

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:*  $\text{reverse}(\text{reverse } xs) = xs$

**Induktionsanfang:**  $xs = []$

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

**Induktionsvoraussetzung:** für  $xs = xs'$  gilt:

$$\text{reverse}(\text{reverse } xs') = xs'$$

**Induktionsschritt:** Sei  $xs = (x:xs')$

$$\begin{aligned} & \text{reverse}(\text{reverse}(x : xs')) &&= (x : xs') \\ & \stackrel{\text{rev.2}}{=} \text{reverse}(\text{reverse } xs' ++ [x]) &&= (x : xs') \\ & \equiv (\text{reverse } [x]) ++ \text{reverse}(\text{reverse } xs') &&= (x : xs') \\ & \stackrel{\text{rev.2}}{=} (\text{reverse } [] ++ [x]) ++ \text{reverse}(\text{reverse } xs') &&= (x : xs') \\ & \stackrel{\text{rev.1}}{=} ([ ] ++ [x]) ++ \text{reverse}(\text{reverse } xs') &&= (x : xs') \\ & \stackrel{(++).1}{=} [x] ++ \text{reverse}(\text{reverse } xs') &&= (x : xs') \\ & \stackrel{\text{nach IV}}{=} [x] ++ xs' &&= (x : xs') \\ & \equiv (x : xs') &&= (x : xs') \end{aligned}$$

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

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

**Induktionsanfang:**  $xs = []$

$$\begin{aligned} reverse([] ++ ys) &= reverse\ ys ++ reverse\ [] \\ &\stackrel{rev.1}{=} reverse([] ++ ys) = reverse\ ys ++ [] \\ &\stackrel{(++).1}{=} reverse\ ys = reverse\ ys \end{aligned}$$

**Induktionsvoraussetzung:** für  $xs = xs'$  gilt:

$$reverse(xs' ++ ys) = reverse\ ys ++ reverse\ xs'$$

**Induktionsschritt:** Sei  $xs = (x:xs')$

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

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

(c) *Behauptung:*  $elem\ a\ (xs ++ ys) = elem\ a\ xs \parallel elem\ a\ ys$

**Induktionsanfang:**  $xs = []$

$$\begin{aligned} elem\ a\ ([] ++ ys) &= elem\ a\ [] \parallel elem\ a\ ys \\ &\stackrel{(++).1}{=} elem\ a\ ys = elem\ a\ [] \parallel elem\ a\ ys \\ &\stackrel{elem.1}{=} elem\ a\ ys = False \parallel elem\ a\ ys \\ &\equiv elem\ a\ ys = elem\ a\ ys \end{aligned}$$

**Induktionsvoraussetzung:** für  $xs = xs'$  gilt:

$$elem\ a\ (xs' ++ ys) = elem\ a\ xs' \parallel elem\ a\ ys$$

**Induktionsschritt:** Sei  $xs = (x:xs')$

**Fall 1 (x immer ungleich y):**

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

**Liefert kein eindeutiges Ergebnis**

**Fall 2 (x==y):**

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

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

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

**Induktionsanfang:**  $xs = []$

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

**Induktionsvoraussetzung:** für  $xs = xs'$  gilt:  
 $(takeWhile\ p\ xs') ++ (dropWhile\ p\ xs') = xs'$

**Induktionsschritt:** Sei  $xs = (x:xs')$

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

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

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

**Induktionsanfang:**  $xs = []$

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

**Induktionsvoraussetzung:** für  $xs = xs'$  gilt:  
 $map\ (f . g)\ xs' = map\ f\ xs' . map\ g\ xs'$

**Induktionsschritt:** Sei  $xs = (x:xs')$

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

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

(f) *Behauptung:*  $\text{map } f . \text{concat} = \text{concat} . \text{map}(\text{map } f)$

**Induktionsanfang:**  $\text{xs} = []$

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

**Induktionsvoraussetzung:** für  $\text{xs} = \text{xs}'$  gilt:  
 $\text{map } f . \text{concat } \text{xs}' = \text{concat} . \text{map}(\text{map } f) \text{xs}'$

**Induktionsschritt:** Sei  $\text{xs} = (\text{x}:\text{xs}')$   
 /

## 2. Aufgabe (4 Punkte)

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

**Induktionsanfang:**  $\text{xs} = []$

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

**Induktionsvoraussetzung:** für  $\text{xs} = \text{xs}'$  gilt:  
 $\text{length}(\text{powset } \text{xs}') = 2^{\text{length}(\text{xs}')}$

**Induktionsschritt:** Sei  $xs = (x:xs')$

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

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

### 3. Aufgabe (4 Bonuspunkte)

Behauptung:  $\text{sumLeaves } t = \text{sumNodes } t + 1$

**Induktionsanfang:** Sei  $t = (\text{Leaf } x)$

$$\begin{aligned}
& \text{sumLeaves } (\text{Leaf } x) = \text{sumNodes } (\text{Leaf } x) + 1 \\
& \stackrel{\text{sumL.1}}{\equiv} 1 &= \text{sumNodes } (\text{Leaf } x) + 1 \\
& \stackrel{\text{sumN.1}}{\equiv} 1 &= 0 + 1 \\
& \equiv 1 &= 1
\end{aligned}$$

**Induktionsvoraussetzung:** für  $t = (\text{Node } x \text{ lt } \text{rt})$  gilt:

$\text{sumLeaves } \text{lt} = \text{sumNodes } \text{lt} + 1$

und

$\text{sumLeaves } \text{rt} = \text{sumNodes } \text{rt} + 1$

**Induktionsschritt:** Sei  $t = (\text{Node } x \text{ lt } \text{rt})$

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

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

4. **Aufgabe** (6 Bonuspunkte)

*Behauptung:*  $sum . tree2List = sumTree$

**Induktionsanfang 1.1:** Sei  $t = Nil$

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

**Induktionsanfang 1.2:** Sei  $t = (Leaf\ x)$

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

**Induktionsvoraussetzung:** für  $t = (Node\ x\ lt\ rt)$  gilt:

$sum . tree2List\ lt = sumTree\ lt$

und

$sum . tree2List\ rt = sumTree\ rt$

**Induktionsschritt:** Sei  $t = (Node\ x\ lt\ rt)$

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

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