Aufgabe 1

a

zipwith (&&) gibt zu zwei Listen von Wahrheitswerten eine Liste von Wahrheitswerten aus, wobei die i-ten Stelle der Liste wahr ist g.d.w. an der i-ten Stelle beide Eingabelisten wahr sind.

```
zipWith (&&)
-+-----  +-
 |       +--> Bool -> Bool -> Bool
 |             |       |       |
 |       +-----+       |       |
 |      /  +-----------+       |
 |     /  / +------------------+
 |    |  |  |
 +-> (a->b->c) -> [a] -> [b] -> [c]
                  /       |       \
                 /        |        \
             [Bool] -> [Bool] -> [Bool]
```

b

map (==2) gibt zu einer Liste von Num eine Liste von Wahrheitswerten, wobei an der i-ten Stelle der Liste steh ob das i-te Element der Eingabeliste = 2 ist.

```
map (== 2)
-+-  -+--
 |    +--> Num a => a -> Bool
 |       +----------+     |
 |      / +---------------+
 |     / /
 +-> (a->b) -> [a] -> b
                |      \
      Num a => [a] -> [Bool]
```

c

*filter (\x -> (x*x==9))* Liefert zu einer Liste von Num eine Liste die nur noch die Elemente enthält, die gleich 3 oder -3 sind.

```
fiter (\x -> (x*x==9))
-+---  -+-----------
 |      +--> Num a => a -> Bool
 |       +------------+     |
 |      /   +---------------+
 |     /   /
 +-> (a->Bool) -> [a] -> [a]
                   |      |
         Num a => [a] -> [a]
```

d

*((\f g -> g.f)(\x->x*x)(\x->x+1)) 2* ist eine übersichtliche Darstellung der Zahl 5 (Typ Num a).

Leider läßt sich der Typ nicht mehr so übersichtlich als Baum darstellen, deshalb erstmal die Typen der Teilfunktionen:

```
((\f g -> g . f)(\x->x*x)(\x->x+1)) 2
  -+-----------  -+-----  -+-----   +
   |              |        |        +-> Num a => a
   |              |        +-> Num a => a -> a
   |              +-> Num a => a -> a
   +-> (a->b) -> (b->c)
```

Nach gründlichem Zählen der Applikationen oder cut'n paste in hugs sieht man leicht, daß sich der Ausdruck zu einer Konstanten vom Typ Num a auswerten muß.

Aufgabe 2

Definitionen:

```
(f 1) filter p []     = []
(f 2) filter p (x:xs) | p x       = x : filter p xs
(f 3)                 | otherwise =     filter p xs

(u3 ) p &&& q = \x (p x && q x)

(m 1) map f []     = []
(m 2) map f (x:xs) = f x : map f xs

(c 1) (f . g) x = f (g x)
```

a

Beweis der extensionalen Geichheit durch Induktion:

```
p, bzw. q haben als Signatur: p :: a -> Bool
a muessen fuer p und q vereinbare typen sein.

z.Z.: filter p (filter q) xs = filter (p &&& q) xs

IA: xs = []
    filter p (filter q [])
  = filter p []                            (f 1)
  = []                                     (f 1)

    filter (p &&& q [])                    (f 1)
  = []
```

```
IV: filter p (filter q xs) = filter (p &&& q) xs

IS: xs -> x:xs

Fall 1: p x == true, q x == true
    filter p (filter q x:xs)
  = filter p (x:(filter q xs))            (f 2)
  = x:filter p (filter q xs)              (f 2)
  = x:filter (p &&& q) xs                 (IV )

    filter (p &&& q) (x:xs)
  = x:filter (p &&& q) xs                 (f 1),(u 3)


Fall 2: p x == true, q x == false
    filter p (filter q (x:xs))
  = filter p (filter q xs)                (f 3)
  = filter (p &&& q) xs                   (IV )

    filter (p &&& q) (x:xs)
  = filter (p &&& q) xs                   (f 3),(u 3)

Fall 3: p x == false, q x == false
    filter p (filter q (x:xs))
  = filter p (filter q xs)                (f 3)
  = filter (p &&& q) xs                   (IV )

    filter (p &&& q) (x:xs)
  = filter (p &&& q) xs                   (f 3),(u 3)

Fall 4: p x == false, q x == true
    filter p (filter q (x:xs))
  = filter p (x:(filter q xs))            (f 2)
  = filter p (filter q xs)                (f 3)
  = filter (p &&& q) xs                   (IV )

    filter (p &&& q) (x:xs)
  = x:filter (p &&& q) xs                 (f 3),(u 3)
```

