Emulating Primality with Multiset Representations of Natural Numbers

Paul Tarau

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

Abstract. Factorization results in multisets of primes and this mapping can be turned into a bijection between multisets of natural numbers and natural numbers. At the same time, simpler and more efficient bijections exist that share some interesting properties with the bijection derived from factorization.

This paper describes mechanisms to emulate properties of *prime numbers* through isomorphisms connecting them to computationally simpler representations involving bijections from natural numbers to *multisets* of natural numbers.

As a result, interesting automorphisms of $\mathbb{N}$ and emulations of the rad, Möbius and Mertens functions emerge in the world of our much simpler multiset representations.

The paper is organized as a self-contained *literate Haskell program*. The code extracted from the paper is available as a standalone program at `http://logic.cse.unt.edu/tarau/research/2011/mprimes.hs`.

Keywords: *bijective datatype transformations, multiset encodings and prime numbers, Möbius and Mertens functions, experimental mathematics and functional programming, automorphisms of* $\mathbb{N}$

1 Introduction

Paul Erdös's statement, shortly before he died, that *"It will be another million years at least, before we understand the primes"* is indicative of the difficulty of the field as perceived by number theorists. The growing number of conjectures [1] and the large number of still unsolved problems involving prime numbers [2] shows that the field is still open to surprises, after thousands of years of effort by some of the brightest human minds.

Interestingly, some significant progress on prime numbers correlates with unexpected paradigm shifts, the prototypical example being Riemann's paper [3] connecting primality and complex analysis, all evolving around the still unsolved *Riemann Hypothesis* [4–7]. The genuine difficulty of the problems and the seemingly deeper and deeper connections with fields ranging from cryptography to quantum physics suggest that unusual venues might be worth trying out.

A number of breakthroughs in various sciences involve small scale emulation of complex phenomena. Common sense analogies thrive on our ability to extrapolate from simpler (or, at least, more frequently occurring and better understood) mechanisms to infer surprising properties in a more distant ontology.

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 [8] and it suggests investigating connections between combinatorial properties of multisets and operations on multisets and multiplicative number theory.

With such methodological hints in mind, this paper will explore mappings between multiset encodings and prime numbers. It is based on our *data type transformation framework* connecting most of the fundamental data types used in computer science with a *groupoid of isomorphisms* [9–11].

The paper is organized as follows. Section 2 revisits the well-known connection between multisets and primes using a variant of Gödel's encoding [12]. Section 3 describes our computationally efficient multiset encoding. Based on these encodings, section 4 explores the analogy between multiset decompositions and factoring and describes a multiplicative monoid structure on multisets that "emulates" properties of the monoid induced by ordinary multiplication as well as generic definitions in terms of multiset encodings of the `rad`, Möbius and Mertens functions. Section 5 describes automorphisms of $\mathbb{N}$ derived from alternative multiset encodings. Section 6 overviews some related work and section 7 concludes the paper.

We organize our literate programming code as a Haskell module, relying only on the List library module:

```
module MPrimes where
import Data.List
```

2 Encoding finite multisets with primes

2.1 Ranking/unranking of sets and finite sequences

First, we define an isomorphism between sets and finite sequences (the Hub of the groupoid of isomorphisms) as described in [9] (also in the Appendix), resulting in the Encoder `set`:

```
set2list xs = shift_tail pred (mset2list xs) where
  shift_tail _ [] = []
  shift_tail f (x:xs) = x:(map f xs)

list2set = (map pred) . list2mset . (map succ)

set :: Encoder [N]
set = Iso set2list list2set
```

We can *rank/unrank* a set represented as a list of distinct natural numbers by observing that it can be seen as the list of exponents of 2 in the number's base 2 representation.

```
nat_set = Iso nat2set set2nat

nat2set n | n≥0 = nat2exps n 0 where
  nat2exps 0 _ = []
  nat2exps n x = if (even n) then xs else (x:xs) where
    xs=nat2exps (n `div` 2) (succ x)

set2nat ns = sum (map (2^) ns)
```

The resulting Encoder is:

```
nat :: Encoder N
nat = compose nat_set set
```

2.2 Encoding Multisets

Multisets [13] are unordered collections with repeated elements. Non-decreasing sequences provide a canonical representation for multisets of natural numbers.

The mapping between finite multisets and primes described in this section goes back to Gödel's arithmetic encoding of formulae [12, 14]. A factorization of a natural number is uniquely described as a multiset of primes. 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. This mapping is the same as the *prime counting function* traditionally denoted $\pi(n)$, which associates to n the number of primes smaller or equal to n, restricted to primes. 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 factoring[1].

```
nat2pmset 1 = []
nat2pmset n = to_prime_positions n
```

Clearly the following holds:

Proposition 1 *p is prime if and only if its decomposition in a multiset given by* `nat2pmset` *is a singleton.*

The function `pmset2nat` (relying on `from_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 = product (map (from_pos_in primes . pred) ns)
```

[1] In contrast to [9], we will assume that our mappings are defined on $\mathbb{N}^+ = \mathbb{N} - \{0\}$ rather than $\mathbb{N}$.

The operations `nat2pmset` and `pmset2nat` form an isomorphism that, using the combinator language defined in [9] (and summarized in the Appendix to ensure that this paper is fully self-contained) provides *any-to-any encodings* between various data types. This gives the Encoder `pmset` for prime encoded multisets as follows:

```
pmset :: Encoder [N]
pmset = compose (Iso pmset2nat nat2pmset) nat
```

working as follows:

```
*MPrimes> as pmset nat 2010
[1,2,3,19]
*MPrimes> as nat pmset [1,2,3,19]
2010
```

For instance, as the factoring of 2010 is $2 * 3 * 5 * 67$, the list [1,2,3,19] contains the positions of the factors, starting from 1, in the sequence of primes.

3 A bijection between finite multisets and natural numbers

We will now define *ranking/unranking* functions for multisets i.e. bijective mappings to/from natural numbers. While finite multisets and sequences representing finite functions 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 set as implemented by `list2mset`:

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

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

The isomorphism between finite multisets and finite functions (seen as finite sequences in $\mathbb{N}$) is specified with two bijections `mset2list` and `list2mset`.

