`from_xindices`), as well as a simple "factorization" operation, `to_xprimes`.

```
  from_xindices :: [m N]→m N
  from_xindices = (liftFun succ) . from_xmset

  to_xindices :: m N→[m N]
  to_xindices  = to_xmset . (liftFun pred)

  to_xfactors :: m N→[m N]
  to_xfactors x = map i2f (to_xindices x) where
    ps = xprimes_from (lift 1)
    i2f i = ps ‘genericIndex‘ (pred (unlift i))
```

4.2 The instance derived for primes

The `Converter` instance `P` describes the actual primes in our framework. It is based on the functions `plist2nat` and `nat2plist` that use the functions `nat2pmset` `pmset2nat` providing the view of natural numbers as multisets of their prime factors.

```
data P a = P a deriving (Show, Read, Eq, Ord)

instance Converter P where
   mix  = mixWith plist2nat
   unmix = unmixWith nat2plist

   lift n = P n
   unlift (P n) = n
```

4.3 The instances derived for from the bijection `nat2nats`

The instances `A` and `B` encapsulate the prime emulations derived from the functions `nats2nat` and `nat2nats`

```
data A a = A a deriving (Show, Read, Eq, Ord)

instance Converter A where
   mix  = mixWith (nats2nat 2)

   unmix  = unmixWith (nat2nats 2)

   lift n = A n
   unlift (A n) = n
```

```
data B a = B a deriving (Show, Read, Eq, Ord)

instance Converter B where
   mix  = mixWith (nats2nat 3)

   unmix  = unmixWith (nat2nats 3)

   lift n = B n
   unlift (B n) = n
```

4.4 The instances derived for from the bijection `nat2xnats`

The instance X encapsulates the prime emulation based on $\nu_2(n)$.

```
data X a = X a deriving (Show, Read, Eq, Ord)
instance Converter X where
  lift n = X n
  unlift (X n) = n

  mix = mixWith (xnats2nat 2)
  unmix = unmixWith (nat2xnats 2)
```

A similar instance, Y based on $\nu_3(n)$ is defined as follows:

```
data Y a = Y a deriving (Show, Read, Eq, Ord)
instance Converter Y where
  lift n = Y n
  unlift (Y n) = n

  mix = mixWith (xnats2nat 3)
  unmix = unmixWith (nat2xnats 3)
```

4.5 Examples comparing the instances

The following examples illustrate the flexibility of the "plug-in" mechanism that this framework provides, when working with actual and emulated factorizations.

```
to_xfactors
*Primes> take 10 (xprimes_from (P 1))
[P 2,P 3,P 5,P 7,P 11,P 13,P 17,P 19,P 23,P 29]
*Primes> take 10 (xprimes_from (A 1))
[A 2,A 3,A 4,A 6,A 7,A 9,A 10,A 15,A 16,A 18]
*Primes> take 10 (xprimes_from (B 1))
[B 2,B 3,B 4,B 5,B 7,B 8,B 9,B 11,B 12,B 13]
*Primes> take 10 (xprimes_from (X 1))
[X 2,X 3,X 5,X 9,X 17,X 33,X 65,X 129,X 257,X 513]
*Primes> take 10 (xprimes_from (Y 1))
[Y 2,Y 4,Y 10,Y 28,Y 82,Y 244,Y 730,Y 2188,Y 6562,Y 19684]
```

One can also observe, that the following holds:

PROPOSITION 5. *There's an infinite number of* multiset primes *in each type of the family $X, Y, \ldots$ and they are exactly the numbers of the form $b^n + 1$.*

The generic equivalent of *factorization* that matches its expected behavior on the instance P representing the usual primes, is illustrated by the following examples:

```
*Primes> to_xfactors (P 36)
[P 2,P 2,P 3,P 3]
*Primes> to_xfactors (P 37)
[P 37]
*Primes> to_xfactors (A 23)
[A 2,A 2,A 3]
*Primes> to_xfactors (A 29)
[A 2,A 10]
*Primes> to_xfactors (X 32)
[X 2,X 2,X 2,X 2,X 2]
*Primes> to_xfactors (X 33)
[X 33]
```

In [8] we provide a detailed comparison in terms of two specific instances of multisets: those corresponding here to data type X and the primes, corresponding here to data type P.

