Dieses Dokument ist vom 10.11.2025. Die aktuelle Version des Dokuments kannst du im moodle oder direkt von GitHub herunterladen.

Dieses Dokument enthält 82 Fragen, 8 kleinere bis größere Aufgaben und andere Ressourcen zum Thema Deklarative Programmierung. Die Inhalte dieses Dokuments sollen dir helfen, dein Verständnis über Haskell und Prolog zu prüfen.

Größere Aufgaben haben wir als Challenges markiert. Diese Aufgaben benötigen öfter mehrere Konzepte und führen zusätzlich Konzepte ein, die nur für das Lösen der Aufgabe wichtig sind.

Wenn du Anmerkungen oder weitere Ideen für Inhalte für dieses Dokument hast, dann schreibe uns gerne über z.B. mattermost an – oder erstellt ein issue oder stellt eine PR auf GitHub.

Funktionale Programmierung

Test 1 Was bedeutet es, wenn eine Funktion keine Seiteneffekte hat?

Test 2 Haskell ist eine streng getypte Programmiersprache. Was bedeutet das?

Test 3 Wenn du eine Schleife in Haskell umsetzen möchtest, auf welches Konzept musst du dann zurückgreifen?

Test 4 In imperativen Programmiersprachen sind Variablen Namen für Speicherzellen, deren Werte zum Beispiel in Schleifen verändert werden können. Als Beispiel betrachte

```
def clz(n):
  k = 0
  while n > 0:
    n //= 2
    k += 1
  return 64 - k

def popcnt(n):
  k = 0
  while n > 0:
    if n % 2 == 1:
      k += 1
    n //= 2
  return k
```

In Haskell sind Variablen keine Namen für Speicherzellen. Wie können wir dieses Programm in Haskell umsetzen? Wo wandert das `k` hin?

Test 5 Auf was müssen wir achten, wenn wir eine rekursive Funktion definieren? Die Antwort ist abhängig von dem, was die Funktion berechnen soll. Denke über die verschiedenen Möglichkeiten nach.

Test 6 Gegeben sei das folgende Haskell-Programm.

```
even :: Int -> Bool
even 0 = True
even n = odd (n - 1)

odd :: Int -> Bool
odd 0 = False
odd n = even (n - 1)
```

- Berechne das Ergebnis von `odd (1 + 1)` händisch.
- Wie sieht der Auswertungsgraph für den Ausdruck `odd (1 + 1)` aus?

- Welcher Pfad entspricht deiner händischen Auswertung?
- Welcher Pfad entspricht der Auswertung wie sie in Haskell stattfindet?
- Welcher Pfad entspricht der Auswertung wie sie in Python sinngemäß stattfindet?

Test 7 Es wird als sauberer Programmierstil angesehen, Hilfsfunktionen, die nur für eine Funktion relevant sind, nicht auf der höchsten Ebene zu definieren. Mithilfe welcher Konstrukte kannst du diese lokal definieren?

Test 8 Das Potenzieren einer Zahl x (oder eines Elements einer Halbgruppe) mit einem natürlich-zahligen Exponent n ist in $\mathcal{O}(\log n)$ Laufzeit möglich[1]. Dafür betrachten wir

$$x^n = \begin{cases} \left(x^{\frac{n}{2}}\right)^2 & \text{falls } n \text{ gerade} \\ x \cdot x\left(x^{\frac{n-1}{2}}\right)^2 & \text{sonst} \end{cases}$$

Implementiere eine Funktion, die diese Variante des Potenzierens umsetzt.

Test 9 Gegeben ist folgender Ausdruck.

```
let v = 3
    w = 5
    x = 4
    y = v + x
    z = x + y
 in y
```

Welche Belegungen der Variablen werden tatsächlich berechnet, wenn wir `y` ausrechnen?

Test 10 Ist der folgende Ausdruck typkorrekt?

```
if 0 then 3.141 else 3141
```

Test 11 Wie werden algebraische Datentypen in Haskell definiert?

Test 12 Was ist charakterisierend für Aufzählungstypen, einen Verbundstypen und einem rekursiven Datentypen? Gebe Beispiele für jeden dieser Typarten an.

