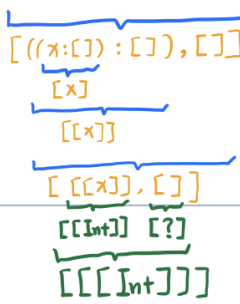
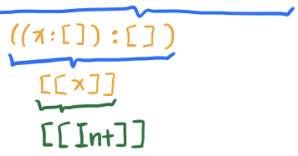
Xiaoting Wang, 406267
Wensheng Zhang, 405521

Aufgabe 5

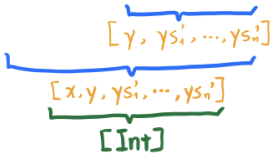
a) `((x:[]) : []) : [] = ((x : []) : []) ++ []`



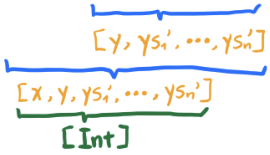


Also ist die Gleichung nicht richtig.

b) `[x] ++ (y : ys) = x : ([y] ++ ys)`



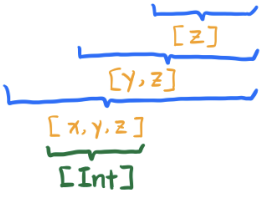
Die Länge ist 2+m



Die Länge ist 2+m.

Die Gleichung ist typkorrekt.

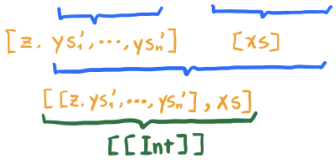
c) `(x : (y : (z : []))) = (x : y) : (z : [])`



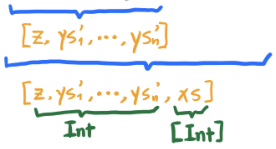
falsch, weil y kein Liste ist, x kann nicht als erstes Element in y einfügen.

Die Länge ist 3.

d) `(z : ys) : (xs : []) = [[z] ++ ys] ++ [xs]`

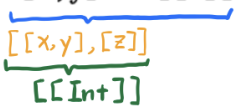


Die Länge ist 2.

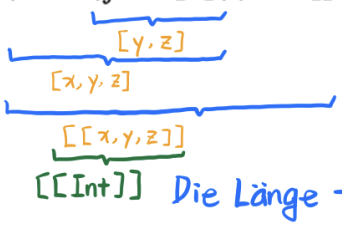


Falsch, weil die Elemente aus dieselber Liste gleiche Typ haben sollen.

e) `[x,y] : [[z]] = (x : (y : [z])) : []`



Die Länge → 2



Die Länge → 1

Obwohl die beide Typen passt, ist die Länge nicht gleich.
Also ist die Gleichung nicht richtig.

Xiaoting Wang, 406267
Wensheng Zhang, 405521

Aufgabe 8

```haskell
-- a)
isMatrix :: [[Int]] -> Bool
isMatrix [] = True
isMatrix [x] = True
isMatrix (x:y:xs) | length x == length y = isMatrix (x:xs)
                  | otherwise            = False

-- b)
dimensions :: [[Int]] -> (Int, Int)
dimensions (x:xs) | not (isMatrix (x:xs)) = (-1, -1)
                  | otherwise             = (zeilen, spalten)
    where zeilen  = length (x:xs)
          spalten = length x

-- c)
isQuadratic :: [[Int]] -> Bool
isQuadratic (x:xs) | (length (x:xs))==(length x) = True
           | otherwise                   = False

-- d)
getRow :: [[Int]] -> Int -> [Int]
getRow xs i | isMatrix xs = xs !! i
            | otherwise   = []

getCol :: [[Int]] -> Int -> [Int]
getCol [] _   = []
getCol (x:xs) i | not (isMatrix (x:xs)) = []
        | otherwise             = (x !! i) : (getCol xs i)

-- e)
trac :: [[Int]] -> [[Int]]
trac (x:xs) = help (x:xs) ((length x)-1)
    where help :: [[Int]] -> Int -> [[Int]]
          help ys (-1) = []
          help ys  n   = (help ys (n-1)) ++ [(getCol ys n)]

-- f)
setEntry :: [[Int]] -> Int -> Int -> Int -> [[Int]]
setEntry (x:xs) 0 j aij = (help x j aij) : xs
    where help :: [Int] -> Int -> Int -> [Int]
          help (y:ys) n aij  | n==0      = aij:ys
                  | otherwise = y:(help ys (n-1) aij)
setEntry (x:xs) i j aij = x : (setEntry xs (i-1) j aij)
```