# Polynomials in Haskell

Andrew Lelechenko
1@dxdy.ru

Barclays, London

MuniHac, 12.09.2020

# Univariate polynomials

## Definition

A polynomial is

- either a monomial $c \cdot x^k$ for $k \geqslant 0$,
- or a sum of two polynomials.

- $2x^3$, $x^2$, $4$ are polynomials.
- $1/x$ is not a polynomial.
- $4 + 2x^3$ and $x + x^2$ are polynomials as well.
- Polynomials can be added and multiplied.

$$(x+1) + (x-1) = 2x,$$
$$(x+1) \times (x-1) = x^2 - 1.$$

# Why Haskell?

- **Question:**
  There are efficient Fortran libraries for polynomial manipulations. Can we use them?
- **Answer:**
  Not really, Fortran libraries are mostly tied to `Double`.
- My main applications are algebra and cryptography, where coefficients are discrete (think of `Integer` and modular arithmetic).
- We would like our implementation of polynomials to be polymorphic in coefficients.
- Haskell has a good track record for polymorphism, but a modest one for performance.

# Haskell translation

```
data Poly a
  = X { power :: Word, coeff :: a }
  | Poly a :+ Poly a
  deriving (Eq)

instance Num a => Num (Poly a) where
  fromInteger n = X 0 (fromInteger n)
  (+) = (:+)

  (x :+ y) * z = (x * z) :+ (y * z)
  x * (y :+ z) = (x * y) :+ (x * z)
  X k c * X l d = X (k + l) (c * d)
```

**Beautiful, innit?**

However polynomial addition, as defined, is not associative:

$$x + (y + z) \neq (x + y) + z.$$

# Fixing associativity

- There are some basic blocks (monomials).
- There is an associative operation (addition) with a neutral element (monomial with zero coefficient).
- Thus, polynomials are monoids over monomials.
- Lists are **free** monoids, let's use them to model a non-free one.

```
data Mono a = X { power :: Word, coeff :: a }
  deriving (Eq, Ord)
mul :: Num a ⇒ Mono a → Mono a → Mono a
mul (X k c) (X l d) = X (k + l) (c * d)

newtype Poly a = Poly [Mono a] deriving (Eq)
instance Num a ⇒ Num (Poly a) where
  Poly xs + Poly ys = Poly (xs <> ys)
  Poly xs * Poly ys = foldl (+) (Poly [])
    (map (\x → Poly (map (mul x) ys)) xs)
```

**Better?**

# Fixing commutativity

Not quite: polynomial addition, as defined, is not commutative:

$$x + y \neq y + x.$$

Commutativity means that all rearrangements of `[Mono a]` must be equivalent.

```
data Mono a = X { power :: Word, coeff :: a }
  deriving (Eq, Ord)

newtype Poly a = Poly [Mono a]
instance Ord a ⇒ Eq (Poly a) where
  (==) = (==) `on` sort
```

**No!** Structural equality is too important to be sacrificed.

# Sorted lists

```
newtype Sorted a = Sorted { unSorted :: [a] }

sorted :: [a] → Sorted a
sorted = Sorted . sort

merge :: Ord a ⇒ [a] → [a] → [a]
merge xs [] = xs
merge [] ys = ys
merge (x : xs) (y : ys)
  | x ⩽ y     = x : merge xs (y : ys)
  | otherwise = y : merge (x : xs) ys
```

**All sorted?**

Not yet: we would also expect that $x + x = 2x$ and $0x = 0$.

# Final touches

```
merge :: (Eq a, Num a) ⇒ [Mono a] → [Mono a] → [Mono a]
merge xs [] = xs
merge [] ys = ys
merge (X k c : xs) (X l d : ys) = case k ‘compare‘ l of
  LT → X k c : merge xs (X l d : ys)
  EQ → case c + d of
    0 → merge xs ys
    e → X k e : merge xs ys
  GT → Y l d : merge (X k c : xs) (ys)

newtype Poly a = Poly [Mono a] deriving (Eq)
instance (Eq a, Num a) ⇒ Num (Poly a) where
  Poly xs + Poly ys = Poly (xs ‘merge‘ ys)
  Poly xs * Poly ys = foldl (+) (Poly [])
    (map (\x → Poly (map (mul x) ys)) xs)
```

**Structural equality is restored!**

# Memory representation

- `Mono a` takes 5+ words: tag, pointer to a power and its value, pointer to a coefficient and its value.
- `[Mono a]` takes 8+ words per monomial.
- If `a` is `Double` or `Int`, this is very expensive.
- Lazy lists accumulate long sequence of thunks, especially in multiplication.
- **Solution:** use vectors!
- Shall we use boxed or unboxed vectors?
- **Solution:** be polymorphic by vector flavour.

```
newtype Poly v a = Poly (v (Word, a)) deriving (Eq)
instance (Eq a, Num a, Vector v (Word, a)) ⇒ Num (Poly v a)
```