```
mset0 :: Encoder [N]
mset0 = Iso mset2list list2mset
```

The resulting isomorphism `mset0` can be applied by using its two components `mset2list` and `list2mset` directly.

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

Equivalently, following [9] (see also summary in Appendix), it can be expressed generically by using the "as" combinator:

```
*MPrimes> as list mset0 [1,3,3,3,4,4]
[1,2,0,0,1,0]
*MPrimes> as mset0 list [1,2,0,0,1,0]
[1,3,3,3,4,4]
```

The combinator "as" derives automatically *any-to-any* encodings, by routing through the appropriate one-to-one transformations (see [9] and the `Appendix`). As a result, we obtain "for free" a bijection from $\mathbb{N}$ to finite multisets of elements of $\mathbb{N}$:

```
*MPrimes> as mset0 nat 2011
[0,0,1,1,2,2,2,2,2]
*MPrimes> as nat mset0 it
2011
```

We will need one small change to convert this into a mapping on $\mathbb{N}^+$.

```
nat2mset1 n=map succ (as mset0 nat (pred n))
mset2nat1 ns=succ (as nat mset0 (map pred ns))

mset :: Encoder [N]
mset = compose (Iso mset2nat1 nat2mset1) nat
```

The resulting mapping, like `pmset`, now works on $\mathbb{N}^+$.

```
*MPrimes> as mset nat 2012
[1,1,2,2,3,3,3,3,3]
*MPrimes> as nat mset it
2012
*MPrimes> map (as mset nat) [1..7]
[[],[1],[2],[1,1],[3],[1,2],[2,2]]
```

Note that these mappings work in linear time and space in the bitsize of the numbers [15]. On the other hand, as prime number enumeration and factoring are involved in the mapping from numbers to multisets, the encoding described in section 2 is intractable for all but small values.

4 Exploring the analogy between multiset decompositions and factoring

As natural numbers can be uniquely represented as multisets of prime factors and, independently, they can also be represented as a multiset with the Encoder `mset` (described in section 3), the following question arises naturally:

Can the computationally efficient encoding `mset` *help emulate some properties of the the difficult to reverse factoring operation?*

4.1 A multiset analog to multiplication

The first step is to define an analog of the multiplication operation in terms of the computationally easy multiset encoding `mset`. Clearly, it makes sense to take inspiration from the fact that factoring of an ordinary product of two numbers can be computed by *concatenating* the multisets of prime factors of its operands. We use the combinator `borrow_from` and the sorted concatenation `sortedConcat` operations (see Appendix) to express this:

```
mprod = borrow_from mset sortedConcat nat
```

Proposition 2 $\langle N^+, mprod, 1\rangle$ *is a commutative monoid i.e.* `mprod` *is defined for all pairs of natural numbers and it is associative, commutative and has 1 as an identity element.*

Proof. After rewriting the definition of `mprod` as the equivalent:

```
mprod_alt n m = as nat mset
  (sortedConcat (as mset nat n) (as mset nat m))
```

the proposition follows immediately from the associativity of the concatenation operation and the order independence of the multiset encoding provided by `mset`. □

Here are a few examples showing that `mprod` has properties similar to ordinary multiplication:

```
*MPrimes> mprod 41 (mprod 33 38) == mprod (mprod 41 33) 38
True
*MPrimes> mprod 33 46 == mprod 46 33
True
*MPrimes> mprod 1 712 == 712
True
```

Given the associativity of `mprod`, it makes sense to define the product of a list of numbers as

```
mproduct ns = foldl mprod 1 ns
```

Note also that any multiset encoding of natural numbers can be used to define a similar commutative monoid structure. In the case of `pmset` we obtain:

```
pmprod n m = as nat pmset
  (sortedConcat (as pmset nat n) (as pmset nat m))
```

This brings us back to observe that:

Proposition 3 $\langle N, pmprod, 1\rangle$ *is a commutative monoid i.e.* `pmprod` *is defined for all pairs of natural numbers and it is associative, commutative and has 1 as an identity element.*

Unsurprisingly, this is the case indeed as one can deduce immediately from the definition of `pmprod` that, on $\mathbb{N}^+$:

$$pmprod \equiv *\tag{1}$$

As obvious as this equivalence is, note that computing `*` is easy, while computing `pmprod` involves factoring which is intractable for large values.

Experimentally (including very large random integers), `mprod x y` $\leq$ `pmprod x y`, leading us to:

Conjecture 1 *mprod* x $y = x * y$ *if and only if* $\exists n \geq 0$ *such that* $x = 2^n$ *or* $y = 2^n$. *Otherwise, mprod* x $y < x * y$.

Fig. 1 shows the self-similar landscape generated by the function `(mprod x y) / (x*y)` for values of x,y in [1..128].

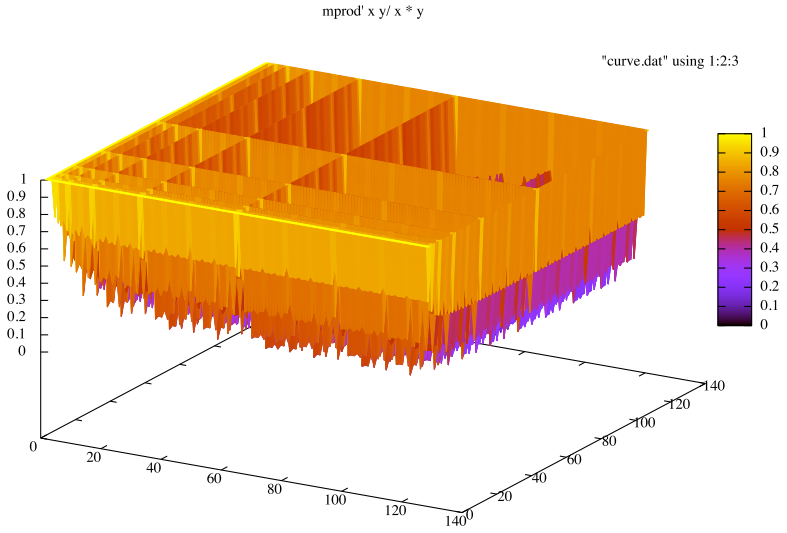


Fig. 1. Ratio between mprod and product

We can derive an exponentiation operation as a repeated application of `mprod`:

```
mexp n 1 = n
mexp n k = mprod n (mexp n (k-1))
```

Let us first observe that the ordinary exponent and our emulated variant correlate as follows:

```
*MPrimes> map (λx→mexp 2 x) [1..8]
[2,4,8,16,32,64,128,256]
*MPrimes> map (λx→2^x) [1..8]
[2,4,8,16,32,64,128,256]

*MPrimes> map (λx→mexp x 2) [1..16]
[1,4,7,16,13,28,31,64,25,52,55,112,61,124,127,256]
*MPrimes> map (λx→x^2) [1..16]
[1,4,9,16,25,36,49,64,81,100,121,144,169,196,225,256]
```

Fig. 2 shows that values for `mexp x 2` follow from below those of the x^2 function and that equality only holds when x is a power of 2.

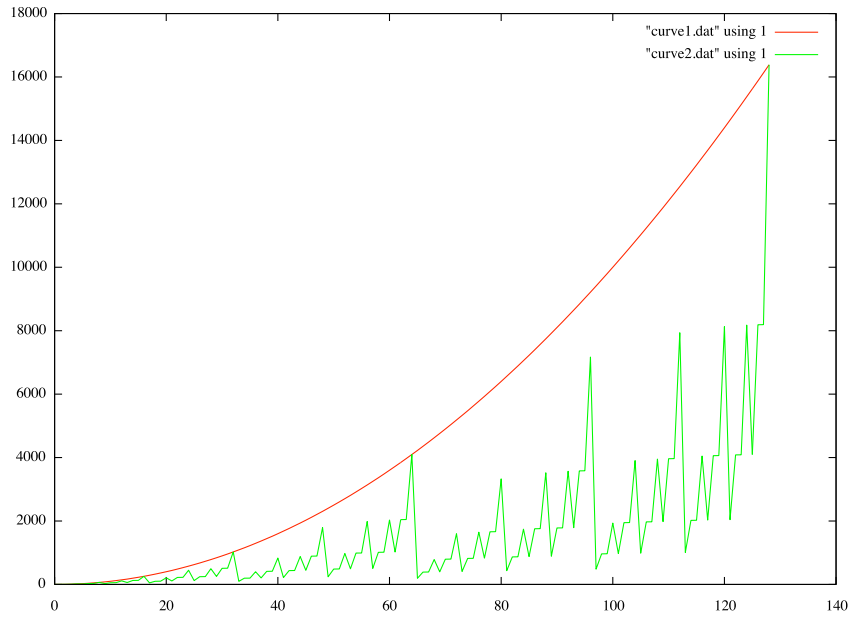


Fig. 2. Square vs. mexp n 2

4.2 Multiset analogues for div, gcd and lcd

Besides the connection with products, natural mappings worth investigating are the analogies between *multiset intersection* and `gcd` of the corresponding numbers or between *multiset union* and the `lcm` of the corresponding numbers. Assuming the definitions of multiset operations provided in the `Appendix`, one can define:

```
mgcd :: N → N → N
```

```
mgcd = borrow_from mset msetInter nat

mlcm :: N → N → N
mlcm = borrow_from mset msetUnion nat

mdivisible :: N→N→Bool
mdivisible n m = mgcd n m==m

mdiv :: N → N → N
mdiv = borrow_from mset msetDif nat

mexactdiv :: N → N → N
mexactdiv n m | mdivisible n m = mdiv n m
```

and note that properties similar to usual arithmetic operations hold:

$$mprod(mgcd\ x\ y)(mlcm\ x\ y) \equiv mprod\ x\ y \tag{2}$$

$$mexactdiv(mprod\ x\ y)\ y \equiv x \tag{3}$$

$$mexactdiv(mprod\ x\ y)\ x \equiv y \tag{4}$$

4.3 Multiset primes

A remarkable algebraic property of $\mathbb{N}$ is that the lattice structure defined by the divisibility relation has an infinite *antichain*: the set of prime numbers. We will now provide a simple "emulation" of primality that shares this property.

Definition 1 *We say that $p > 1$ is a multiset-prime (or* mprime *shortly), if its decomposition as a multiset is a singleton.*

The following holds

Proposition 4 *$p > 1$ is a multiset prime if and only if it is not mdivisible by any number in $[2..p-1]$.*

Proof. This follows immediately by observing that singleton multisets are the first to contain a given number as the multiset [a,b] corresponds to a number strictly larger than the numbers corresponding to multisets [a] and [b]. □

We are now ready to "emulate" primality in our multiset monoid by defining `is_mprime` (or alternatively `alt_is_mprime`) as a recognizer for *multiset primes* and `mprimes` as a generator of their infinite stream:

