

- Kreuzen Sie gelöste Aufgaben im OLAT Kurs des Proseminars an.
- Lösen Sie Programmieraufgaben in `Template_06.hs` und laden Sie diese Datei in OLAT hoch.
- Ihre Template-Datei sollte mit `ghci` ohne Fehlermeldung kompilieren.

Aufgabe 1 *Polynome*

5 P.

Gegeben ist der Datentyp `data Polynom = Polynom [Int]` in Haskell, der Polynome repräsentiert. Dabei entspricht die Liste der Integer den Koeffizienten des Polynoms, beginnend beim konstanten Term. Zum Beispiel repräsentiert `Polynom [1,-2,3]` das Polynom $3x^2 - 2x + 1$.

1. Implementieren Sie die Funktion `normalize :: Polynom -> Polynom`, welche alle führenden Nullen (d.h. Koeffizienten der höchsten Grade, die 0 sind) entfernt. Außerdem sollen Sie `Polynom` zu einer Instanz von `Show` machen. Dabei sollten alle Terme mit Koeffizient 0 weggelassen werden. (2 Punkte)

Beispiel: `normalize (Polynom [0,1,7,0,4,0,0]) == Polynom [0,1,7,0,4]`

`show (Polynom [5,0,-4,3,0]) == "3x^3+(-4)x^2+5"`

Hinweis: Die Haskell-Funktion `reverse` und die gegebene Funktion `allZero :: [Int] -> Bool` könnten hilfreich sein.

2. Machen Sie `Polynom` zu einer Instanz von `Eq` und `Ord`. Führende Nullen sollen dabei ignoriert werden, d.h. `Polynom [1,5,4,0] == Polynom [1,5,4]` sollte `True` ergeben. Für die Ordnung (`Ord`) gilt: Zuerst wird der Grad des Polynoms verglichen. Haben zwei Polynome den gleichen Grad, entscheidet der Koeffizient des höchsten Terms; ist dieser gleich, werden die nächstniedrigen Koeffizienten verglichen usw. (1 Punkt)
3. Machen Sie `Polynom` zu einer Instanz von `Num`. Für Polynome bedeutet das konkret:
 - `(+)` sollte die polynomielle Addition implementieren.
Beispiel: `Polynom [1,2] + Polynom [0,3] == Polynom [1,5]`.
 - `(*)` sollte die polynomielle Multiplikation implementieren.
Beispiel: `Polynom [1,2,3] * Polynom [5,6] == Polynom [5,16,27,18]`.
 - `negate` sollte jeden Koeffizienten des Polynom negieren.
Beispiel: `negate (Polynom [1,2,3]) == Polynom [-1,-2,-3]`.
 - `fromInteger` sollte einen Integer in ein konstantes Polynom umwandeln.
Beispiel: `fromInteger 5 == Polynom [5]`.
 - `abs` wandelt alle Koeffizienten in ihre Beträge um.
 - `signum` gibt das Vorzeichen des Koeffizienten des höchsten Terms an.

(2 Punkte)

Aufgabe 2

5 P.

Ein `Monoid` ist eine algebraische Struktur, die aus einer assoziativen binären Operation $\circ$ und einem neutralen Element e besteht, wobei die folgenden Gesetze für alle x, y, z erfüllt sind:

- $x \circ (y \circ z) = (x \circ y) \circ z$

- $x \circ e = e \circ x = x$

Wir modellieren ein Monoid in Haskell mit der folgenden Klasse.

```
class MonoidC a where
  binop :: a -> a -> a
  neutral :: a
```

1. Betrachten Sie die folgenden Instanzen:

- Zahlen mit Multiplikation als binäre Operation.
- Boolesche Werte mit logischem UND als binäre Operation.
- Listen, in denen die binäre Operation Verkettung ist.

In allen drei Fällen kann die Wahl des neutralen Elements aus den Monoid-Gesetzen abgeleitet werden.

Definieren Sie die beschriebenen `MonoidC` Instanzen für `Integer`, `Bool` und `[a]`. (1 Punkt)

2. Wir betrachten einfache Abstimmungssequenzen, um die Stimmen auf unseren Social-Media-Posts zu verfolgen, wobei der Buchstabe 'U' einen Upvote (+1) repräsentiert; und der Buchstabe 'D' ein Downvote (-1) repräsentiert. Zum Beispiel repräsentieren "UU", "UUUD" und "UDDUUU" eine Gesamtabstimmung von +2, während "DDD" und "DDUDD" eine Gesamtabstimmung von -3 darstellen.

Definieren Sie eine Funktion `normaliseVote`, die jede Abstimmungssequenz vereinfacht, so dass sie keine benachbarten Paaren ("UD" oder "DU") die sich aufheben beinhalten. Alle Sequenzen, die die gleiche Gesamtabstimmung darstellen, sollten sich auf dieselbe Form normalisieren. Zum Beispiel, alle Sequenzen, die eine Gesamtabstimmung von +2 darstellen, sollten sich auf "UU" normalisieren. (1 Punkt)

3. Wir definieren einen Datentyp `VoteSeq` mit Konstruktor `VS :: String -> VoteSeq`, um diese Abstimmungssequenzen darzustellen. Machen Sie `VoteSeq` zu einer Instanz von `Eq`, `Show` und `MonoidC`. Stellen Sie sicher, dass das Abstimmungssequenzen normalisiert werden. Darüber hinaus sollte `VS xs == VS ys` `True` zurückgeben, wenn `xs` und `ys` die gleiche Gesamtstimmzahl darstellen. (1 Punkt)

4. Definieren Sie eine Funktion `combine`, die eine Liste von Elementen $x_1, \dots, x_n$ in einem Monoid erhält und daraus die Kombination dieser Elemente berechnet, d.h. $x_1 \circ \dots \circ x_n \circ e$. Die Funktionsdefinition sollte den (allgemeinsten) Typ von `combine` enthalten. (1 Punkt)

5. Nach Abschluss der bisherigen Aufgaben:

- Beschreiben Sie informell, was die Funktion `combine :: [Integer] -> Integer` berechnet.
- Beschreiben Sie informell, was die Funktion `combine :: [String] -> String` berechnet.
- Beschreiben Sie, warum `combine [[4,2,0]]` und `combine [4,2,0]` sich unterscheiden.
- Erklären Sie den Unterschied zwischen dem Ergebnis von `combine ["UUDDU", "UDDDU", "DDDU"]` und dem von `combine [VS "UUDDU", VS "UDDDU", VS "DDDU"]`. (1 Punkt)