Test 13 Geben ist der Typ `IntList` mit `data IntList = Nil | Cons Int IntList`. Weiter kann mithilfe der Funktion

```
lengthIntList :: IntList -> Int
lengthIntList Nil         = 0
lengthIntList (Cons _ xs) = 1 + lengthIntList xs
```

die Länge einer solchen Liste berechnet werden. Du möchtest nun auch die Längen von Listen berechnen, die Buchstaben, Booleans oder Gleitkommazahlen enthalten. Was stört dich am bisherigen Vorgehen? Kennst du ein Konzept mit dessen Hilfe du besser an dein Ziel kommst?

Test 14 Wie ist die Funktion `lengthIntList :: IntList -> Int` aus dem vorherigen Test definiert?

Test 15 Du hast einen Datentypen definiert und möchtest dir Werte des Typen nun z.B. im GHCi anzeigen lassen. Was kannst du tun, um an dieses Ziel zu kommen?

Test 16 Wie definieren wir Funktionen?

Test 17 Gebe ein Listendatentypen an, für den es nicht möglich ist, kein Element zu enthalten.

[1] Binäre Exponentiation

Test 18 In Programmiersprachen wie Java greifen wir Daten komplexer Datentypen zu, indem wir auf Attribute von Objekten zugreifen oder getter-Methoden verwenden. Wie greifen wir auf Daten in Haskell zu?

Test 19 Wie sieht eine Datentypdefinition im Allgemeinen aus?

Test 20 Welchen Typ haben

- `(:)` und `[]`,
- `Just` und `Nothing`,
- `Left` und `Right`?

Test 21 Was ist parametrischer Polymorphismus?

Test 22 Welche Typkonstruktoren des kinds `* -> *` kennst du?

Test 23 Welchen kind hat `Either a`?

Test 24 Beim Programmieren vernachlässigen redundante Syntax. Gibt es einen Unterschied zwischen `f 1 2` und `f(1, 2)`

Test 25 Welches Konzept erlaubt es uns, dass wir Funktionen auf Listen nicht für jeden konkreten Typen angeben müssen?

Test 26 Wie gewinnt man aus einem Typkonstruktor einen Typ?

Test 27 Visualisiere `[1, 2, 3]` als Baum, wie du es in der Vorlesung kennengelernt hast. Zur Erinnerung: die inneren Knoten sind Funktionen und die Blätter Werte, die nicht weiter ausgerechnet werden können.

Test 28 Ist `[32, True, "Hello, world!"]` ein valider Haskell-Wert? Warum ja oder nein?

Test 29 Was ist der Unterschied zwischen einem Typ und einem Typkonstruktor?

Test 30 Gegeben ist

```
data Pair a b = Pair a b
```

Wie unterscheidet sich der Typ von

```
data Pair a = Pair a a
```

Challenge 1

- Der größte gemeinsamen Teiler (ggT) zweier Ganzzahlen kann mithilfe des euklidschen Algorithmus berechnet werden. Implementiere das Verfahren.

$$\gcd(x, y) = \begin{cases} |x| & \text{falls } y = 0 \\ \gcd(y, x \bmod y) & \text{sonst} \end{cases}$$

- Alternativ kann der ggT auch berechnet werden, indem wir das Produkt des Schnittes der Primfaktorzerlegung der beiden Zahlen betrachten, also

$$\prod(\mathrm{PF}(x) \cap \mathrm{PF}(y))$$

  wobei PF die Menge der Primfaktoren der gegebenen Zahl (mit entsprechenden Mehrfachvorkommen) beschreiben soll. Implementiere diesen Ansatz.

