

Informatik I - Gruppe 1 - Übungsblatt 6

Ausgabe: 14.11.2022

Abgabe: 21.11.1022

Tutor: Tim Völker

Ali Kurt 528961

Thomas Kujawa 463620

Felix Hoff 366927

Aufgabe T6.1: Rekursion (3 Punkte)

Gegeben sei eine Funktion f , definiert durch:

$$f(n) = \begin{cases} 2 \cdot f(n-1) + 1 & \text{für } n \geq 1 \\ 1 & \text{sonst} \end{cases}$$

(a) Geben Sie eine Wertetabelle an, die für $n = 0, 1, 2, 3, 4, 5, 6$ das Ergebnis $f(n)$ angibt.

n	0	1	2	3	4	5	6
$f(n)$	1	1	3	7	15	31	63

(b) Geben Sie den schrittweisen Ablauf der rekursiven Funktion als Expansion und Kontraktion für den Wert $n = 4$ an.

$$\begin{aligned} f(4) &= 2 \cdot f(4-1) + 1 \\ &= 2 \cdot f(3) + 1 \\ &= 2 \cdot (2 \cdot f(2) + 1) + 1 \\ &= 2 \cdot (2 \cdot (2 \cdot f(1) + 1) + 1) + 1 \\ &= 2 \cdot (2 \cdot (2 \cdot 1 + 1) + 1) + 1 \\ &= 2 \cdot (2 \cdot (3) + 1) + 1 \\ &= 2 \cdot (7) + 1 \\ &= 15 \end{aligned}$$

(c) In welchem Verhältnis oder mathematischen Zusammenhang stehen $n \geq 0$ und der Funktionswert $f(n)$? Geben Sie die Funktion $f(n)$ in einer nicht-rekursiven Form an.

$$f(n) = 2^n - 1$$

Aufgabe T6.2: ADT Spezifikation (4 Punkte)

In der Vorlesung haben Sie abstrakte Datentypen (ADT) kennengelernt. Dabei wurde in der Vorlesung die Spezifikation eines Datentyps *A* vorgestellt (Folie 291). Sie sollen nun 2 Realisierungen des Datentyps *A* angeben, die nicht bereits in der Vorlesung vorgestellt wurden (Bool, Nat0 und anderen Datentypen aus Num) und welche die Signatur des Datentyps *A* realisieren. Überprüfen Sie für beide Realisierungen, ob diese die Spezifikation erfüllen.

```
1  --Vorlesungsbeispiel für den ADT A
2  --Signatur:
3  class ADT a where
4      a, b :: a
5      op1  :: a -> a
6      op2, op3 :: a -> a -> a
7  --Semantik
8      op1 a = b
9      op2 a x = x
10     op3 b x = x
11
12 --Beispiel 1: ADT Color
13 --Signatur:
14 class Color c where
15     red, green, blue, yellow :: c
16     kompleColor              :: c -> c
17     mixColor                  :: c -> Float -> c
18     lightenColor              :: c -> c -> c
19 --Semantik
20     kompleColor red = green
21     mixColor blue yellow = green
22     lightenColor c f = c
23
24 --Beispiel 2: ADT Fish
25 --Signatur:
26 class Fish f where
27     shark, trout :: f
28     saltwaterFish :: f -> Bool
29     isLarger, isStronger :: f -> f -> f
30 --Semantik
31     saltwaterFish f = True
32     isLarger f g = f
33     isStronger f g = g
```

Informatik I - Gruppe 1 - Übungsblatt 6

Ausgabe: 14.11.2022

Abgabe: 21.11.1022

Tutor: Tim Völker

Ali Kurt 528961

Thomas Kujawa 463620

Felix Hoff 366927

Aufgabe P6.3: Binomialkoeffizienten / Pascalsches Dreieck (2 Punkte)

Zur Ermittlung der Binomialkoeffizienten $\binom{n}{k}$ kann das Pascalsche Dreieck zu Hilfe gezogen werden. Den passenden Koeffizienten findet man in der n -ten Zeile an der k -ten Stelle:

				1					
				1		1			
			1		2		1		
		1		3		3		1	
	1		4		6		4		1
	1	5		10		10	5		1
	1	6	15		20	15	6	1	
1	7	21	35	35	21	7	1		

Dabei kann dies mathematisch durch folgende Funktion f definiert werden:

$$\begin{aligned} f(n, 0) &= 1 \\ f(n, n) &= 1 \\ f(n, k) &= \begin{cases} 0 & \text{für } k > n \\ f(n-1, k-1) + f(n-1, k) & \text{sonst} \end{cases} \end{aligned}$$