```
is_mprime p | p >1 = 1==length (as mset nat p)

alt_is_mprime p | p>1 =
  []==[n|n←[2..p-1],p `mdivisible` n]

mprimes = filter is_mprime [2..]
```

Trying out `mprimes` gives:

```
*MPrimes> take 10 mprimes
[2,3,5,9,17,33,65,129,257,513]
```

suggesting the following proposition:

Proposition 5 *There's an infinite number of* multiset primes *and they are exactly the numbers of the form* $2^n + 1$.

Proof. The proof follows immediately by observing that the first value of `as mset nat n` that contains k, is $n = 2^k + 1$ and that numbers of that form are exactly the numbers resulting in singleton multisets. □

The following example illustrates this property.

```
*MPrimes> map (as mset nat) [1..9]
[[],[1],[2],[1,1],[3],[1,2],[2,2],[1,1,1],[4]]
        ^^^       ^^^                     ^^^
        2+1       4+1                     8+1
```

We can now implement faster versions of `mprimes` and `is_mprime`:

```
mprimes' = map (λx→2^x+1) [0..]

is_mprime' p | p>1 = p==
  last (takeWhile (λx→x≤p) mprimes')
```

4.4 An analog to the "rad" function

Definition 2 *n is square-free if each prime on its list of factors occurs exactly once.*

The `rad(n)` function (`A007947` in [16]) is defined as follows:

Definition 3 *rad(n) is the largest square-free number that divides n*

Clearly, `rad` can be computed by factoring, then trimming multiple occurrences with the `nub` library function and finally by multiplying the resulting primes with `product`.

```
rad n = product (nub (to_primes n))
```

Note that `rad` can also be computed by trimming multiplicities in a multiset representation of n i.e. after defining respectively

```
pfactors n = nub (as pmset nat n)
mfactors n = nub (as mset nat n)
```

we can define `prad` ≡ `rad` and its multiset equivalent `mrad`:

```
prad n =  as nat pmset (pfactors n)

mrad n =  as nat mset (mfactors n)
```

```
*MPrimes> map rad [2..16]
[2,3,2,5,6,7,2,3,10,11,6,13,14,15,2]
*MPrimes> map prad [2..16]
[2,3,2,5,6,7,2,3,10,11,6,13,14,15,2]
*MPrimes> map mrad [2..16]
[2,3,2,5,6,3,2,9,10,11,6,5,6,3,2]
```

A comparison of the plots of the two functions (Fig. 3) shows that `rad`'s chaotic behavior corresponds to a more regular, self-similar behavior in the case of `mrad`.

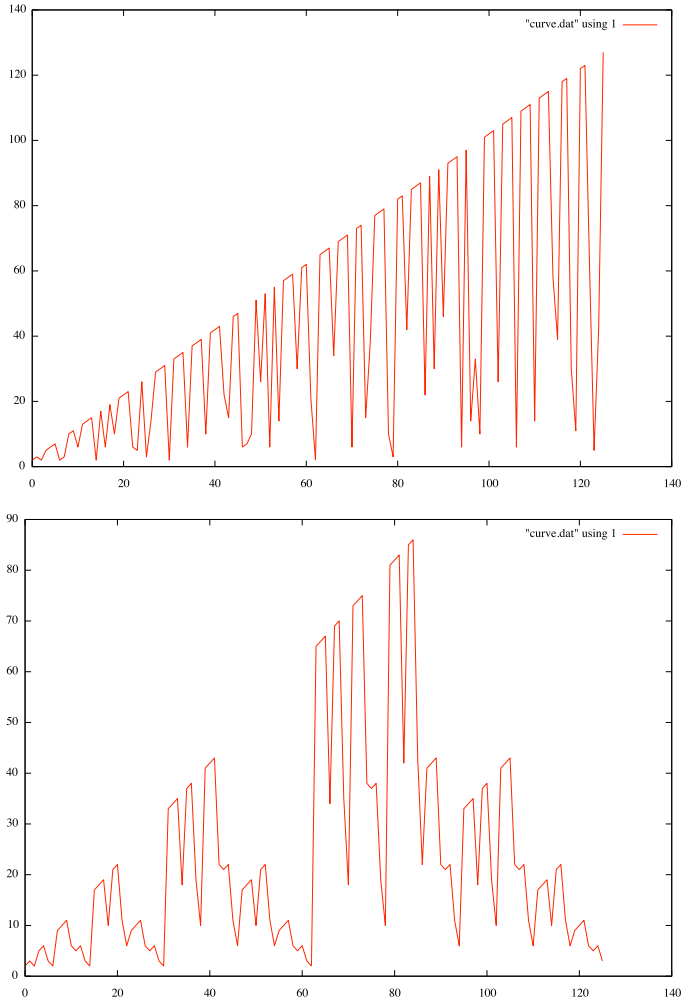


Fig. 3. rad(n) and mrad(n) on $[2..2^7 - 1]$

One can further explore if this "emulation" of the `rad` function can bring some light on the well known connections between the `rad` function and the famous *abc conjecture* [17].

4.5 Emulating the Möbius and Mertens functions

The Möbius function separates square free primes with even and odd number of factors from numbers that are not square free.

$$\mu(n) = \begin{cases} 1 & \text{if } n = 1 \\ 0 & \text{if } p^2 \textit{ divides } n \text{ for some prime } p \\ (-1)^r & \text{if } n \text{ has } r \text{ distinct prime factors} \end{cases}$$

A generalization that parameterizes it by the type `t` of a multiset encoding of natural numbers is given as follows.