Challenge 2 Die Ableitung einer Funktion $f : \mathbb{R} \to \mathbb{R}$ kann mithilfe des Differenzenquotienten $\frac{f(x+h)-f(x)}{h}$ für kleines h approximiert werden. Ein andere Methode zur Berechnung der Ableitung ist symbolisches Differenzen und ähnelt dem, wie wir analytisch Ableitungen berechnen. Eine Funktion sei dargestellt durch den folgenden Typ:

```haskell
data Fun = X            -- x     (Variable x)
         | E            -- e     (Euler's constant)
         | Num Double   -- c     (Constant)
         | Ln Fun       -- ln    (Natural logarithm)
         | Fun :+: Fun  -- f + g (Addition)
         | Fun :-: Fun  -- f - g (Subtraction)
         | Fun :*: Fun  -- f * g (Multiplication)
         | Fun :/: Fun  -- f / g (Division)
         | Fun :<: Fun  -- f o g (Composition)
         | Fun :^: Fun  -- f ^ g (Exponentiation)

-- Example
f :: Fun
f = (E :^: X) :<: (X :*: X)  -- (e^x) o (x * x) = e^(x^2)

-- Example
g :: Fun
g :: let x = X
         x2 = x :*: x
         x3 = x2 :*: x
     in x3 :+: x2 :+: x :+: Num 1.0  -- x^3 + x^2 + x + 1
```

- Implementiere eine Funktion `($$) :: Fun -> Double -> Double`, die eine gegebene Funktion in einem gegebenen Punkt auswertet.
- Implementiere eine Funktion `derive :: Fun -> Fun`, die eine gegebene Funktion ableitet.[2]

Challenge 3 In Einführung in die Algorithmik hast du verschiedene Varianten des `mergesort`-Algorithmus kennengelernt. Eine davon hat ausgenutzt, dass in einer Eingabeliste bereits aufsteigend sortierte Teillisten vorkommen können, um den Algorithmus zu beschleunigen.[3] Implementiere diese Variante in Haskell.

Für den Anfang kannst du annehmen, dass die Eingabelisten vom Typ `[Int]` sind. Wenn wir Typklassen behandelt haben, kannst du `Ord a => [a]` nutzen.

Challenge 4 Entwickle einen Datentyp `Ratio`, um rationale Zahlen

$$\frac{p}{q} \in \mathbb{Q}, \quad p \in \mathbb{Z}, q \in \mathbb{N}, p \text{ und } q \text{ teilerfremd}$$

darzustellen. Implementiere die Operationen: Addition, Subtraktion, Multiplikation, Divison. Implementiere weiter eine Funktion, die die rationale Zahl als reelle Zahl mit einer festen Anzahl von Nachkommastellen darstellt.

Test 31 Wie können wir es hinkriegen, dass die invalide Liste `[32, True, "Hello, world!"]` ein valider Haskell-Wert wird? Mithilfe welches Hilfstypen kriegen das hin?

Test 32 Du hast bereits viele Funktionen kennengelernt, die in der Haskell `base`-library implementiert sind. Anstatt eine konkrete Liste dieser Funktionen anzugeben, möchten wir dich motivieren, folgende Dokumentationen verschiedener Module anzuschauen.

- Prelude
- Data.List

Wenn du merkst, die Implementierung einer bekannten Funktion fällt dir ad hoc nicht ein, nehme dir Zeit und überlege, wie du sie implementieren könntest.

[2] Zusammenfassung der Ableitungsregeln

[3] Falls du interessiert bist: In der Haskell `base`-library wird `sort` aus `Data.List` sehr ähnlich implementiert: Data.List.sort.

Test 33 Hier ist eine fehlerhafte Implementierung eines Datentyps für einen knotenbeschrifteten Binärbäumen.

```
data Tree a = Empty | Node Tree a Tree
```

Was ist der Fehler?

Test 34 In imperativen Programmierung iterieren wir über Listen oft in folgender Form (in Java).

```
List<Integer> a = new ArrayList<>();
a.add(3); a.add(1); a.add(4); a.add(1); a.add(5);

List<Integer> b = new ArrayList<>();
for (int i = 0; i < a.size(); i++) {
  b.add(2 * a.get(i));
}
```

Wenn wir diesen Code naiv in Haskell übersetzen, könnten wir z.B.

```
double :: [Int] -> Int -> [Int]
double xs i | i < length xs = 2 * xs !! i : double xs (i + 1)
            | otherwise     = []
```

Das wollen wir niemals so tun.

- Wie unterscheiden sich die Laufzeiten?
- Optimiere die Funktion `double`, sodass sie lineare Laufzeit in der Länge der Liste hat.