5. A Catalan connection

The recursive application of the `unmix` and `mix` functions can be used to unfold a natural number to a tree containing successive layers of canonical decomposition as sequences. The ordered rooted trees of type `CTree`, with empty leaves will be used host the result of this decomposition.

```
data CTree = C [CTree] deriving (Show,Read,Eq)
```

We encapsulate the unfolding operation `toTree` and its folding counterpart `fromTree` in the type class `CatalanView`, as this tree representation is a typical member of the *Catalan family* [13] of combinatorial structures.

```
class Converter m ⇒ CatalanView m where
  toTree :: m N → CTree
  toTree mn = C (map toTree (unmix mn))

  fromTree :: CTree → m N
  fromTree (C xs) = mix (map fromTree xs)
```

The function `morphTree` provides, generically, a transformation between two such tree views, provided by two instances of the type class `CatalanView`.

```
  morphTree :: (CatalanView m') ⇒ m' N → m N
  morphTree  = fromTree . toTree
```

The function `tsize` computes the size of a tree.

```
  tsize :: (N→m N)→N→ N
  tsize t n = ts (toTree (t n)) where
   ts (C []) = 1
   ts (C xs) = succ (sum (map ts xs))
```

After adding the usual instances

```
instance CatalanView P
instance CatalanView A
instance CatalanView B
instance CatalanView X
instance CatalanView Y
```

we can define the functions `p, a, b, x, y` corresponding, respectively, to target types `P, A, B, X, Y`.

```
p n = morphTree n :: P N
a n = morphTree n :: A N
b n = morphTree n :: B N
x n = morphTree n :: X N
y n = morphTree n :: Y N
```

The following examples illustrate the use of these functions.

```
*Primes> toTree (P 2014)
C [C [C [C []],C [C [],C []],C [C [],C [C []]]]
*Primes> toTree (A 2014)
C [C [],C [C [C []],C []],C [C [C [C [C []]]]]]
*Primes> fromTree it :: P N
P 2014
*Primes> toTree (X 2014)
C [C [C []],C [],C [],C [],C [C []],C [],C [],C [],C []]
```

Figures 1, 2 and 3 illustrate a DAG representation (with sharing of identical nodes) of the trees generated by `toTree` on types `A`, `P` and `X`.

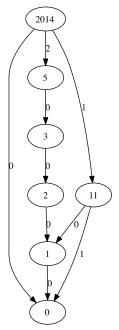

Figure 1: DAG corresponding to unfolding 2014 through data type A

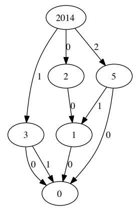

Figure 2: DAG corresponding to unfolding 2014 through data type P

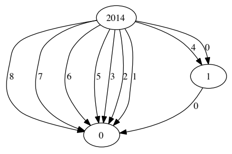

Figure 3: DAG corresponding to unfolding 2014 through data type X

In combination with `unlift` the function a, b, p, x, y can be used to define (infinite) permutations of $\mathbb{N}$.

```
*Primes> map (unlift.p.A) [0..15]
[0,1,2,4,3,10,6,5,30,16,9,8,24,7,12,126]
*Primes> map (unlift.p.B) [0..15]
[0,1,2,4,10,3,30,6,126,5,16,708,12,9,58,5380]
*Primes> map (unlift.p.X) [0..15]
[0,1,2,3,4,5,8,7,6,9,14,11,24,17,26,15]
*Primes> map (unlift.p.Y) [0..15]
[0,1,3,2,7,5,8,15,11,6,17,31,26,23,13,14]
```

Intuitively, these permutations of $\mathbb{N}$ amplify the "imperfection" of the emulation of, in this case prime factorization, by the other instances of the class `Converter`, as shown in figures 4 and 5.

6. Measuring the information loss through factorization

In [5] a comprehensive information theoretic analysis o f multiplication is studied, in terms of the information content of a number vs. its factors (and in particular its prime factors).