```
mobius t n = if nub ns == ns then f ns else 0 where
  ns = as t nat n
  f ns = if even (genericLength ns) then 1 else -1
```

For `t=pmset` one obtains the instance corresponding to *primes* (sequence A008683 in [16]) while `t=mset` provides an instance corresponding to *mprimes*.

```
*MPrimes> map (mobius pmset) [1..16]
[1,-1,-1,0,-1,1,-1,0,0,1,-1,0,-1,1,1,0]

*MPrimes> map (mobius mset) [1..16]
[1,-1,-1,0,-1,1,0,0,-1,1,1,0,0,0,0,0]
```

Surprisingly this corresponds to the apparently identical sequences A132971 and A085357 in [16], related to enumeration of ternary trees and infinite Fibonacci words. We postpone exploring this connections for now, and define, in a similar way a generalization of the Mertens function (A002321 in [16])

$$M(x) = \sum_{n \leq x} \mu(n)$$

that accumulates values of the Möbius function up to n:

```
mertens t n = sum (map (mobius t) [1..n])
```

working as follows

```
*MPrimes> map (mertens pmset) [1..16]
[1,0,-1,-1,-2,-1,-2,-2,-2,-1,-2,-2,-3,-2,-1,-1]

*MPrimes> map (mertens mset) [1..16]
[1,0,-1,-1,-2,-1,-1,-1,-2,-1,0,0,0,0,0,0]
```

The Mertens conjecture (disproved by Odlyzko and te Riele, [18]) states that

$$|M(n)| < \sqrt{n}, \ for \ n > 1$$

After defining

```
mertens2 t n = m^2 where m = mertens t n

counterex_mertens t m = [n|n←[2..m],mertens2 t n ≥ n^2]
```

we can show that it holds for small values for both `t=mset` and `t=pmset`

```
*MPrimes> counterex_mertens pmset 1000
[]
*MPrimes> counterex_mertens mset 1000
[]
```

Fig. 4 shows the more regular, fractal-like behavior of the Mertens function derived from `mset` in contrast with the more chaotic and strongly oscillating behavior of the original Mertens function derived from `pmset`.

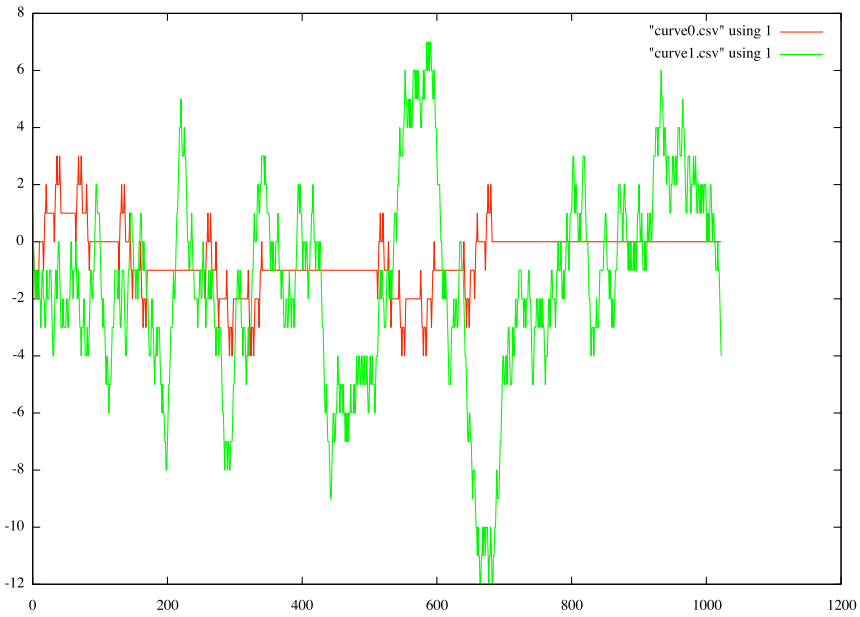


Fig. 4. Mertens functions for mset and pmset

A connection between the Riemann Hypothesis, originating from a representation of the inverse of the Riemann ζ function as

$$\frac{1}{\zeta(s)} = \sum_{n=1}^{\infty} \frac{\mu(n)}{n^s}$$

has lead to an equivalent elementary formulation (attributed to Littlewood) of the Riemann Hypothesis [6, 19] as

$$M(x) = O(x^{1/2+\varepsilon}) \; \forall \varepsilon > 0 \tag{5}$$

By instantiating this statement to a Mertens function parameterized by a simple multiset representation like `mset` one obtains an analogue of the Riemann Hypothesis in much simpler and possibly more tractable context.

Conjecture 2 *The inequality 5 holds for the the instance of $M(x)$ derived from* `mset` *i.e. computed by the function* `mertens mset`*.*

This leads to speculating that, for instance, connecting values of ε between the emulation (derived from `mset`) and the original Martens function (derived from `pmset`) could provide interesting insight on the Riemann Hypothesis as such.

After defining:

```
mlt k m = [n|n←[k..m],mertens2 mset n< mertens2 pmset n]
meq k m = [n|n←[k..m],mertens2 mset n==mertens2 pmset n]
mgt k m = [n|n←[k..m],mertens2 mset n>mertens2 pmset n]
```

experiments indicate that bounds of the `mset` function might be dominated by bounds of the `pmset` function for large values of $k < m$.