Test 35 Die `(!!)`-Funktion ist unsicher in dem Sinne, dass sie für invalide Listenzugriffe einen Fehler wirft. Die Funktion `(!?) :: [a] -> Int -> Maybe a` ist eine sichere Variante von `(!!)`. Sie macht den Fehlerfall explizit durch die Wahl des Ergebnistypen. Was tut diese Funktion voraussichtlich? Implementiere diese Funktion.[4]

Test 36 `(++) :: [a] -> [a] -> [a]` wird verwendet, um zwei Listen aneinanderzuhängen. Wenn wir eine Funktion induktiv über den Listentypen definieren wie z.B. `square :: [Int] -> [Int]`, die jeden Listeneintrag quadrieren soll, dann können wir das wie folgt tun.

```
square :: [Int] -> [Int]
square []     = []
square (x:xs) = [x * x] ++ square xs
```

Die Funktion ist zwar korrekt aber nicht Haskell-idiomatisch, d.h., eine Person, die Erfahren im Programmieren von Haskell ist, würde dies nicht so schreiben. Was müssten wir an der Funktion ändern, damit sie idiomatisch ist.

Test 37 Die Funktion `show` kann genutzt werden, um Werte eines beliebigen Datentyp in eine String-Repräsentation zu überführen. Warum kann `show` nicht als Funktion vom Typ `a -> String` implementiert sein?

Challenge 5 In den Übungsaufgaben hast du einen Suchbaum ohne Höhenbalancierung implementiert. Die Rotationen für einen AVL-Baum lassen sich durch das pattern matching in Haskell vergleichsweise elegant implementieren - erinnere dich z.B. an die Implementierung aus Einführung in die Algorithmik, die recht verbos ist.

Die Höhe eines Teilbaums kann z.B. als weiteres Attribut im Knoten gespeichert werden. Eine ineffizientere Variante ist es, die Höhe mit einer Funktion wiederkehrend zu berechnen. Letztere Variante ist für den Anfang übersichtlicher.

[4] Diese Funktion ist auch bereits vorimplementiert: `(!?)` in `Data.List`.

Implementiere eine Funktion `rotate :: SearchTree a -> SearchTree a`, die einen Teilbaum an der Wurzel rebalanciert, sollte der Teilbaum unbalanciert sein. Diese Funktion kannst du dann nutzen, um die gängigen Operationen auf Suchbäumen anzupassen.[5]

Test 38 Formuliere QuickCheck-Eigenschaften, die die Funktionen

- `isElem :: Int -> SearchTree Int -> Bool`,
- `toList :: SearchTree Int -> Int`,
- `insert :: Int -> SearchTree Int -> SearchTree Int` und
- `delete :: Int -> SearchTree Int -> SearchTree Int`

erfüllen sollen. `isElem` überprüft, ob eine Ganzzahl in gegebenen Suchbaum enthalten ist. `toList` konvertiert einen Suchbaum in eine Liste. `insert` fügt eine Ganzzahl in einen Suchbaum ein. `delete` löscht eine Ganzzahl aus einen Suchbaum.

Wie kannst du die Suchbaum-Eigenschaft spezifizieren (dafür brauchst du weitere Funktionen)?

Test 39 QuickCheck-Eigenschaften werden mit zufällig generierten Werten getestet. Hin und wieder kommt es vor, dass diese Werte Vorbedingungen erfüllen müssen, damit wir Eigenschaften von Funktionen testen können. Wie können wir das erreichen?

Test 40 Wie können wir eine Funktionen teilweise auf Korrektheit testen – also wie können wir für eine beliebige Eingabe verifizieren, dass die Ausgabe korrekt ist?

Test 41 Was sind Funktionen höherer Ordnung?

Test 42 Wie definieren wir Lambda-Abstraktionen bzw. anonyme Funktionen?

Test 43 Warum ist der Typ `(a -> b) -> c` nicht identisch zum Typ `a -> b -> c`? Welcher andere Typ ist identisch zu letzterem?

Test 44 Mit welchen Konzepten gehen die Linksassoziativität der Funktionsapplikation und die Rechtsassoziatvität des Typkonstruktors `(->)` gut Hand in Hand?

