8. Aufgabenblatt vom Samstag, den 14. Dezember 2019 zur Vorlesung

ALP I: Funktionale Programmierung

Bearbeiter: A. Rudolph und F. Formanek
Tutor: Stephanie Hoffmann
Tutorium 06

Abgabe: bis Montag, den 06. Januar 2020, 10:10 Uhr

1. **Aufgabe** (24 Punkte)

   (a) *Behauptung: reverse(reverse xs) = xs*
   Induktion über Liste xs der Länge n

   Induktionsanfang: xs = [ ]
   reverse(reverse [ ]) $\overset{rev.1}{=}$ reverse [ ] $\overset{rev.1}{=}$ [ ]

   Induktionsvorraussetzung: für xs = xs' gilt:
   reverse(reverse xs') = xs'

   Indukionsschritt: Sei xs = (x:xs')
   reverse(reverse (x:xs')) $\overset{rev.2}{=}$ reverse(reverse xs' ++ [x])
   ≡ (reverse [x]) ++ reverse (reverse xs')
   $\overset{rev.2}{\leftrightarrow}$ (reverse ([ ]) ++ [x]) ++ reverse(reverse xs') $\overset{rev.1}{\leftrightarrow}$
   ([ ] ++ [x]) ++ reverse(reverse xs') $\overset{(++).1}{=}$
   [x] ++ reverse(reverse xs') $\overset{nachIV}{=}$
   [x] ++ xs' ≡ (x:xs')

   Das bedeutet, dass die Behauptung für alle xs (endliche Listen) gilt.

   (b) *Behauptung: reverse(xs ++ ys) = reverse ys ++ reverse xs*

   Induktionsanfang: xs = [ ]
   reverse([] ++ ys) = reverse ys ++ reverse [] $\overset{rev.1}{=}$
   reverse([] ++ ys) = reverse ys ++ [] $\overset{(++).1}{=}$
   reverse ys = reverse ys

   Induktionsvorraussetzung: für xs = xs' gilt:
   reverse(xs' ++ ys) = reverse ys ++ reverse xs'

   Indukionsschritt: Sei xs = (x:xs')
   reverse((x:xs') ++ ys) = reverse ys ++ reverse(x:xs') $\overset{rev.2}{=}$

reverse((x:xs') ++ ys) = reverse ys ++ (reverse xs' ++ [x]) $\overset{(++).2}{=}$
reverse(x:(xs'++ys)) = reverse ys ++ (reverse xs' ++ [x]) $\overset{rev.2}{=}$
reverse(xs'++ys) ++ [x] = reverse ys ++ (reverse xs' ++[x]) $\overset{nachIV}{=}$
reverse ys ++ reverse xs' ++ [x] = reverse ys ++ (reverse xs' ++ [x]) $\equiv$
reverse ys ++ reverse xs' ++ [x] = reverse ys ++ reverse xs' ++ [x]

Das bedeutet, dass die Behauptung für alle xs (endliche Listen) gilt.

(c) *Behauptung: elem a (xs ++ ys) = elem a xs || elem a ys*

Induktionsanfang: xs = [ ]
elem a ([ ] ++ ys) = elem a [ ] || elem a ys $\overset{(++).1}{=}$
elem a ys = elem a [] || elem a ys $\overset{elem.1}{=}$
elem a ys = False || elem a ys $\equiv$
elem a ys = elem a ys

Induktionsvorraussetzung: für xs = xs' gilt:
elem a (xs' ++ ys) = elem a xs' || elem a ys

Indukionsschritt: Sei xs = (x:xs')
elem a ((x:xs') ++ ys) = elem a (x:xs') || elem a ys $\overset{(++).1}{=}$
elem a (x:(xs' ++ ys)) = elem a (x:xs') || elem a ys $\overset{elem.3}{=}$
elem a (x:(xs' ++ ys)) = elem a ys || elem a ys $\equiv$
elem a (x:(xs' ++ ys)) = elem a ys $\equiv$
elem a [x] || elem a (xs' ++ ys) = elem a ys $\overset{nachIV}{=}$
elem a [x] || (elem a xs' || elem a ys) = elem a ys $\equiv$
elem a (x:[]) || (elem a xs' || elem a ys) = elem a ys $\overset{elem.1}{=}$
False || (elem a xs' || elem a ys) = elem a ys $\equiv$
elem a xs' || elem a ys = elem a ys

TODO: Überarbeiten. Ergebnis ist falsch

Das bedeutet, dass die Behauptung für alle xs (endliche Listen) gilt.

(d) *Behauptung: (takeWhile p xs) ++ (dropWhile p xs) = xs*

Induktionsanfang: xs = [ ]
(takeWhile p [ ]) ++ (dropWhile p [ ]) = [ ] $\overset{takeW.1}{=}$
[ ] ++ (dropWhile p [ ]) = [ ] $\overset{dropW.1}{=}$
[ ] ++ [ ] = [ ] $\overset{(++).1}{=}$

[ ] = [ ]

Induktionsvorraussetzung: für xs = xs' gilt:
(takeWhile p xs') ++ (dropWhile p xs') = xs'

Indukionsschritt: Sei xs = (x:xs')
(takeWhile p (x:xs')) ++ (dropWhile p (x:xs')) = (x:xs') $\overset{takeW.2}{=}$
(x:(takeWhile p xs')) ++ (dropWhile p (x:xs')) = (x:xs') $\overset{dropW.2}{=}$
(x:(takeWhile p xs')) ++ (dropWhile p xs') = (x:xs') $\overset{(++).2}{=}$
x:((takeWhile p xs') ++ (dropWhile p xs') = (x:xs') $\overset{nachIV}{=}$
(x:xs') = (x:xs')

Das bedeutet, dass die Behauptung für alle xs (endliche Listen) gilt.