Implementieren Sie die Funktion `pascal :: Int -> Int -> Int`, die die Parameter n und k entgegennimmt und den Binomialkoeffizienten zurückliefert. Implementieren Sie dies als rekursive Funktion. Geben Sie die Berechnung für `pascal (3, 1)` als Kommentar an. Geben Sie außerdem an, ob es sich um eine direkte oder indirekte Rekursion handelt.

```
1 pascal :: Int -> Int -> Int
2 pascal n k
3     | k == 0 = 1
4     | n == k = 1
5     | otherwise = if k > n
6         then 0
7         else
8             pascal (n-1) (k-1) + pascal (n-1) k
```

Aufgabe P6.4: Rationale Zahlen (3 + 3 = 6 Punkte)

In der Vorlesung haben Sie eine Haskell-Implementierung der rationalen Zahlen kennengelernt. In dieser Aufgaben soll die Implementierung des Datentyps `Ratio` erweitert werden. Folgen Sie dabei dem Programm-entwurf der Vorlesung und erstellen Sie (1) Funktionsköpfe, (2) Beispiele und (3) Funktionsrümpfe sowie (4) eine Überprüfung anhand der generierten Beispiele.

(a) Entwickeln Sie eine Funktion `reduceFraction`, die eine rationale Zahl entgegen nimmt und die vollständig gekürzte rationale Zahl zurück gibt. Implementieren Sie dazu die Hilfsfunktion `ggT`, die den größten gemeinsamen Teiler zweier Zahlen a und b bestimmt, indem sie a zurück gibt, falls $b = 0$. Ansonsten soll der größte gemeinsame Teiler von b und $a \% b$ bestimmt werden.

(b) Entwickeln Sie Funktionen `fromRatio` und `toRatio`, die eine Konvertierung zwischen dem Datentyp `Float` und den rationalen Zahlen vornehmen. Die rationale Zahl soll dazu in ihrer vollständig gekürzten Form zurück gegeben werden. Nicht jede rationale Zahl kann als Gleitkommazahl exakt dargestellt werden. Runden Sie die Gleitkommazahlen dazu im Kontext dieser Aufgabe auf 5 Nachkommastellen.

Hinweis: Sie können die Haskell-Datei aus dem Learnweb als Startpunkt nutzen.

```
1 import Distribution.Simple.Flag (Flag)
2 -- Datentyp für rationale Zahlen repräsentiert durch zwei Ganzzahlen für Zähler und Nenner
3 data Ratio = MkRatio Integer Integer
4
5
6 -- Gleichheit für rationale Zahlen
7 instance Eq Ratio where
8     (MkRatio nom1 den1) == (MkRatio nom2 den2) = (nom1 * den2 == nom2 * den1)
9
10 -- Ordnung für rationale Zahlen
11 instance Ord Ratio where
12     (MkRatio nom1 den1) <= (MkRatio nom2 den2) =
13         if ((signum den1) * (signum den2) == 1)
14             then (nom1 * den2 <= nom2 * den1)
15             else (nom2 * den1 <= nom1 * den2)
16
17 -- Rechenoperationen für rationale Zahlen inklusive Addition, Multiplikation, Negierung,
18   ↳ Betragsfunktion, Vorzeichenfunktion und Konvertierung von Integer
19 instance Num Ratio where
20     (+) (MkRatio nom1 den1) (MkRatio nom2 den2) = (MkRatio (nom1 * den2 + nom2 * den1) (den1 *
21   ↳ den2))
22     (*) (MkRatio nom1 den1) (MkRatio nom2 den2) = (MkRatio (nom1 * nom2) (den1 * den2))
23     negate (MkRatio nom den) = (MkRatio (- nom) den)
24     abs (MkRatio nom den) = (MkRatio (abs nom) (abs den))
25     signum (MkRatio nom den) = (MkRatio ((signum nom) * (signum den)) 1)
26     fromInteger i = (MkRatio i 1)
```

Informatik I - Gruppe 1 - Übungsblatt 6

Ausgabe: 14.11.2022

Abgabe: 21.11.1022

Tutor: Tim Völker

Ali Kurt 528961

Thomas Kujawa 463620

Felix Hoff 366927

```
25
26
27
28 -- Funktion zum Anzeigen von rationalen Zahlen
29 -- Die Notation wird später in der Vorlesung erläutert
30 instance Show Ratio where
31     show (MkRatio nom den) = (show nom) ++ "/" ++ (show den)
32
33 ggT :: Integer -> Integer -> Integer
34 ggT a b = if b == 0 then a
35           else ggT b (mod a b)
36
37 myRatio :: Ratio
38 myRatio = MkRatio 2 3
39
40 reduceFraction :: Ratio -> Ratio
41 reduceFraction (MkRatio nom den) = MkRatio (div nom (ggT nom den)) (div den (ggT nom den))
42
43 fromRatio :: Ratio -> Float
44 fromRatio (MkRatio nom den) = roundFive (fromIntegral nom / fromIntegral den)
45
46 roundFive :: Float -> Float
47 roundFive f = fromIntegral(round (f*10^5))/(10^5)
48
49 decimalPlace :: Float -> Integer
50 decimalPlace f = truncate (roundFive f*10^5)
51
52 toRatio :: Float -> Ratio
53 toRatio f = reduceFraction (MkRatio (decimalPlace f) 100000)
54
```