Test 45 Zu welchen partiell applizierten Funktionen verhalten sich folgenden Funktionen identisch?

- `succ :: Int -> Int` (die Inkrementfunktion)
- `pred :: Int -> Int` (die Dekrementfunktion)
- `length :: [a] -> Int`
- `sum :: [Int] -> Int`
- `product :: [Int] -> Int`

Test 46 Was ist partielle Applikation?

Test 47 Was ist Currying?

Test 48 Welche Funktionen höherer Ordnung hast du kennengelernt im Kontext der generischen Programmierung? Was ist das Ziel dieser Funktionen?

Test 49 Die Funktionen `map :: (a -> b) -> [a] -> [b]` und `filter :: (a -> Bool) -> [a] -> [a]` lassen sich alle mithilfe `foldr :: (a -> r -> r) -> r -> [a] -> r` ausdrücken. Wie erreichen wir dies?

Test 50 Gegeben seien folgende Funktionen:

- `rgbToHsv :: RGB -> HSV`, die eine Farbe von einer Darstellung in einen anderen konvertiert, und

[5] Rebalancierung eines AVL-Baum

- `hue :: HSV -> Float`, die den Farbwert einer Farbe im Wertebereich $[0°, 360°)$ im HSV-Farbraum zurückgibt.

Du bekommst als Eingabe einen Bild, das hier als Liste von `RGB`-Werten dargestellt ist. Jeder `RGB`-Wert korrespondiert zu einem Pixel. Schreibe eine Funktion, die berechnet, wie viele blaue Pixel das Bild hat. Hier bezeichene eine Farbe als blau, wenn ihr Farbwert zwischen 200° und 250° (inklusiv) liegt. Nutze für die Definition der Funktion sowohl `map` als auch `filter`.

Test 51 Was sind sections im Kontext von Funktionen höherer Ordnung?

Test 52 Welche der Faltungsfunktion auf Listen ergibt sich aus dem Verfahren zur Erzeugung von Faltungsfunktionen, das du für beliebige Datentypen kennengelernt hast?

Test 53 Was ist der Unterschied zwischen `foldl` und `foldr`? Wann liefern `foldl` und `foldr` das gleiche Ergebnis?

Test 54 Wie gewinnen wir aus `foldr` die Identitätsfunktion auf Listen? In den Übungen hast du gelernt, wie man Werte anderer Typen falten kann. Wie gewinnt man aus diesen Funktionen die Identitätsfunktionen auf den jeweiligen Typen?

Test 55 Gegeben sind folgende Datentypen
- `data Tree a = Empty | Leaf a | Branch (Tree a) (Tree a)`,
- `data Rose a = Node [Rose a]`.

Welche Typen haben die jeweiligen Datenkonstruktoren und wie führen wir diese in die Signatur der jeweiligen Faltungsfunktion über? Wo benötigen wir rekursive Aufrufe der jeweiligen Faltungsfunktionen?

Test 56 Betrachte die Funktion

```
f :: [a] -> b
f []     = e
f (x:xs) = g x (f xs)
```

Nach diesem induktiven Muster sind viele Funktionen auf Listen implementiert. Nehme als Beispiel die Funktion `sum :: [Int] -> [Int]`.

```
sum :: [Int] -> Int
sum []     = 0
sum (x:xs) = x + sum xs
```

Dieses Muster haben wir in `foldr` abstrahiert. Wo wandern die jeweiligen Bestandteile der abstrakten Funktion `f` hin, wenn wir `f` mithilfe von `foldr` definieren. Was passiert insbesondere mit dem rekursiven Aufruf von `f`?

Test 57 Wie kannst du mithilfe von Faltung viele Elemente in einen Suchbaum einfügen oder lösen? Implementiere
- `insertMany :: [Int] -> SearchTree Int -> SearchTree Int` und
- `deleteMany :: [Int] -> SearchTree Int -> SearchTree Int`.

Du kannst davon ausgehen, dass du die Einfüge- und Löschfunktion für einzelne Elemente bereits hast.