(e) *Behauptung: map (f . g) xs = map f xs . map g xs*

Induktionsanfang: xs = [ ]
map (f . g) [ ] = map f [ ] . map g [ ] $\overset{map.1}{=}$
[ ] = [ ]

Induktionsvorraussetzung: für xs = xs' gilt:
map (f . g) xs' = map f xs' . map g xs'

Indukionsschritt: Sei xs = (x:xs')
map (f . g) (x:xs') = map f (x:xs') . map g (x:xs') $\overset{map.2}{=}$
g(f(x)):map(f . g) xs' = g(f(x)):(map f xs' . map g xs') $\overset{nachIV}{=}$
g(f(x)):(map f xs' . map g xs') = g(f(x)):(map f xs' . map g xs')

Das bedeutet, dass die Behauptung für alle xs (endliche Listen) gilt.

(f) *Behauptung: map f . concat = concat . map(map f)*

Induktionsanfang: xs = [ ]
map f . concat [] = concat . map(map f) [] $\overset{map.1}{=}$
concat [ ] = concat . map(map f) [ ] $\overset{concat.1}{=}$
concat [ ] = (foldr (++) [ ] [ ]) . map(map f) $\overset{foldr.1}{=}$
concat [ ] = [ ] . map(map f) $\overset{map.1}{=}$
concat [ ] = [ ] $\overset{concat.1}{=}$
[ ] = [ ]

Induktionsvorraussetzung: für xs = xs' gilt:

map f . concat xs' = concat . map(map f) xs'

Indukionsschritt: Sei xs = (x:xs')

map f . concat (x:xs') = concat . map(map f) (x:xs') $\overset{map.2}{=}$
concat(f(x):map f xs') = concat . map(map f) (x:xs')
TO BE CONTINUED

Das bedeutet, dass die Behauptung für alle xs (endliche Listen) gilt.

2. *Behauptung: length(powset xs)* $= 2^{(length(xs))}$

   Induktionsanfang: xs = [ ]
   length(powset [ ]) = $2^{(length([])}$ $\overset{e.1}{=}$
   length(powset [ ]) = 2^1 $\overset{powset.1}{=}$
   length([[ ]]) = 2^1 $\overset{e.1}{=}$
   $1 = 2^1 \equiv$
   $1 = 1$

   Induktionsvorraussetzung: für xs = xs' gilt:
   length(powset xs') = $2^{(length(xs'))}$

   Indukionsschritt: Sei xs = (x:xs')
   length(powset (x:xs')) = $2^{(length(x:xs'))}$ $\overset{powset.2}{=}$
   length(powset xs' ++ [x:ys | *ys* <- powset xs']) = $2^{(length(x:xs'))}$ $\equiv$
   length(powset xs') + (length [x:ys | *ys* <- powset xs']) = $2^{(length(x:xs'))}$ $\overset{nachIV}{=}$
   $2^{(length(xs'))}$ + (length [x:ys | *ys* <- powset xs']) = $2^{(length(x:xs'))}$ $\overset{e.1}{=}$
   $2^{(length(xs'))}$ + (length(powset xs') + 1) = $2^{(length(x:xs'))}$ $\overset{nachIV}{=}$
   $2^{(length(xs'))} + 2^{((length(xs'))+1)} = 2^{(length(x:xs'))} \equiv$
   $2^{(length(xs')} * 2^1 = 2^{(length(xs')} * 2^1$

   Das bedeutet, dass die Behauptung für alle xs (endliche Listen) gilt.

3. *Behauptung: sumLeaves t = sumNodes t + 1*

   Induktionsanfang: Sei t = (Leaf x)
   sumLeaves (Leaf x) = sumNodes (Leaf x) + 1 $\overset{sumLeaves.1}{=}$
   1 = sumNodes (Leaf x) + 1 $\overset{sumNodes.1}{=}$
   $1 = 0 + 1 \equiv$
   $1 = 1$

   Induktionsvorraussetzung: für t = (Node x lt rt) gilt:

sumLeaves lt = sumNodes lt + 1
und
sumLeaves rt = sumNodes rt + 1

Indukionsschritt: Sei t = (Node x lt rt)
sumLeaves (Node x lt rt) = sumNodes (Node x lt rt) + 1 $\stackrel{sumLeaves.2}{=}$
sumLeaves lt + sumLeaves rt = sumNodes (Node x lt rt) + 1 $\stackrel{sumNodes.2}{=}$
sumLeaves lt + sumLeaves rt = 1 + sumNodes lt + sumNodes rt + 1 $\stackrel{nachIV}{=}$
sumNodes lt + 1 + sumNodes rt + 1 = 1 + sumNodes lt + sumNodes rt + 1 $\equiv$
sumNodes lt + sumNodes rt + 2 = sumNodes lt + sumNodes rt + 2

Das bedeutet, dass die Behauptung für alle t (endliche Binärbäume) gilt.