b

Beweis der extensionalen Geichheit durch Induktion:

```
f muss als Signatur f :: a -> b haben,
p                   p :: b -> Bool

z.Z.: (filter p . map f) xs = (map f . filter (p . f)) xs

IA: xs = []
    (filter p . map f) []
    filter (map f [])                   (c 1)
  = filter p                            (m 1)
  = []                                  (f 1)

    (map f . filter (p . f)) []
    map f (filter (p . f) [])           (c 1)
  = map f []                            (f 1)
  = []                                  (m 1)

IV: (filter p . map f) xs = map f . filter (p . f) xs

IS: xs -> x:xs

Fall 1: p (f x) == True
    (filter p . map f) (x:xs)
  = filter p (map f (x:xs))             (c 1)
  = filter p (f x : map f xs)           (m 2)
  = f x : filter p (map f xs)           (f 2)
  = f x : (filter p . map f) xs         (c 1) rückwärts
  = f x : map f . filter (p . f) xs     (IV )

    (map f . filter (p . f)) (x:xs)
  = map f (filter (p . f) (x:xs))       (c 1)
  = map f (x : filter (p . f) xs)       (f 2, vor)
  = f x : map f (filter (p . f) xs)     (m 2)
  = f x : (map f . filter (p . f)) xs   (c 1) rückwärts

Fall 2: p (f x) == false
    (filter p . map f) (x:xs)
  = filter p (map f (x:xs))             (c 1)
  = filter p (f x : map f xs)           (m 2)
  = filter p (map f xs)                 (f 3)
  = (filter p . map f) xs               (c 1) rückwärts
  = (map f . filter (p . f)) xs         (IV )

    (map f . filter (p . f)) (x:xs)
  = map f (filter (p . f) (x:xs))       (c 1)
  = map f (filter (p . f) xs)           (f 3, vor)
  = (map f . filter (p . f)) xs         (c 1) rückwärts
```

Aufgabe 3

a

Die einzige Bedingung an den Datentyp der Liste ist, daß zwei Elemente bezüglich Gleichheit prüfbar sind.

```
> anzahl  :: Eq a => a -> [a] -> Int
> anzahl x = length.filter (== x)

testruns:
        Main> anzahl 'z' "tztztztztztzt"
        6
        (104 reductions, 158 cells)
        Main> anzahl 5 [1,5,1,5,1]
        2
        (60 reductions, 91 cells)
        Main> anzahl 1 [1,5,1,5,1]
        3
        (65 reductions, 75 cells)
        Main> anzahl 7 []
        0
```

b

Auch hier ist wieder die einzige Bedingung, daß zwei Elemente bezüglich Gleichheit prüfbar sind.

```
> ueberein :: Eq a => [a] -> [a] -> [Bool]
> ueberein  = zipWith (==)

testruns:
        Main> ueberein "Hallo Welt" "Hello World"
        [True,False,True,True,True,True,True,False,False,False]
        (154 reductions, 408 cells)
        Main> ueberein [1..] [1..]
        [True,True,True,True,True,True,True,True,True,True,True,True,
         True,True,True,True,True,True,True,True,True,True,True,True,
         True,True,True,True,True,True,True,True,True,True,True,True,
         True,True,True,True,True,True,True,True,True,True,True,True,
         True,True,True,True^C
         {Interrupted!}
```