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*
   Induktionsanfang: xs = [ ]

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

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

   Indukionsschritt: Sei xs = (x:xs')

$$
\begin{aligned}
& reverse(reverse(x : xs')) && = (x : xs') \\
\overset{rev.2}{\equiv}\ & reverse(reversexs' ++[x]) && = (x : xs') \\
\equiv\ & (reverse[x]) ++reverse(reversexs') && = (x : xs') \\
\overset{rev.2}{\equiv}\ & (reverse[] ++[x]) ++reverse(reversexs') && = (x : xs') \\
\overset{rev.1}{\equiv}\ & ([] ++[x]) ++reverse(reversexs') && = (x : xs') \\
\overset{(++).1}{\equiv}\ & [x] ++reverse(reversexs') && = (x : xs') \\
\overset{nachIV}{\equiv}\ & [x] ++xs' && = (x : xs') \\
\equiv\ & (x : xs') && = (x : xs')
\end{aligned}
$$

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

(b) *Behauptung: reverse(xs ++ ys) = reverse ys ++ reverse xs*
Induktionsanfang: xs = [ ]

$$\begin{aligned} & reverse([]++ys) = reverse\ ys\ ++ reverse\ [] \\ \overset{rev.1}{\equiv} & reverse([]++ys) = reverse\ ys\ ++ [] \\ \overset{(++).1}{\equiv} & reverse\ ys \qquad\quad = reverse\ ys \end{aligned}$$

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

Indukionsschritt: Sei xs = (x:xs')

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

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 = []

$$\begin{aligned} & elem\ a\ ([]\ ++ys) = elem\ a\ []\ ||\ elem\ a\ ys \\ \overset{(++).1}{\equiv} & elem\ a\ ys \qquad = elem\ a\ []\ ||\ elem\ a\ ys \\ \overset{elem.1}{\equiv} & elem\ a\ ys \qquad = False \quad ||\ elem\ a\ ys \\ \equiv\ & elem\ a\ ys \qquad = elem\ a\ ys \end{aligned}$$

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

Indukionsschritt: Sei xs = (x:xs')

Fall 1 (x immer ungleich y):

$$
\begin{aligned}
& elem\ a\ ((x:xs')\ ++\ ys) && = elem\ a\ (x:xs')\ ||\ elem\ a\ ys \\
\overset{(++).2}{\equiv}\ & elem\ a\ (x:(xs'\ ++\ ys)) && = elem\ a\ (x:xs')\ ||\ elem\ a\ ys \\
\overset{elem.3}{\equiv}\ & elem\ a\ (x:(xs'\ ++\ ys)) && = elem\ a\ ys \qquad ||\ elem\ a\ ys \\
\equiv\ & elem\ a\ (x:(xs'\ ++\ ys)) && = elem\ a\ ys \\
\equiv\ & elem\ a\ [x]\ ||\ elem\ a\ (xs'\ ++\ ys) && = elem\ a\ ys \\
\overset{nachIV}{\equiv}\ & elem\ a\ [x]\ ||\ (elem\ a\ xs'\ ||\ elem\ a\ ys) && = elem\ a\ ys \\
\equiv\ & elem\ a\ (x:[])\ ||\ (elem\ a\ xs'\ ||\ elem\ a\ ys) && = elem\ a\ ys \\
\overset{elem.1}{\equiv}\ & False\ ||\ (elem\ a\ xs'\ ||\ elem\ a\ ys) && = elem\ a\ ys \\
\equiv\ & elem\ a\ xs'\ ||\ elem\ a\ ys && = elem\ a\ ys
\end{aligned}
$$

Liefert kein eindeutiges Ergebnis

Fall 2 (x==y):

$$
\begin{aligned}
& elem\ a\ ((x:xs')\ ++\ ys) && = elem\ a\ (x:xs')\ ||\ elem\ a\ ys \\
\overset{(++).2}{\equiv}\ & elem\ a\ (x:(xs'\ ++\ ys)) && = elem\ a\ (x:xs')\ ||\ elem\ a\ ys \\
\overset{elem.2}{\equiv}\ & elem\ a\ (x:(xs'\ ++\ ys)) && = True \qquad ||\ elem\ a\ ys \\
\equiv\ & elem\ a\ (x:(xs'\ ++\ ys)) && = True \\
\equiv\ & elem\ a\ [x]\ ||\ elem\ a\ (xs'\ ++\ ys) && = True \\
\overset{elem.2}{\equiv}\ & True\ ||\ elem\ a\ (xs'\ ++\ ys) && = True \\
\equiv\ & True && = True
\end{aligned}
$$