Test 58 Wir können `map :: (a -> b) -> [a] -> [b]` mithilfe von `foldr` wie folgt implementieren:

```
map f xs = foldr (\x ys -> f x : ys) [] xs
```

Vereinfache den Lambda-Ausdruck mithilfe von Funktionen höherer Ordnung.

Test 59 Wenn wir die Listenkonstruktoren in `foldr` einsetzen, erhalten wir die Identitätsfunktion auf Listen, also

```
foldr (:) [] :: [a] -> [a].
```

Wenn wir das Gleiche mit `foldl` und angepassten `(:)` machen, also

```
foldl (flip (:)) [] :: [a] -> [a],
```

dann erhalten wir nicht die Identitätsfunktion auf Listen. Warum und was bekommen wir stattdessen heraus?

Test 60 Es gibt viele andere hilfreiche Funktionen höherer Ordnung in der Haskell Prelude. Eine von diesen ist `zipWith :: (a -> b -> c) -> [a] -> [b] -> [c]`. Sie verknüpft jeweils zwei Elemente aus den jeweiligen Listen unter der gegeben Funktion.

- Implementiere `zipWith` mithilfe von `map, uncurry, zip`.
- Implementiere `zip` mithilfe von `zipWith`.
- Implementiere das Prädikat `isSorted` mithilfe von `zipWith`.

Challenge 6 Gegeben sei die Funktion Faltungsfunktion `foldTree :: (r -> a -> r -> r) -> r -> Tree a -> r` für einen knotenbeschrifteten Binärbaum gegeben durch `data Tree a = Empty | Node (Tree a) a (Tree a)`.

Wie auch für Listen lassen sich eine Reihe von bekannten Funktionen auf Bäume übertragen.[6] Implementiere die Funktionen

- `any :: (a -> Bool) -> Tree a -> Bool` und `and :: (a -> Bool) -> Tree a -> Bool`,
- `elem :: Int -> Tree Int -> Bool` und `notElem :: Int -> Tree Int -> Bool`,
- `toList :: Tree a -> Bool`,
- `null :: Tree a -> Bool` (überprüft, ob der Baum leer ist),
- `length :: Tree a -> Int`,
- `maximum :: Tree Int -> Int` und `minimum :: Tree Int -> Int`, und
- `sum :: Tree Int -> Int` und `product :: Tree Int -> Int`.

Test 61 Gegeben seien die Funktion

```
f :: a -> b
g :: a -> b -> c
```

sowie die Kompositionsfunktion `(.) :: (b -> c) -> (a -> b) -> a -> c`.

In der Typdefinition von `(.)` scheint das erste Argument, eine einstellige Funktion zu sein. Ist der Ausdruck

```
g . f
```

trotzdem typkorrekt, obwohl `g` eine zweistellige Funktion ist? Wenn ja, wie werden die Typvariablen – insbesondere das `c` der Komposition – unifiziert?

Test 62 Welche Funktion verbirgt sich hinter `foldr ((++) . f) []` und was ist ihr Typ?

Test 63 Versuche in den folgenden Ausdrücken, Teilausdrücke schrittweise durch bekannte Funktionen zu ersetzen und gegebenenfalls zu vereinfachen.

- `foldr (\x ys -> f x : ys) [] (foldr (\x ys -> g x : ys) [] xs)`,
- `map (\_ -> y) xs`,
- `foldr (\x ys -> if x `mod` 2 == 1 then x - 1 : ys else ys) [] xs`,

[6] Diese Funktionen lassen sich auf alle faltbaren Datentypen verallgemeinern. Für Interessierte: Dies wird mithilfe der Typklasse `Foldable` festgehalten – diese Typklasse behandeln wir aber in der Vorlesung voraussichtlich nicht.

- `foldl (\ys x -> x : ys) [] xs` und
- `flip (curry fst) x`.[7]

Challenge 7 Sei $m, n \in \mathbb{N}$. Gegeben seien
- die Identitätsfunktion auf $\{0, ..., n-1\}$ mit $\pi_0 : \{0, ..., n-1\} \to \{0, ..., n-1\}, x \mapsto x$ und
- eine endliche Folge von Paaren $((a_i, b_i))_{i \in \{1,...,m\}}$ mit $a_i \in \{0, ..., n-1\}, b_i \in \{0, ..., n-1\}$ für alle $i \in \{1, ..., m\}$.