Ali Kurt 528961
Thomas Kujawa 463620
Felix Hoff 366927

6

Informatik I - Gruppe 1 - Übungsblatt 6

Ausgabe: 14.11.2022

Abgabe: 21.11.1022

Tutor: Tim Völker

Ali Kurt 528961

Thomas Kujawa 463620

Felix Hoff 366927

```
1
2 {-# OPTIONS_GHC -Wno-unrecognised-pragmas #-}
3 {-# HLINT ignore "Use camelCase" #-}
4 --Datentyp BtreeFloat der nur Elemente vom Typ Float abspeichert
5 data BtreeFloat = Nil | Node Float BtreeFloat BtreeFloat
6   deriving (Eq, Show, Ord)
7
8 --Funktion die eine Gleitkommazahl in den Binärbaum einfügt. Für jeden Knoten wird der Wert s
9   ↳ mit dem Wert des aktuellen Knoten si verglichen. Ist s < si dann wird der Wert in den
10  ↳ linken Binärbaum eingefügt, ist s > si dann wird der Wert in den rechten Binärbaum
11  ↳ eingefügt. Sind beide Werte gleich wird der Wert nicht eingefügt. Der Wert s wird soweit
12  ↳ nach unten durchgereicht bis ein Blatt erreicht wird. Dort wird dann ein neuer Knoten mit
13  ↳ dem Wert s und zwei leeren Kindern (Binärbäumen) eingefügt
14
15 insert :: BtreeFloat -> Float -> BtreeFloat
16 insert Nil flt = Node flt Nil Nil
17 insert (Node val left right) flt
18   | flt < val = Node val (insert left flt) right
19   | otherwise = Node val left (insert right flt)
20
21 --Funktion die die Tiefe eines Binärbaumes angibt. Dabei ist die Tiefe beschrieben durch den
22   ↳ maximal längsten Pfad von der Wurzel bis zu einem Blatt
23
24 depth :: BtreeFloat -> Int
25 depth Nil = 0
26 depth (Node val left right)
27   | depth left > depth right = 1 + depth left
28   | depth left < depth right = 1 + depth right
29   | depth left == depth right = 1 + depth right
30   | otherwise = 1 + depth right
31
32 --Funktion die den größten Knotenwert innerhalb des Binärbaumes zurückgibt. Sollte die Eingabe
33   ↳ nur ein leerer Knoten sein, dann soll max_node den Wert 0 zurückgeben
34
35 max_node :: BtreeFloat -> Float
36 max_node Nil = 0
37 max_node (Node val Nil Nil) = val
38 max_node (Node val left right) = max_node right
39
40 --Funktion die die Differenz zwischen dem kürzesten und dem längsten Pfad im Binärbaum
41   ↳ zurückgibt. Ein Pfad sei dabei die Anzahl der Knoten vom Wurzelknoten (dem Startknoten) zu
42   ↳ einem Blattknoten
43
44 path_diff :: BtreeFloat -> Int
45 path_diff Nil = depth Nil - shortest_path Nil
46 path_diff (Node left val right) = depth (Node left val right) - shortest_path (Node left val
47   ↳ right)
```

Informatik I - Gruppe 1 - Übungsblatt 6

Ausgabe: 14.11.2022

Abgabe: 21.11.1022

Tutor: Tim Völker

Ali Kurt 528961

Thomas Kujawa 463620

Felix Hoff 366927

```
35  --Hilfsfunktion die die minimale Tiefe eines Binärbaumes angibt. Dabei ist die minimale Tiefe
    ↳ beschrieben durch den maximal kürzesten Pfad von der Wurzel bis zu einem Blatt
36  shortest_path :: BtreeFloat -> Int
37  shortest_path Nil = 1
38  shortest_path (Node val left right)
39      | shortest_path left > shortest_path right = 1 + shortest_path right
40      | shortest_path left < shortest_path right = 1 + shortest_path left
41      | shortest_path left == shortest_path right = 1 + shortest_path left
42      | otherwise = 1 + shortest_path left
43
44  --tree aus dem Beispiel
45  tree :: BtreeFloat
46  tree = Node 1 (Node 2 (Node 3 (Node 4 Nil Nil) (Node 5 (Node 6 Nil Nil) Nil)) Nil) (Node 7 Nil
    ↳ Nil)
```