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

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

$$
\begin{aligned}
& (takeWhile\ p\ [])\ ++\ (dropWhile\ p\ []) && = [] \\
\overset{takeW.1}{\equiv}\ & []\ ++\ (dropWhile\ p\ []) && = [] \\
\overset{dropW.1}{\equiv}\ & []\ ++\ [] && = [] \\
\overset{(++).1}{\equiv}\ & [] && = []
\end{aligned}
$$

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

Indukionsschritt: Sei xs = (x:xs')

$$
\begin{aligned}
& (takeWhile\ p\ (x:xs'))\ ++\ (dropWhile\ p\ (x:xs')) = (x:xs') \\
\overset{takeW.2}{\equiv} & (x:(takeWhile\ p\ xs'))\ ++\ (dropWhile\ p\ (x:xs')) = (x:xs') \\
\overset{dropW.2}{\equiv} & (x:(takeWhile\ p\ xs'))\ ++\ (dropWhile\ p\ xs') \qquad = (x:xs') \\
\overset{(++).2}{\equiv} & x:((takeWhile\ p\ xs')\ ++\ (dropWhile\ p\ xs') \qquad = (x:xs') \\
\overset{nachIV}{\equiv} & (x:xs') \qquad = (x:xs')
\end{aligned}
$$

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 = [ ]

$$
\begin{aligned}
& map\ (f\ .\ g)\ [] = map\ f\ []\ .\ map\ g\ [] \\
\overset{map.1}{\equiv} & [] \qquad = []
\end{aligned}
$$

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

Indukionsschritt: Sei xs = (x:xs')

$$
\begin{aligned}
& map\ (f\ .\ g)\ (x:xs') \qquad = map\ f\ (x:xs')\ .\ map\ g\ (x:xs') \\
\overset{map.2}{\equiv} & g(f(x)) : map(f\ .\ g)\ xs' \qquad = g(f(x)) : (map\ f\ xs'\ .\ map\ g\ xs') \\
\overset{nachIV}{\equiv} & g(f(x)) : (map\ f\ xs'\ .\ map\ g\ xs') = g(f(x)) : (map\ f\ xs'\ .\ map\ g\ xs')
\end{aligned}
$$

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

(f) *Behauptung: map f . concat = concat . map(map f)*
Induktionsanfang: xs = [ ]

$$
\begin{aligned}
map\ f\ .\ concat\ [] &= concat\ .\ map(map\ f)\ [] \\
\overset{map.1}{\equiv} concat\ [] &= concat\ .\ map(map\ f)\ [] \\
\overset{concat.1}{\equiv} concat\ [] &= (foldr\ (++)\ []\ [])\ .\ map(map\ f) \\
\overset{foldr.1}{\equiv} concat\ [] &= []\ .\ map(map\ f) \\
\overset{map.1}{\equiv} concat\ [] &= [] \\
\overset{concat.1}{\equiv} [] &= []
\end{aligned}
$$

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

Indukionsschritt: Sei xs = (x:xs')
/

2. **Aufgabe** (4 Punkte)
Behauptung: length(powset xs) $= 2^{(length(xs))}$

Induktionsanfang: xs = [ ]

$$
\begin{aligned}
length(powset\ []) &= 2^{(length\ [])} \\
\overset{e.1}{\equiv} length(powset\ []) &= 2^1 \\
\overset{pow.1}{\equiv} length([[]]) &= 2^1 \\
\overset{e.1}{\equiv} 1 &= 2^1 \\
\equiv\ 1 &= 1
\end{aligned}
$$

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

Indukionsschritt: Sei xs = (x:xs')

$$
\begin{array}{rll}
 & length(powset\ (x : xs')) & = 2^{(length(x:xs'))} \\
\overset{pow.2}{\equiv} & length(powset\ xs'\ ++\ [x : ys \mid ys \leftarrow powset\ xs']) & = 2^{(length(x:xs'))} \\
\equiv & length(powset\ xs')\ +\ (length\ [x : ys \mid ys \leftarrow powset\ xs']) & = 2^{(length(x:xs'))} \\
\overset{nachIV}{\equiv} & 2^{(length\ xs')}\ +\ (length\ [x : ys \mid ys \leftarrow powset\ xs']) & = 2^{(length(x:xs'))} \\
\overset{e.1}{\equiv} & 2^{(length\ xs')}\ +\ (length(powset\ xs')\ +\ 1) & = 2^{(length(x:xs'))} \\
\overset{nachIV}{\equiv} & 2^{(length\ xs')}\ +\ 2^{((length\ xs')\ +\ 1)} & = 2^{(length(x:xs'))} \\
\equiv & 2^{(length\ xs')}\ *\ 2^1 & = 2^{(length\ xs')}\ *\ 2^1
\end{array}
$$