As prime factors contain together more bits than their product, multiplication results in general in *loss of information*. However,

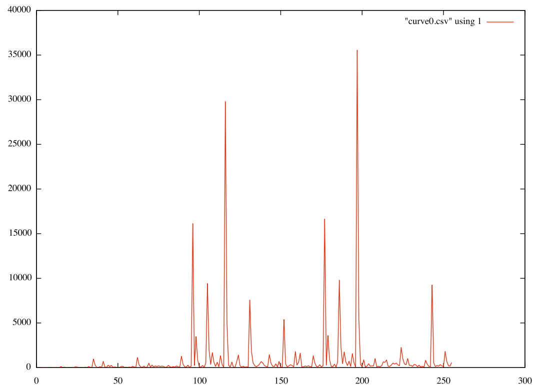

Figure 4: Permutation of $\mathbb{N}$ obtained by morphing between `P` and `A` views on $[0..2^8 - 1]$

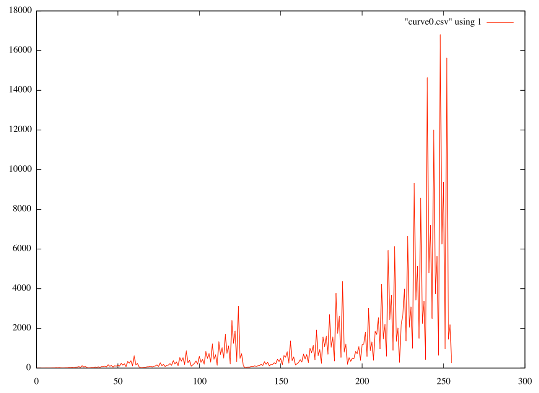

Figure 5: Permutation of $\mathbb{N}$ obtained by morphing between `P` and `X` views on $[0..2^8 - 1]$

if one considers the *positions in the stream of primes* as a basis for canonical representations of factorization, we will see that this situation is *reversed*.

6.1 Multiplication, generically

Multiplication corresponds to sorted concatenation of the multisets of factors, illustrated below:

```
*Primes> 12*15
180
*Primes> to_xfactors (P 12)
[P 2,P 2,P 3]
*Primes> to_xfactors (P 15)
[P 3,P 5]
*Primes> to_xfactors (P 180)
[P 2,P 2,P 3,P 3,P 5]
```

As shown in [8] one can also work with the the indices in the multiset of primes directly. We define multiplication in new type class `ArithOps` that can be extended, along the lines of [8], with various other operations.

```
class (Ord (m N),Converter m)⇒ArithOps m where
  multiply :: (m N)→(m N)→(m N)
  multiply x y = from_xindices
    (sort (to_xindices x ++ to_xindices y))
```

The following holds:

PROPOSITION 6. `Multiply` *defines semigroup with unit* `P 1` *on data type* `P` *(corresponding to* $\mathbb{N}^+$*). The same structure is defined on data type* `A,B,X,Y` *and their generalizations for an arbitrary* `k`.

After adding the usual instances

```
instance ArithOps P
instance ArithOps A
instance ArithOps B
instance ArithOps X
instance ArithOps Y
```

one can observe that `multiply` works as expected on type `P` and it has similar basic properties on its alternatives.

```
*Primes> multiply (P 10) (P 5)
P 50
*Primes> multiply (P 5) (P 10)
P 50
*Primes> multiply (P 5) (P 1)
P 5
*Primes> multiply (A 10) (A 5)
A 86
*Primes> multiply (A 5) (A 10)
A 86
*Primes> multiply (A 5) (A 1)
A 5
*Primes> multiply (X 10) (X 5)
X 26
*Primes> multiply (X 5) (X 10)
X 26
*Primes> multiply (X 5) (X 1)
X 5
```

DEFINITION 4. *We call* canonical factorization *the representation provided by our* `unmix` *bijection from* $\mathbb{N}$ *to* $[\mathbb{N}]$*, for each of the instances of the type class* `Converter`.

It follows from their bijection to arbitrary finite sequences in $[\mathbb{N}]$ that canonical factorization is information theoretically minimal.

In contrast to [5], where it is shown that information is lost by multiplication when considering the actual factors (and primes in particular), we will show that, when considering our *canonical factorization*, it is the other way around.

