

Rechnen mit Polynomen

Sven Eric Panitz

16. April 2019

Inhaltsverzeichnis

1	Polynomberechnungen	1
2	Der Datentyp für Polynome	1
2.1	Instanz von ToLaTeX	2
3	Aufgaben	4
4	Lernzuwachs	6

1 Polynomberechnungen

In dieser Aufgabe beschäftigen wir uns mit Polynomen. Ein Polynom besteht aus der Addition von Monomen. Ein Monom hat einen Koeffizienten und einen Exponenten. Ein Beispiel eines Polynoms aus 6 Monomen ist:

$$6 * x^6 - 2 * x^5 - 4 * x^3 + 3 * x + 3$$

2 Der Datentyp für Polynome

Wir definieren das Modul für Polynomberechnungen:

Haskell: Poly

```
> module Poly where
```

Wir verwenden für die Koeffizienten den Datentyp Rational.

Haskell: Poly

```
> import Data.Ratio
```

Ein Polynom wird als eine Liste von Monomen dargestellt. Somit werden wir auf einige Listenfunktionen zurückgreifen.

Haskell: Poly

```
> import Data.List
```

Ein Monom ist schlicht der Produkttyp aus einem `Rational` für den Koeffizienten und einer Zahl für den Exponenten. Wir verwenden vorerst die Standarddefinition für Gleichheit und Anzeige als String.

Haskell: Poly

```
> data Monom = M Rational Int
> deriving (Show,Eq)
```

Ein Polynom ist eine Liste von Monomen. Wir kapseln die Liste von Monomen mit einem Konstruktor `P`, damit wir den neuen Typ `Poly` zu Instanzen von unterschiedlichen Typklassen machen können.

Haskell: Poly

```
> newtype Poly = P [Monom]
> deriving (Show,Eq)
```

Ein kleiner Hilfsoperator soll uns ermöglichen, einfache Polynome schnell zu erstellen.

Haskell: Poly

```
> infix 8 #^
> a #^ b = P [M a b]
```

Somit lässt sich zum Beispiel das Polynom $3 * x^2$ einfach notieren als:

Haskell: Poly

```
> polyEx = 3#^2
```

2.1 Instanz von ToLaTeX

Um unsere Polynome, aber auch andere Datentypen, in einem `LATEX`-Dokument schön anzeigen zu lassen, definieren wir eine kleine Typklasse, die die Daten in einem direkt im

Math-Mode von L^AT_EX integrierbaren String umwandeln:

Haskell: Poly

```
> class ToLaTeX a where
>   toLaTeX :: a -> String
```

Als erstes wird für alle Typen von rationalen Zahlen eine Instanz der Typklasse ToLaTeX definiert:

Haskell: Poly

```
> instance (Integral a, Eq a, Num a, Show a) => ToLaTeX (Ratio a) where
>   toLaTeX r
>   | n==1 = show z
>   | otherwise = "\\frac{"++show z++"}{"++show n++"}"
>   where
>     z = numerator r
>     n = denominator r
```

Zunächst wird der Typ Monom als Instanz dieser Typklasse definiert.

Haskell: Poly

```
> instance ToLaTeX Monom where
```

Wenn der Exponent 0 ist, ist der Koeffizient allein anzuzeigen.

Haskell: Poly

```
>   toLaTeX (M k 0) = toLaTeX k
```

Ist der Exponent 1 und der Koeffizient 1, dann wird das Monom einfach durch die Funktionsvariable x ausgedrückt.

Haskell: Poly

```
>   toLaTeX (M k 1)
>   | k==1 = "x"
```

Wenn der Koeffizient nicht 1 ist, dann multipliziert dieser die Funktionsvariable x :

Haskell: Poly

```
>   | otherwise = toLaTeX k++"*x"
```

Wenn es einen Koeffizienten gibt, dann wird der entsprechende L^AT_EX-Code für Exponenten verwendet.

Haskell: Poly

```
> toLaTeX (M k e)
>   |k==1 = "x^{++show e++}"
>   |otherwise = toLaTeX k++"*x^{++show e++}"
```

Schließlich der \LaTeX -Code für ein ganzes Polynom.