Wir definieren $\pi_{i,j}$ als

$$\pi_{i,j}(k) = \begin{cases} i & \text{falls } k = j \\ j & \text{falls } k = i \\ k & \text{sonst} \end{cases}$$

für alle $k \in \{0, ..., n-1\}$.

Wir betrachten die Paare als Vertauschungen der Bilder der Abbildung π_0, d.h.,

$$\pi_i = \pi_{a_i,b_i} \circ \pi_{i-1} \text{ für } i \in \{1, ..., m\}.$$

- Implementiere eine Funktion `swaps :: Int -> [(Int, Int)] -> [Int]`, die π_m mithilfe von Listen berechnet. Der erste Parameter bestimmt die Menge $\{0, ..., n-1\}$ und der zweite die Folge.
- Implementiere eine Funktion `swaps :: Int -> [(Int, Int)] -> Int -> Int`, die π_m mithilfe von Funktion berechnet. (Hier ist der erste Parameter unter Umständen redundant.)
- Welche Vor- und Nachteile haben die jeweiligen Ansätze im Vergleich?

Test 64 Eta-reduziere die folgende Ausdrücke:
- `sum xs = foldr (+) 0 xs`,
- `add a b = a + b` und
- `\x ys -> (:) x ys`.

Test 65 Implementiere die Funktion `insert :: Int -> a -> Map Int a -> Map Int a`, die ein Schlüssel-Wert-Paar in eine `Map Int a` einfügt. Die `Map` ist wie folgt repräsentiert `type Map k v = k -> v`.

Test 66 Was ist ein abstrakter Datentyp? Was sind die Bestandteile eines abstrakten Datentyps?

Test 67 Wie definieren wir die Semantik der einem abstrakten Datentyp gehörenden Operationen? Wie definieren wir sie insbesondere nicht?

Test 68 Wieso ist das sofortige Nutzen einer Gleichheit auf einem abstrakten Datentypen problematisch? Was sollte man stattdessen tun?

Test 69 Zur Spezifikation der Semantik nutzen wir Gesetze, die bestimmen, wie verschiedene Operationen miteinander interagieren. Dafür benötigen wir verschiedene Werte oftmals unterschiedlicher Datentypen. Wo kommen diese her und wie sind sie quantifiziert?

Test 70 Welche Eigenschaften sollten die für einen abstrakten Datentypen formulierten Gesetze erfüllen?

[7] „Your scientists were so preoccupied with whether or not they could, that they didn't stop to think if they should." Jenseits solcher kleinen Verständnisfragen gilt weiterhin, dass wir verständlichen Code schreiben wollen. Solche Ausdrücke sind häufig schwieriger zu verstehen – auch wenn es unterhaltsam ist, sich solche Ausdrücke auszudenken.

Challenge 8 Gebe folgende abstrakte Datentypen an: Paar, Menge, stack, queue, double-ended queue, knotenbeschrifteter Binärbaum, priority queue.

Welcome weary traveler, you've come far. Why don't you stop here and get some rest before you continue your journey.

(Mit den Inhalten der Vorlesung kannst du die Tests und Challenges aktuell bis hier hin lösen.)

Test 71 Was sind Typklassen?

Test 72 Wie unterscheidet sich der Polymorphismus, der durch Typklassen ermöglicht wird, vom parametrischen Polymorphismus?

Test 73 In einem vorherigen Test wurdest du bereits gefragt, wieso `show` nicht als Funktion mit dem Typ `a -> String` implementiert sein. Wieso wird die Funktion durch `Show a => a -> String` gerettet?

Test 74 Welche Typklassen kennst du? Was ermöglichen sie konkret in den Einzelfällen?

Test 75 Überlade die Operationen `(+)`, `(-)`, `(*)`, `abs`, `signum`, `fromInteger` für den Datentypen `data Mat22 a = Mat22 a a a a`, der (2×2)-Matrizen repräsentieren soll – `abs`, `signum`, `fromInteger` kannst du z.B. komponentenweise implementieren.[8]

Test 76 Mit

$$\begin{pmatrix} f_{n+1} & f_n \\ f_n & f_{n-1} \end{pmatrix}^n = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n$$

und der binären Exponentiation und `Mat22 Integer` aus einem vorherigen Tests kannst du die n-te Fibonacci-Zahl in logarithmischer Laufzeit in n berechnen. Implementiere das Verfahren.