6.2 Computing the amount of information lost by factorization

After defining the representation size for bijective base-`b`

```
repsize b n = genericLength (toBijBase b n)

bitsize = repsize 2
```

we introduce the new type class `Comparator`, to facilitate comparison between the information content corresponding to bijections between $\mathbb{N}$ and $[\mathbb{N}]$, as well as their multiset counterparts.

```
class (Converter m,Ord (m N))⇒Comparator m where
  infoLoss :: (N→m N)→N→N→N
  infoLoss t k n = f (unlifts.unmix.t) n where
    f u n = (repsize k n) - s where
      ns = u n
      s = sum (map (repsize k) ns)
```

The function `infoLoss` computes the difference between the representation sizes of a natural number and its expansion to a canonical list representation. In particular for $k = 2$ it computes the difference between their bitsizes in bijective base-2.

```
  totalInfoLoss:: (N→m N)→N→N
  totalInfoLoss t n = sum (map (infoLoss t 2) [0..2^n-1])

  prefixSumLoss:: (N→m N)→N→[N]
  prefixSumLoss t n =
    scanl1 (+) (map (infoLoss t 2) [0..2^n-1])
```

The functions `totalInfoLoss` and `prefixSumLoss` count respectively the total information loss and its prefix sums on the interval $[0..2^n-1]$.

After adding the instances

```
instance Comparator P
instance Comparator A
instance Comparator B
instance Comparator X
instance Comparator Y
```

one can observe that that information loss is positive for each of our instances.

```
*Primes> map (infoLoss A 2) [0..15]
[0,1,0,1,2,0,0,2,1,1,2,2,2,3,0,1]
*Primes> map (infoLoss B 2) [0..15]
[0,1,0,1,0,2,0,1,1,2,0,0,0,2,0,1]
*Primes> map (infoLoss P 2) [0..15]
[0,1,0,2,1,1,0,3,2,2,1,2,1,1,1,4]
*Primes> map (infoLoss X 2) [0..15]
[0,1,0,2,1,1,1,3,1,2,1,2,2,2,2,4]
*Primes> map (infoLoss Y 2) [0..15]
[0,1,1,1,2,1,1,3,2,2,2,3,2,2,2,2]
```

Note however, that for larger bases that does not hold for every n:

```
*Primes> infoLoss P 10 104
-1
*Primes> infoLoss A 20 20
-1
*Primes> infoLoss X 10 10
-1
```

It is easy to prove that `infoLoss 2` is always positive in the case of the family of instances `A`, `B` ... as the list computed by `unmix` is obtained by removing a digit from their bijective base-2 representation.

It is also likely to be easy to prove that the same holds for instances `X`, `Y` ... where the left side of the bijection is exponentially larger than the right side.

The case for instance `P` is more intricate (but also more interesting), and one might first suspect that the the proof would involve deep properties of the primes.

We state this claim as:

CONJECTURE 1. *The function* `infoLoss 2` *computing the difference of bitsizes of* n *and the sum of the bitsizes of the canonical factors of* n*, for instances* `P,A,B,...,X,Y,...` *is positive for all* $n \in \mathbb{N}$.

After observing that `bitsize x` is the same as $b(x) = \lfloor log_2(x+1) \rfloor$ we can restate this conjecture for instance `P` in terms of standard arithmetic notations (and prove it), as follows:

PROPOSITION 7. *For* $n \in \mathbb{N}$*, let* $n+1 = p_0^{j_0} \ldots, p_i^{j_i}$*, where* $p_0, \ldots, p_i$ *are the prime factors of* $n+1$ *in increasing order. Let* $[q_0, \ldots, q_m]$ *be the list of the positions in the sequence of primes starting with* 2 *of these factors with their respective multiplicities. Let* $[q_1 - q_0, \ldots, q_m - q_{m-1}]$ *be the list of their consecutive differences, clearly all* ≥ 0*. Let* $b(x) = \lfloor log_2(x+1) \rfloor$.

Then $b(n) \geq \sum_{l=1}^{m} b(q_l - q_{l-1})$.