Haskell: Poly

```
> instance ToLaTeX Poly where
>   toLaTeX (P []) = "0"
>   toLaTeX (P (x:xs))
>     = (toLaTeX x)
>       ++ (foldl (++) ""
>                $ map (\m@(M k e)->
>                        (if (k>=0) then "+" else "")++toLaTeX m)
>                  xs)
```

3 Aufgaben

- a) Schreiben Sie eine Operatorfunktion, die das Polynom für einen Wert als Argument ausrechnet.

Haskell: Poly

```
> infix 9 §
> (§) :: Poly -> Rational -> Rational
> (P xs) § x = 0
```

Ein Polynom stellt eine Funktion dar. Wir rechnen mit den Polynomen auf rationalen Zahlen. Mit dem Operator \S soll die Polynomfunktion auf ein konkretes Argument angewendet werden.

- b) Schreiben Sie eine Funktion zur Normalisierung von Polynomen.

Haskell: Poly

```
> normalize :: Poly -> Poly
> normalize = id
```

Ein normalisiertes Polynom soll folgende Regeln befolgen:

- Es enthält kein Monom mit einem Koeffizienten 0.
- Es enthält keine zwei Monome mit gleichem Exponenten.

- Die Monome sind mit absteigenden Exponenten sortiert.

Die Funktion `normalize` soll dafür sorgen, dass ein Polynom zu einem äquivalenten normalisierten Polynom umgeformt wird.

- c) Schreiben Sie eine Funktion zur Addition von Polynomen.

Haskell: Poly

```
> add :: Poly -> Poly -> Poly
> add p1 p2 = P[]
```

Es soll für zwei Polynome p_1 und p_2 gelten:

$$(p_1 + p_2)(x) = p_1(x) + p_2(x)$$

Das Ergebnispolynom soll dabei normalisiert sein.

- d) Schreiben Sie eine Funktion für das Negat eines Polynoms.

Haskell: Poly

```
> negat :: Poly -> Poly
> negat = id
```

Es soll für ein Polynome p gelten:

$$(negat(p))(x) = -p(x)$$

- e) Schreiben Sie eine Funktion zur Multiplikation von Polynomen.

Haskell: Poly

```
> mult :: Poly -> Poly -> Poly
> mult p1 p2 = P[]
```

Es soll für zwei Polynome p_1 und p_2 gelten:

$$(p_1 * p_2)(x) = p_1(x) * p_2(x)$$

Das Ergebnispolynom soll dabei normalisiert sein.

- f) Schreiben Sie eine Operator-Funktion zur Division mit Rest.

Haskell: Poly

```
> infix 2 /\
> (/%) :: Poly -> Poly -> (Poly, Poly)
> p1 /\ p2 = (p1, p2)
```

Das Ergebnis ist ein Paar. Verfahren Sie nach der Polynomdivision, wie sie auf Wikipedia beschrieben ist: de.wikipedia.org/wiki/Polynomdivision.

- g) Schreiben Sie eine Funktion, die die erste Ableitung eines Polynoms berechnet.

Haskell: Poly

```
> derivation :: Poly -> Poly
> derivation = id
```

Das Ergebnispolynom soll normalisiert sein.

- h) Machen Sie Poly zu einer Instanz von der Typklasse Num.

Haskell: Poly

```
> instance Num Poly where
>   fromInteger x = error "not implemented"
>   negate p = negat p
>   p1 + p2 = add p1 p2
>   abs (P xs) = error "not implemented"
>   signum _ = error "not implemented"
>   p1 * p2 = mult p1 p2
```

Sie haben schon das Negat, die Addition und die Multiplikation. Damit ist schon fast alles vorhanden, um die Typklasse Num zu implementieren.

Wenn Poly die Typklasse implementiert, können Sie das einführende Beispielpolynom wie folgt definieren.

Haskell: Poly

```
> example1 = 6 #^ 6 - 2#^5 - 4#^3 + 3#^1 + 3
```

Und hierauf weitere Rechnungen durchführen:

Haskell: Poly

```
> example2 = 2#^3 + 2#^1 - 3
> res1 = example1 /% example1
> res2 = example1 /% example2
```

4 Lernzuwachs

- `newtype` statt eines Typsynonyms, um den Typ zur Instanz einer Typklasse machen zu können.
- Symbolisches Rechnen auf einem Datentyp.

- Instanz der Typklasse `Num` für einen eigenen Datentyp deklarieren und von allen Operatoren und Literalen profitieren.