```
*MPrimes> length (mlt 1000 1100)
95
*MPrimes> length (meq 1000 1100)
6

*MPrimes> length (mgt 1000 1100)
0
*MPrimes> length (mgt 2000 2100)
2
*MPrimes> length (mgt 3000 3100)
2
```

5 Deriving automorphisms of $\mathbb{N}$

Definition 4 *An* automorphism *is an isomorphism for which the source and target are the same.*

A nice property of automorphisms is that, given the isomorphisms provided by the data transformation framework [9], they propagate from one data type to another. In our case, the multiset representations provided by `pmset` and `mset` induce two automorphisms on $\langle \mathbb{N}, *, 1 \rangle$

```
auto_m2p 0 = 0
auto_m2p n = as nat pmset (as mset nat n)

auto_p2m 0 = 0
auto_p2m n = as nat mset (as pmset nat n)
```

working as follows:

```
*MPrimes> map auto_m2p [0..31]
[0,1,2,3,4,5,6,9,8,7,10,15,12,25,18,27,16,11,14,
 21,20,35,30,45,24,49,50,75,36,125,54,81]

*MPrimes> map auto_p2m [0..31]
[0,1,2,3,4,5,6,9,8,7,10,17,12,33,18,11,16,65,14,
 129,20,19,34,257,24,13,66,15,36,513,22,1025]
```

After extending mprod to mimic pmprod (i.e. *) w.r.t. its behavior on `0`

```
mprod' 0 _ = 0
mprod' _ 0 = 0
mprod' x y | x>0 && y>0= mprod x y
```

note that, as expected, one can define functors that transport `mprod'` into `(*)` and transport `(*)` into `mprod'` as follows:

```
emulated_mprod' x y =
  auto_p2m ((auto_m2p x) * (auto_m2p y))

emulated_prod x y =
  auto_m2p (mprod (auto_p2m x) (auto_p2m y))
```

Note that `emulated_mprod' x y` works as if defined by `mprod' x y` and `emulated_prod x y` works as if defined by `x*y`.

```
*MPrimes> mprod' 20 30
376
*MPrimes> emulated_mprod' 20 30
376
*MPrimes> 20*30
600
*MPrimes> emulated_prod 20 30
600
```

Fig. 5 shows the quickly amplifying reshuffling of the sequence [0..] generated by the functions `auto_m2p`for values in $[0..2^6 - 1]$.

An experimental comparison is between the values for which `auto_m2p` is strictly larger or smaller than `auto_p2m`:

```
lt_mp n = length [x|x←[0..n],auto_m2p x < auto_p2m x]
eq_mp n = length [x|x←[0..n],auto_m2p x == auto_p2m x]
gt_mp n = length [x|x←[0..n],auto_m2p x > auto_p2m x]
```

indicates a shift from `eq_mp` for small numbers to `lt_mp` for larger ones.

```
*MPrimes> map eq_mp [0..15]
[1,2,3,4,5,6,7,8,9,10,11,11,12,12,13,13]
*MPrimes> map gt_mp [0..15]
[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1]
*MPrimes> map lt_mp [0..15]
[0,0,0,0,0,0,0,0,0,0,0,1,1,2,2,2]

*MPrimes> map eq_mp [1000..1015]
```

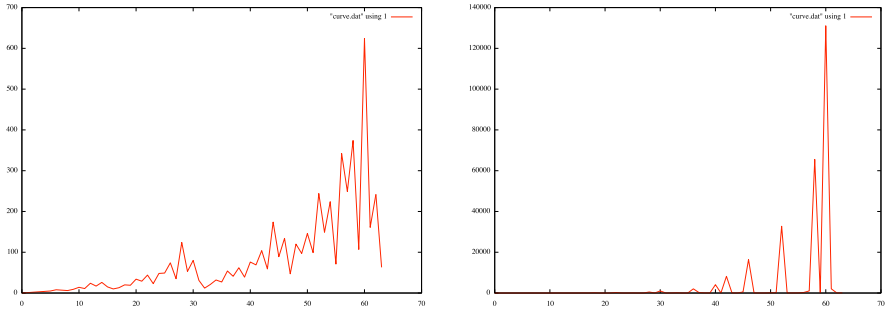


Fig. 5. The automorphisms auto_m2p and auto_p2m

```
[43,43,43,43,43,43,43,43,43,43,43,43,43,43,43,43]
*MPrimes> map gt_mp [1000..1015]
[321,322,322,323,323,323,323,324,325,325,325,325,326,326,327,328]
*MPrimes> map lt_mp [1000..1015]
[637,637,638,638,639,640,641,641,641,642,643,644,644,645,645,645]