Proof For $n = 0$ equality holds, the list on the right (factorization of 1) being empty. For $n > 0$ observe that $\sum_{l=1}^{m} b(q_l - q_{l-1}) = \sum_{l=1}^{m} \lfloor log_2(1 + q_l - q_{l-1}) \rfloor \leq \lfloor \sum_{l=1}^{m} log_2(1 + q_l - q_{l-1}) \rfloor \leq \lfloor \sum_{l=1}^{m} log_2(q_l) \rfloor = \lfloor log_2(\prod_{l=1}^{m} q_i) \rfloor \leq \lfloor log_2(\prod_{l=0}^{m} p_i) \rfloor = \lfloor log_2(n+1) \rfloor = b(n)$

Observe that no "deep properties" of primes are involved, and the proof follows from a sequence of rather obvious inequalities.

Note that total information losses are comparable for all data types, `P` being closest to `B`.

```
*Primes> totalInfoLoss A 10
4379
*Primes> totalInfoLoss B 10
2987
*Primes> totalInfoLoss P 10
3965
*Primes> totalInfoLoss X 10
5447
*Primes> totalInfoLoss Y 10
5920
```

Figures 6 7 and 8 illustrate the information lost by canonical factorization representations for instances `P`,`A` and `X`.

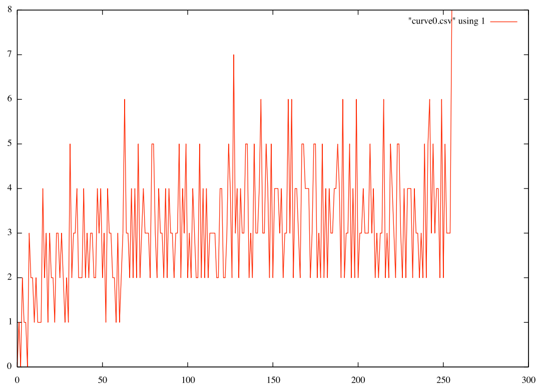

Figure 6: Information loss for `P` on $[0..2^8 - 1]$

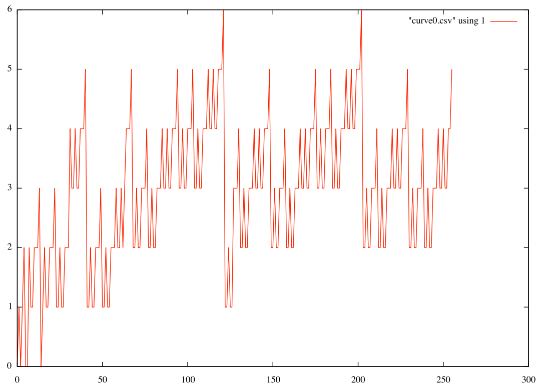

Figure 7: Information loss for `A` on $[0..2^8 - 1]$

figures 9 and 10 compare prefix sums for the information lost by canonical factorization representations for type `P` and the instances matching it closely on initial segments of $\mathbb{N}$.

7. Emulating the behavior of w, Ω and b-smooth numbers

DEFINITION 5. *The function $\omega(x)$ counts the number of distinct prime factors of x. The function $\Omega(x)$ counts the number of not necessarily distinct prime factors of x.*

We will implement generic versions of these two functions parameterized by an arbitrary multiset representation `t`. Note that Haskell's `nub` function is used in function `omega` to remove duplicates from a multiset.

