

Übung: Funktionen und Typen 1

- Algebraische Typen und Pattern-Matching
- Listen

1 Aufgabe

Gegeben sei der folgende Datentyp

```
1 data Car = Car
2   { model :: Model
3     , make  :: Make
4     , year  :: Integer
5     , color :: Color
6     , power :: Horsepower
7   }
```

(a) Deklarieren Sie geeignete Datentypen `Model`, `Make`, `Color`, `Horsepower`.

(b) Erstellen Sie drei Autos:

`ford` Einen roten Ford Fiesta mit *70PS* aus dem Jahr 2017.

`ferrari` Einen grünen Ferrari (andere Parameter frei).

`zoe` Einen weissen Renault Zoe (andere Parameter frei).

(c) Implementieren Sie eine `Ord` Instanz für den `Car` Typ. Beachten Sie die Regeln der Typklasse (<http://hackage.haskell.org/package/base-4.12.0.0/docs/Data-Ord.html>).

(d) Importieren Sie `Data.List` und führen Sie die Funktion

```
sort [zoe, ferrari, ford]
```

aus.

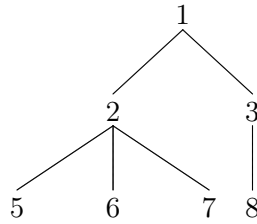
2 Aufgabe

Wir erinnern uns an die Implementierung von Binärbäumen aus der Vorlesung:

```
1 data Tree a
2   = Node (Tree a) a (Tree a)
3   | Leaf a
```

(a) Implementieren Sie eine Funktion `collect :: Tree a -> [a]`, die alle Elemente eines Binärbaumes in einer Liste “aufsammelt”.

- (b) Passen Sie den Datentyp aus der Vorlesung so an, dass beliebig verzweigte Bäume modelliert werden können (nicht nur Binärbäume).
- (c) Erstellen Sie mit ihrem Datentyp den Baum



3 Aufgabe

- (a) Implementieren Sie einen Typ `NatNumber` entsprechend der Definition:

Eine natürliche Zahl ist entweder Null oder Nachfolger einer natürlichen Zahl.

- (b) Implementieren Sie eine Funktion `eval: NatNumber -> Integer`, die natürliche Zahlen “auswertet” und eine Funktion `uneval`, die sich zu `eval` “dual” verhält (also Integers in NatNumbers konvertiert).
- (c) Definieren Sie mit einem Pattern-Match eine Funktion `fact: NatNumber -> NatNumber`. Halten Sie sich dabei möglichst exakt¹ an die mathematische Definition:

$$\begin{aligned} fact(0) &= 1 \\ fact(n+1) &= (n+1) \cdot fact(n) \end{aligned}$$

Hinweis: Sie müssen dazu die Grundoperationen der Addition und Multiplikation “Bootsrappen”.

4 Aufgabe

Lesen Sie diesen Eintrag über die “Programmiersprache” Fractran. Implementieren Sie einen Fractran interpreter (beachten Sie dazu die Anmerkungen in der Vorlesung)

`execute: Program -> Input -> Output`

Definieren Sie wo nötig selber geeignete Datentypen.

¹Achten Sie insbesondere darauf, dass links “ $n+1$ ” steht und nicht rechts “ $n-1$ ”.