*MPrimes> map eq_mp [2000..2015]
[48,48,48,48,48,48,48,48,48,48,48,48,48,48,48,48]
*MPrimes> map gt_mp [2000..2015]
[590,591,592,592,592,592,593,593,593,594,594,594,594,595,596,597]
*MPrimes> map lt_mp [2000..2015]
[1363,... 1364,1365,1366,1366,1367,1368,1368,1369,1370,1371,...,1371]
```

An interesting open problem related to these automorphisms is to prove or disprove that the permutations of $\mathbb{N}$ induced by `auto_m2p` and `auto_p2m` cannot contain infinite cycles.

6 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 [19, 8, 2, 6, 5].

Natural Number encodings of various set-theoretic constructs have triggered the interest of researchers in fields ranging from Axiomatic Set Theory and Foundations of Logic to Complexity Theory and Combinatorics [20–22]. In combinatorics they show up as *Ranking* functions [23] that can be traced back to Gödel numberings [12, 14] associated to formulae. Together with their inverse *unranking* functions they are also used in combinatorial generation algorithms for various data types [24].

This paper relies on the compositional and extensible data transformation framework (summarized in the Appendix) that connects most of the fundamental data types used in computer science with a *groupoid of isomorphisms*. A large (100+ pages) unpublished draft [25], provides encodings between more than 60 different data types. The basic idea of the framework is described in [10] and some of its applications to computational mathematics in [9]. A compact Prolog implementation of the framework with focus on mapping between complex data structures is described in [11].

7 Conclusion and Future Work

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.

This has resulted in a methodology for emulating more difficult number theoretic phenomena through simpler isomorphic representations. In a way this parallels *abstract interpretation* [26] by using a simpler domain to approximate interesting properties in a more complex one.

We are in the process of lifting our Haskell implementation to a generic type class based model, along the lines of [15], which allows experimenting with instances parameterized by arbitrary bijections between $\mathbb{N}$ and $[\mathbb{N}]$. 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. Future work will also focus on finding a matching additive operation for the multiset induced commutative monoid.

Acknowledgment

We thank NSF (research grant 1018172) for support.

References

1. Cégielski, P., Richard, D., Vsemirnov, M.: On the additive theory of prime numbers. Fundam. Inform. **81**(1-3) (2007) 83–96
2. Crandall, R., Pomerance, C.: Prime Numbers–a Computational Approach. Second edn. Springer, New York (2005)
3. Riemann, B.: Ueber die anzahl der primzahlen unter einer gegebenen grösse. Monatsberichte der Berliner Akademie (November 1859)
4. Miller, G.L.: Riemann's hypothesis and tests for primality. In: STOC, ACM (1975) 234–239
5. Lagarias, J.C.: An Elementary Problem Equivalent to the Riemann Hypothesis. The American Mathematical Monthly **109**(6) (2002) pp. 534–543
6. Conrey, B.: The Riemann Hypothesis. Not. Amer. Math. Soc. **60** (2003) 341–353

7. Chaitin, G.: Thoughts on the riemann hypothesis. Math. Intelligencer **26**(1) (2004) 4–7
8. Pippenger, N.: The average amount of information lost in multiplication. IEEE Transactions on Information Theory **51**(2) (2005) 684–687
9. Tarau, P.: A Groupoid of Isomorphic Data Transformations. In Carette, J., Dixon, L., Coen, C.S., Watt, S.M., eds.: Intelligent Computer Mathematics, 16th Symposium, Calculemus 2009, 8th International Conference MKM 2009 , Grand Bend, Canada, Springer, LNAI 5625 (July 2009) 170–185
10. Tarau, P.: Isomorphisms, Hylomorphisms and Hereditarily Finite Data Types in Haskell. In: Proceedings of ACM SAC'09, Honolulu, Hawaii, ACM (March 2009) 1898–1903
11. Tarau, P.: An Embedded Declarative Data Transformation Language. In: Proceedings of 11th International ACM SIGPLAN Symposium PPDP 2009, Coimbra, Portugal, ACM (September 2009) 171–182
12. Gödel, K.: Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I. Monatshefte für Mathematik und Physik **38** (1931) 173–198
13. Singh, D., Ibrahim, A.M., Yohanna, T., Singh, J.N.: An overview of the applications of multisets. Novi Sad J. Math **52**(2) (2007) 73–92
14. Hartmanis, J., Baker, T.P.: On Simple Goedel Numberings and Translations. In Loeckx, J., ed.: ICALP. Volume 14 of Lecture Notes in Computer Science., Berlin Heidelberg, Springer (1974) 301–316
15. Tarau, P.: Declarative modeling of finite mathematics. In: PPDP '10: Proceedings of the 12th international ACM SIGPLAN symposium on Principles and practice of declarative programming, New York, NY, USA, ACM (2010) 131–142
16. Sloane, N.J.A.: The On-Line Encyclopedia of Integer Sequences. (2010) published electronically at www.research.att.com/∼njas/sequences.
17. Granville, A.: ABC allows us to count squarefrees. International Mathematics Research Notices **1998**(19) (1998) 991
18. A. M. Odlyzko, A.M., Te Riele, H.J.J.: Disproof of the Mertens conjecture. J. Reine Angew. Math **357** (1985) 138–160
19. Derbyshire, J.: Prime Obsession: Bernhard Riemann and the Greatest Unsolved Problem in Mathematics. Penguin, New York (2004)
20. Kaye, R., Wong, T.L.: On Interpretations of Arithmetic and Set Theory. Notre Dame J. Formal Logic Volume **48**(4) (2007) 497–510
21. Avigad, J.: The Combinatorics of Propositional Provability. In: ASL Winter Meeting, San Diego (January 1997)
22. Kirby, L.: Addition and multiplication of sets. Math. Log. Q. **53**(1) (2007) 52–65
23. Martinez, C., Molinero, X.: Generic algorithms for the generation of combinatorial objects. In Rovan, B., Vojtas, P., eds.: MFCS. Volume 2747 of Lecture Notes in Computer Science., Berlin Heidelberg, Springer (2003) 572–581
24. Ruskey, F., Proskurowski, A.: Generating binary trees by transpositions. J. Algorithms **11** (1990) 68–84
25. Tarau, P.: Declarative Combinatorics: Isomorphisms, Hylomorphisms and Hereditarily Finite Data Types in Haskell (January 2009) Unpublished draft, http://arXiv.org/abs/0808.2953, updated version at http://logic.cse.unt.edu/tarau/research/2010/ISO.pdf, 150 pages.
26. Cousot, P., Cousot, R.: Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In: 4th ACM Symp. Principles of Programming Languages. (1977) 238–278

Appendix

An Embedded Data Transformation Language

We will describe briefly the embedded data transformation language used in this paper as a set of operations on a groupoid of isomorphisms. We refer to ([9, 25]) for details.

The Groupoid of Isomorphisms We implement an isomorphism between two objects X and Y as a Haskell data type encapsulating a bijection f and its inverse g.

$$X \underset{g = f^{-1}}{\overset{f = g^{-1}}{\rightleftarrows}} Y$$

We will call the *from* function the first component (a *section* in category theory parlance) and the *to* function the second component (a *retraction*) defining the isomorphism. The isomorphisms are naturally organized as a *groupoid*.