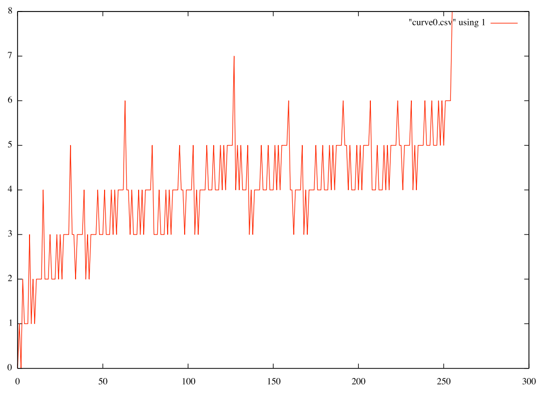

Figure 8: Information loss for `X` on $[0..2^8 - 1]$

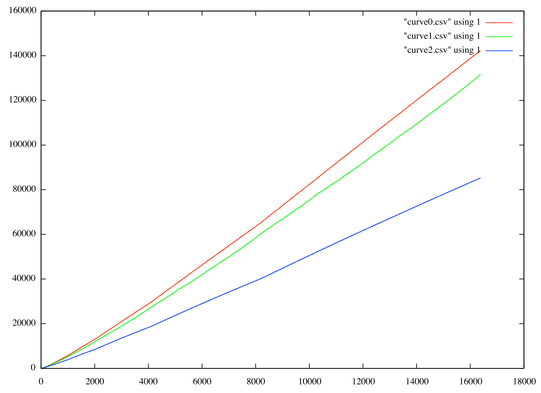

Figure 9: Prefix sum of information loss for `Y=red/upper`, `X=yellow/middle` and `P=blue/lower` on $[1..2^{14}]$

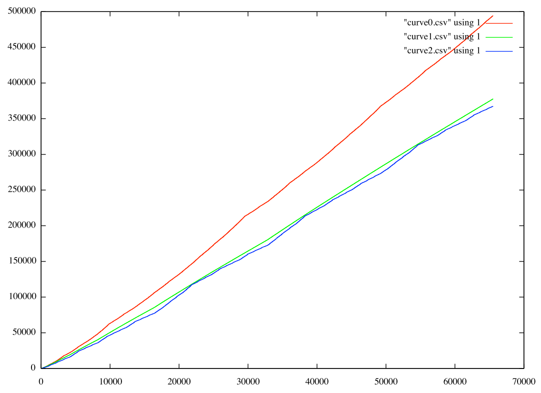

Figure 10: Prefix sum of information loss for `A=red/upper`, `P=yellow/middle` and `B=blue/lower` on $[1..2^{16}]$

```
bigomega :: Converter m ⇒ (N → m N) → N → [Int]
bigomega t m = map  (length . to_xmset . t) [1..2^m]
omega t m = map  (length . nub . to_xmset . t) [1..2^m]
```

To compare them for various values of `t` we will use their prefix sums up to a power of 2, defined as follows:

```
bigomega_sum :: Converter m ⇒ (N → m N) → N → [Int]
bigomega_sum t m = scanl (+) 0 (bigomega t m)
omega_sum t m = scanl (+) 0 (omega t m)
```

Figures 11 and 12 show that ω and Ω have a similar behavior on an initial segment of $\mathbb{N}$ for some of our data types. Note that values for `A,B` constrain in Fig. 11 from above and below the values of `P` while in Fig. 12 `P` is followed closely from below by `A`.

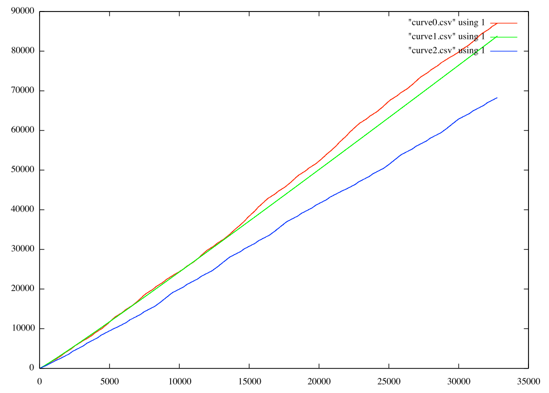

Figure 11: Sum of the ω function for `A=red/upper`, `P=yellow/middle` and `B=blue/lower` on $[1..2^{15}]$

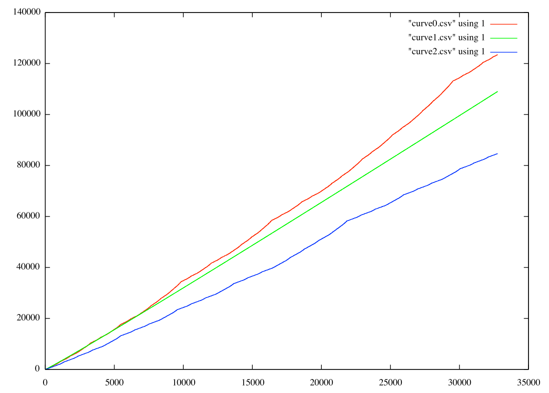

Figure 12: Sum of the Ω function for `P=red/upper`, `A=yellow/middle` and `X=blue/lower` on $[1..2^{15}]$