Da du eine `Num`-Instanz auf `Mat22` definiert hast, kannst du den `(^)`-Operator zum binären Exponentiation nutzen.

An vielen Stellen in den bisherigen Selbsttests haben wir oft einen konkreten Typen (z.B. `Int`) genutzt, für den es bestimmte Typklasseninstanzen gibt. Das ist meistens der Fall gewesen, wenn wir Gleichheit auf Werten oder eine Vergleichsoperation auf Werten brauchten. Schau dir die bisherigen Selbsttests erneut an und überlege dir, wo du Typen verallgemeinern kannst.

Test 77 Welche Funktionen musst du implementieren, damit eine `Eq`-Instanz vollständig definiert ist?

Test 78 Welche Funktionen musst du implementieren, damit eine `Ord`-Instanz vollständig definiert ist?

Test 79 In nicht streng getypten Programmiersprachen haben wir oft, mit impliziter Typkonversion zu tun.[9] Implementiere eine Funktion `ifThenElse`, die als Bedingung Werte beliebiger Typen entgegennehmen kann. Ziel ist es, dass das folgende Ausdruck ausgewertet werden kann.

```
let a = ifThenElse 0 3 4
    b = ifThenElse [5] 6 7
```

[8] Oft sind an Funktionen von Typklassen Bedingungen bzw. Gesetze, die erfüllt werden sollen, gekoppelt. Das für `abs` und `signum` wird durch den Vorschlag nicht erfüllt.

[9] Diese wollen nun für einen Moment nach Haskell zurückholen, um sie dann ganz schnell wieder zu vergessen.

```
    c = ifThenElse Nothing 8 9
in a + b + c  -- 19
```

Test 80 Eine Halbgruppe ist eine Struktur $(H, *)$, wobei $*$ eine assoziative, binäre Verknüpfung $* : H \times H \to H$ ist. Ein Monoid erweitert die Halbgruppe um ein neutrales Element bzgl. $*$.

Definiere Typklassen `Semigroup` und `Monoid`, diese Strukturen implementieren. Gebe auch beispielhaft ein paar Instanzen für diese an.

Test 81 Wie können wir mit `foldl` auf unendlichen Listen mit keinem Ergebnis rechnen?

Test 82 Gegeben ist der Typ `data Deep a b = Deep [Maybe (Either a (Deep a b))]`. Die Faltungsfunktion sieht im Wesentlichen so aus.

```
foldDeep :: ([Maybe (Either a r)] -> r) -> Deep a b -> r
foldDeep fdeep (Deep x) = fdeep (f x)
```

Wie können wir `f :: [Maybe (Either a (Deep a b))] -> [Maybe (Either a r)]` mithilfe von `foldDeep fdeep :: Deep a b -> r` definieren?

Hinweise zu Tests und Challenges

Hinweis zu Challenge 1 Zur Darstellung der Multimengen eignen sich sortierte Listen gut.

Hinweis zu Challenge 1 Zur Berechnung des Schnittes können zwei sortierte Listen parallel durchlaufen werden. Wenn zwei gleiche Elemente zu Beginn der Liste stehen, wird eines der Elemente zum Ergebnis hinzugefügt. Im anderen Fall überspringen wir das jeweils kleinere Element der beiden.

Weitere Ressourcen

Wenn du auf der Suche nach weiteren Übungsaufgaben bist, mit denen du deine Programmierkenntnisse in Prolog verbessern möchtest, bietet sich die Liste P-99: Ninety-Nine Prolog Problems an. Lösungen sind ebenso auf der Seite verfügbar. Für Haskell gibt es eine ähnliche Seite H-99: Ninety-Nine Haskell Problems.

Weitere Links:
- Learn You A Haskell
- Haskelite: Ein Schritt-für-Schritt Interpreter für (eine Teilmenge von) Haskell
- Haskell Cheatsheet