```
data Iso a b = Iso (a→b) (b→a)

from (Iso f _) = f

to (Iso _ g) = g

compose :: Iso a b → Iso b c → Iso a c
compose (Iso f g) (Iso f' g') = Iso (f' . f) (g . g')

itself = Iso id id

invert (Iso f g) = Iso g f
```

Assuming that for any pair of type `Iso a b`, $f \circ g = id_b$ and $g \circ f = id_a$, we can now formulate *laws* about these isomorphisms.

The data type `Iso` *has a groupoid structure, i.e. the* compose *operation, when defined, is associative,* itself *acts as an identity element and* invert *computes the inverse of an isomorphism.*

The Hub: Finite Sequences of Natural Numbers To avoid defining $\frac{n(n-1)}{2}$ isomorphisms between n objects, we choose a *Hub* object to/from which we will actually implement isomorphisms.

Choosing a *Hub* object is somewhat arbitrary, but it makes sense to pick a representation that is relatively easy convertible to various others and scalable to accommodate large objects up to the runtime system's actual memory limits.

We will choose as our *Hub* object *finite sequences of natural numbers*. They can be seen as `finite functions` from an initial segment of $\mathbb{N}$, say $[0..n]$, to $\mathbb{N}$. We will represent them as lists i.e. their Haskell type is `[N]`.

```
type N = Integer
type Hub = [N]
```

We can now define an *Encoder* as an isomorphism connecting an object to *Hub*

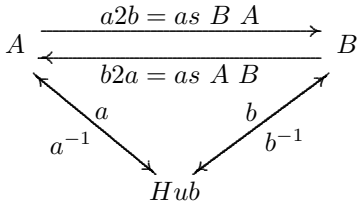
```
type Encoder a = Iso a Hub
```

together with the combinators *as* providing an *embedded transformation language* for routing isomorphisms through two *Encoders*.

```
as :: Encoder a → Encoder b → b → a
as that this x = g x where
   Iso _ g = compose that (invert this)
```

The combinator "as" adds a convenient syntax such that converters between A and B can be designed as:

```
a2b x = as B A x
b2a x = as A B x
```



A particularly useful combinator that transports binary operations from an Encoder to another, borrow_from, can be defined as follows:

```
borrow_from :: Encoder a → (a → a → a) →
               Encoder b → (b → b → b)
borrow_from lender op borrower x y = as borrower lender
   (op (as lender borrower x) (as lender borrower y))
```

Given that [N] has been chosen as the root, we will define our finite function data type *list* simply as the identity isomorphism on sequences in [N].

```
list :: Encoder [N]
list = itself
```

Primes

The following code implements factoring 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 =
  map (succ . (to_pos_in (h:ps))) qs where
    (h:ps)=genericTake n primes
    qs=to_factors n h ps

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

from_pos_in xs n = xs !! (fromIntegral n)
```

Multiset Operations

The following functions provide multiset analogues of the usual set operations, under the assumption that multisets are represented as non-decreasing sequences.

```
msetInter, msetDif, msetSymDif, msetUnion ::
  (Ord a) ⇒ [a] → [a] → [a]

msetInter [] _ = []
msetInter _ [] = []
msetInter (x:xs) (y:ys) | x==y = x:msetInter xs ys
msetInter (x:xs) (y:ys) | x<y = msetInter xs (y:ys)
msetInter (x:xs) (y:ys) | x>y = msetInter (x:xs) ys

msetDif [] _ = []
msetDif xs [] = xs
msetDif (x:xs) (y:ys) | x==y = msetDif xs ys
msetDif (x:xs) (y:ys) | x<y = x:msetDif xs (y:ys)
msetDif (x:xs) (y:ys) | x>y = msetDif (x:xs) ys

msetSymDif xs ys =
  sortedConcat (msetDif xs ys) (msetDif ys xs)

msetUnion xs ys =
  sortedConcat (msetSymDif xs ys) (msetInter xs ys)

sortedConcat xs ys = sort (xs ++ ys)
```