DEFINITION 6. *A natural number $n \in \mathbb{N}$ is b-smooth, if all its prime factors are smaller or equal than b.*

The following functions provide generic definitions of b-smoothness for arbitrary instances of our `Converter` class:

```
isBsmooth :: Comparator m ⇒ m N → m N → Bool
isBsmooth b n = []==filter (>b) (to_xfactors n)

smooth_set :: Comparator m ⇒ m N → N → [N]
smooth_set b m =
  unlifts (filter (isBsmooth b) (lifts [2..2^m]))
```

Figure 13 shows the distribution of 5-smooth numbers for `B, P` and `Y`. Note that the instance `P`, is constrained from above by `B` and from below by `Y`.

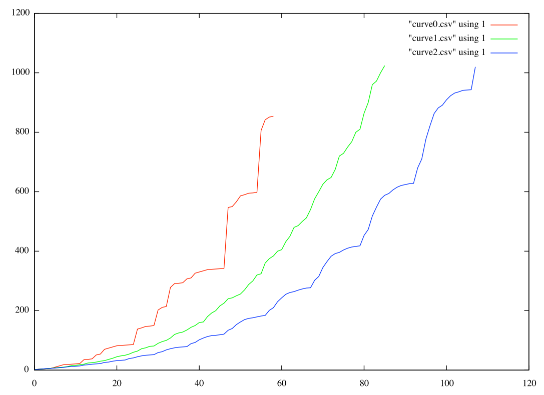

Figure 13: Distribution of 5-smooth numbers for B=red/upper, P=yellow/middle and Y=blue/lower

The study of b-smooth numbers is interesting given their applications to factorization algorithms and cryptography. The fact that it can be defined for alternative data types like `A, B, X, Y` and not just for primes, and that similar behaviors are observed, might be useful in this context.

8. Related work

There's a huge amount of work on prime numbers and related aspects of multiplicative and additive number theory. Studies of prime number distribution and various probabilistic and information theoretic aspects also abound.

While we have not made use of any significantly advanced facts about prime numbers, the following references circumscribe the main topics to which our experiments can be connected [3–5, 7, 14].

In combinatorics, natural number encodings show up as *ranking* functions [15] that can be traced back to Gödel numberings [11, 12] associated to formulas. Together with their inverse *unranking* functions, they are also used in combinatorial generation algorithms for various data types [13].

In [8] we provide a detailed comparison in terms of the specific instance of multisets (those corresponding here to data type `X`) and primes (corresponding here to data type `P`), with focus on basic operations like product, gcd, etc. as well as in terms of more interesting concepts like the `rad` and `Mertens` functions.

9. Conclusion

We have explored some computational analogies between multisets, natural number encodings and prime numbers in a framework for experimental mathematics implemented as a literate Haskell program.

We have lifted our Haskell implementations to a generic type class based model, along the lines of [16], which allows experimenting with instances parameterized by arbitrary bijections between $\mathbb{N}$ and $[\mathbb{N}]$.

A concept of canonical factorization, representing the decomposition of a number in a generic way has been introduced and its recursive application has provided a tree/DAG view uniquely associated to a natural number, through the corresponding member of the Catalan family of combinatorial structures.

Interesting correlations have been found between information losses due to factorization between primes and the other instances of our generic framework.

As an object of future work, of special interest in this direction are multiset decompositions of a natural number in $O(log(log(n)))$ factors, similar to the $\omega(x)$ and $\Omega(x)$ functions counting the distinct and non-distinct prime factors of x, to mimic more closely the distribution of primes.

Acknowledgment

We thank NSF (research grant 1018172) for support.

References

[1] B. Riemann, Ueber die anzahl der primzahlen unter einer gegebenen grösse, Monatsberichte der Berliner Akademie.