**Vectors make polynomial arithmetic $20\times$ faster!**

# Dense polynomials

- We arrived to `Poly a` $\sim$ `[Mono a]` by choosing a sorted list as a canonical representative of a class of equivalent polynomials.
- Another choice of canonical representative is a polynomial of the same degree, but with monomials for all powers present.
- It's enough to store coefficients alone: `Poly a` $\sim$ `[a]`.
- This representation allows to implement asymptotically-superior algorithms:
  - Karatsuba multiplication $O(n^{1.585})$,
  - fast Fourier transform $O(n \log n)$.

| length | `polynomial`, $\mu s$ | `poly`, $\mu s$ | speedup |
|---:|---:|---:|---:|
| 100 | 1733 | 33 | 52$\times$ |
| 1000 | 442000 | 1456 | 303$\times$ |

# User interface targeting REPL

- We need a nice way to input and output polynomials.
- What about `deriving (Show, Read)`?
- Derived `Show` looks nowhere close to a mathematical expression.

```
Poly {unPoly = [(0,2),(1,-3),(2,1)]}
```

- What about a hand-written `Show`?

```
> :set -XOverloadedLists
> [(2,1), (1,-3), (0,2)] :: VPoly Int
1 * X^2 + (-3) * X + 2
```

- But writing correspondent `Read` would be abysmal!
- There is no point to support

```
read "X^2 - 3 * X + 2"
```

if one can define `X` as a pattern and write immediately

```
X^2 - 3 * X + 2 :: Poly Int
```

# Division with remainder

- We defined addition and multiplication, and subtraction poses no problem. What about division?

$$\begin{aligned} \text{quotRem}\ \ x^2\ x &= (x, 0), \\ \text{quotRem}\ (x^2+1)\ x &= (x, 1), \\ \text{quotRem}\ (x^2+1)\ (x+1) &= (x-1, 2). \end{aligned}$$

- Things get difficult for integral coefficients:

$$\begin{aligned} \text{quotRem}\ \ x\ 2.0 &= (0.5x, 0), \\ \text{quotRem}\ \ x\ 2\quad &= ? \end{aligned}$$

```
quotRemPoly
  :: Fractional a
  ⇒ Poly a → Poly a → (Poly a, Poly a)
quotRemPoly = <long division algorithm>
```

# Greatest common divisor

- Usually gcd is computed using Euclid's algorithm, which involves repeated quotRem:

$$\gcd(70, 25) = \gcd(25, 20) = \gcd(20, 5) = 5.$$

- Is it possible to implement gcd without access to quotRem? E. g., $\gcd(2x^2 - 2, 5x + 5) = x + 1$.
- It appears that having access to gcd on coefficients is enough to implement gcd for polynomials.
- Multiply, not divide!

$$\begin{aligned}\gcd(2x^2 - 2, 5x + 5) &\sim \gcd(10x^2 - 10, 10x + 10) \sim \\ &\sim \gcd(10x^2 - 10 - x(10x + 10), 10x + 10) \sim \\ &\sim \gcd(-10x - 10, 10x + 10) \sim 10x + 10.\end{aligned}$$

# What's wrong with `Integral`?

```haskell
class (Real a, Enum a) ⇒ Integral a where
  quotRem :: a → a → (a, a)
  toInteger :: a → Integer
  ...
gcd :: Integral a ⇒ a → a → a
```

- Restrictive superclasses, which has nothing to do with `quotRem`.
- Obnoxious `toInteger`, which coupled with `Num.fromInteger` means that only subrings of `Integer` can be `Integral`.
- Function `gcd` is constrained to domains, allowing division with reminder, which is too restrictive for polynomials.

# Integral done right

```
class Num a ⇒ GcdDomain a where
  gcd :: a → a → a
  default gcd :: (Eq a, Euclidean a) ⇒ a → a → a

class GcdDomain a ⇒ Euclidean a where
  quotRem :: a → a → (a, a)

instance GcdDomain a ⇒ GcdDomain (Poly a) where
  ...
instance Fractional a ⇒ Euclidean (Poly a) where
  ...
```

# Project structure

- All actual code is in `Internal` subtree.
- Public, user-facing modules are just dummies with re-exports.
- This approach helps a lot with circular dependencies between modules.

```
Data
└── Poly
    ├── Internal
    │   ├── Dense
    │   │   ├── DFT.hs
    │   │   ├── Field.hs
    │   │   └── GcdDomain.hs
    │   ├── Dense.hs
    │   ├── Sparse
    │   │   ├── Field.hs
    │   │   └── GcdDomain.hs
    │   └── Sparse.hs
    ├── Semiring.hs
    ├── Sparse
    │   ├── Multi.hs
    │   └── Semiring.hs
    └── Sparse.hs
└── Poly.hs
```

# Package poly

- Polynomials, polymorphic by coefficients and containers.
- Full-featured:
  - Dense and sparse representations.
  - Laurent polynomials, allowing negative powers.
  - Type-safe polynomials over many variables.
  - Special polynomial sequences.
- GC- and cache-friendly implementation.
- Unrivaled performance amongst Haskell native packages.
- 1000+ tests, 98% test coverage.

# Thank you!

@ 1@dxdy.ru    Bodigrim
github.com/Bodigrim/poly    github.com/Bodigrim/my-talks