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

3. **Aufgabe** (4 Bonuspunkte)
Behauptung: sumLeaves t = sumNodes t + 1

Induktionsanfang: Sei t = (Leaf x)

$$
\begin{array}{rll}
 & sumLeaves\ (Leaf\ x) & = sumNodes\ (Leaf\ x)\ +\ 1 \\
\overset{sumL.1}{\equiv} & 1 & = sumNodes\ (Leaf\ x)\ +\ 1 \\
\overset{sumN.1}{\equiv} & 1 & = 0\ +\ 1 \\
\equiv & 1 & = 1
\end{array}
$$

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)

$$
\begin{array}{rll}
 & sumLeaves\ (Node\ x\ lt\ rt) & = sumNodes\ (Node\ x\ lt\ rt)\ +\ 1 \\
\overset{sumL.2}{\equiv} & sumLeaves\ lt\ +\ sumLeaves\ rt & = sumNodes\ (Node\ x\ lt\ rt)\ +\ 1 \\
\overset{sumN.2}{\equiv} & sumLeaves\ lt\ +\ sumLeaves\ rt & = 1\ +\ sumNodes\ lt\ +\ sumNodes\ rt\ +\ 1 \\
\overset{nachIV}{\equiv} & sumNodes\ lt\ +\ 1 +\ sumNodes\ rt\ +\ 1 & = 1\ +\ sumNodes\ lt\ +\ sumNodes\ rt\ +\ 1 \\
\equiv & sumNodes\ lt\ +\ sumNodes\ rt\ +\ 2 & = sumNodes\ lt\ +\ sumNodes\ rt\ +\ 2
\end{array}
$$

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

4. **Aufgabe** (6 Bonuspunkte)
Behauptung: sum . tree2List = sumTree

Induktionsanfang 1.1: Sei t = Nil

$$\begin{aligned} & sum\ .\ tree2List\ Nil = sumTree\ Nil \\ \overset{sumT.1}{\equiv}\ & sum\ .\ tree2List\ Nil = 0 \\ \overset{t2L.1}{\equiv}\ & sum\ [] \qquad\qquad = 0 \\ \overset{sum.1}{\equiv}\ & 0 \qquad\qquad\qquad = 0 \end{aligned}$$

Induktionsanfang 1.2: Sei t = (Leaf x)

$$\begin{aligned} & sum\ .\ tree2List\ (Leaf\ x) = sumTree\ (Leaf\ x) \\ \overset{sumT.2}{\equiv}\ & sum\ .\ tree2List\ (Leaf\ x) = x \\ \overset{t2L.2}{\equiv}\ & sum\ [x] = x \\ \equiv\ & sum(x : []) = x \\ \overset{sum.2}{\equiv}\ & x\ +\ sum\ [] = x \\ \overset{sum.1}{\equiv}\ & x\ +\ 0 = x \\ \equiv\ & x = x \end{aligned}$$

Induktionsvorraussetzung: für t = (Node x lt rt) gilt:
sum . tree2List lt = sumTree lt
und
sum . tree2List rt = sumTree rt

Indukionsschritt: Sei t = (Node x lt rt)

$$\begin{aligned} & sum\ .\ list2Tree\ (Node\ x\ lt\ rt) = sumTree\ (Node\ x\ lt\ rt) \\ \overset{sumT.3}{\equiv}\ & sum\ .\ list2Tree\ (Node\ x\ lt\ rt) = x\ +\ sumTree\ lt\ +\ sumTree\ rt \\ \overset{l2T.3}{\equiv}\ & sum(tree2List\ lt\ ++\ [x]\ ++\ tree2List\ rt) = x\ +\ sumTree\ lt\ +\ sumTree\ rt \\ \equiv\ & sum\ [x]\ +\ (sum(tree2List\ lt))\ +\ (sum(tree2List\ rt)) = x\ +\ sumTree\ lt\ +\ sumTree\ rt \\ \overset{nachIV}{\equiv}\ & sum\ [x]\ +\ sumTree\ lt\ +\ sumTree\ rt = x\ +\ +sumTree\ lt\ +\ sumTree\ rt \\ \overset{nachIA}{\equiv}\ & x\ +\ sumTree\ lt\ +\ sumTree\ rt = x\ +\ sumTree\ lt\ +\ sumTree\ rt \end{aligned}$$

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