[2] G. L. Miller, Riemann's hypothesis and tests for primality, in: STOC, ACM, 1975, pp. 234–239.
URL http://dblp.uni-trier.de/db/conf/stoc/stoc75.html#Miller75

[3] J. C. Lagarias, An Elementary Problem Equivalent to the Riemann Hypothesis, The American Mathematical Monthly 109 (6) (2002) pp. 534–543.

[4] B. Conrey, The Riemann Hypothesis, Not. Amer. Math. Soc. 60 (2003) 341–353.

[5] N. Pippenger, The average amount of information lost in multiplication, IEEE Transactions on Information Theory 51 (2) (2005) 684–687.

[6] P. Cégielski, D. Richard, M. Vsemirnov, On the additive theory of prime numbers., Fundam. Inform. 81 (1-3) (2007) 83–96.
URL http://dblp.uni-trier.de/db/journals/fuin/fuin81.html#CegielskiRV07

[7] R. Crandall, C. Pomerance, Prime Numbers–a Computational Approach, 2nd Edition, Springer, New York, 2005.

[8] P. Tarau, Emulating Primality with Multiset representations of Natural Numbers, in: A. Cerone, P. Pihlajasaari (Eds.), Proceedings of 8th International Colloquium on Theoretical Aspects of Computing, ICTAC 2011, Springer, LNCS 6916, Johannesburg, South Africa, 2011, pp. 218–238.

[9] Wikipedia, Bijective numeration — wikipedia, the free encyclopedia, [Online; accessed 2-June-2012] (2012).
URL http://en.wikipedia.org/w/index.php?title=Bijective_numeration&oldid=483462536

[10] D. Singh, A. M. Ibrahim, T. Yohanna, J. N. Singh, An overview of the applications of multisets, Novi Sad J. Math 52 (2) (2007) 73–92.

[11] K. Gödel, Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I, Monatshefte für Mathematik und Physik 38 (1931) 173–198.

[12] J. Hartmanis, T. P. Baker, On Simple Goedel Numberings and Translations., in: J. Loeckx (Ed.), ICALP, Vol. 14 of Lecture Notes in Computer Science, Springer, Berlin Heidelberg, 1974, pp. 301–316.
URL http://dblp.uni-trier.de/db/conf/icalp/icalp74.html#HartmanisB74

[13] D. L. Kreher, D. Stinson, Combinatorial Algorithms: Generation, Enumeration, and Search, The CRC Press Series on Discrete Mathematics and its Applications, CRC PressINC, 1999.
URL http://books.google.com/books?id=0ArDOdcWNQcC

[14] J. Derbyshire, Prime Obsession: Bernhard Riemann and the Greatest Unsolved Problem in Mathematics, Penguin, New York, 2004.

[15] C. Martinez, X. Molinero, Generic algorithms for the generation of combinatorial objects., in: B. Rovan, P. Vojtas (Eds.), MFCS, Vol. 2747 of Lecture Notes in Computer Science, Springer, Berlin Heidelberg, 2003, pp. 572–581.

[16] P. Tarau, Declarative modeling of finite mathematics, in: PPDP '10: Proceedings of the 12th international ACM SIGPLAN symposium on Principles and practice of declarative programming, ACM, New York, NY, USA, 2010, pp. 131–142.

Appendix

Primes

The following code implements a factorization function (`to_primes`), a primality test (`is_prime`) and a generator for the infinite stream of `primes`.

```
primes = 2 : filter is_prime [3,5..]

is_prime p = [p]==to_primes p

to_primes n|n>1 =
  to_factors n p ps where (p:ps) = primes

to_factors n p ps | p*p > n = [n]
to_factors n p ps | 0==n `mod` p =
  p : to_factors (n `div` p)  p ps
to_factors n p ps@(hd:tl) = to_factors n hd tl

to_prime_positions n | n>1 =
  map (to_pos_in (h:ps)) qs where
    (h:ps)=genericTake n primes
    qs=to_factors n h ps

from_prime_positions ns = product
  (map  (from_pos_in primes) ns)

to_pos_in xs x = fromIntegral i where
  Just i=elemIndex x xs

from_pos_in xs n = genericIndex xs n
```