

## 4. Übung - Programmierung

Induktive Beweise und Lambda-Kalkül

SS 18

# Induktive Beweise

## Prinzip

- ▶ gegeben sei eine **geordnete** Menge und eine Behauptung
  - ▶ **Induktionsanfang**: zeige Behauptung für ein kleinstmögliches Element " $0$ "
  - ▶ **Induktionsvoraussetzung/-hypothese**: Nimm an, Behauptung gilt für ein "beliebiges aber festes" Element " $n$ "
  - ▶ **Induktionsschritt**: Zeige, dass dann Behauptung für dessen Nachfolger " $n+1$ " gilt
- ⇒ Behauptung muss für alle gelten, die "größer" sind als " $0$ "

# Induktion über Listen

ZZ:  $\text{sum}(\text{foo } xs) = 2 * \text{sum } xs - \text{length } xs, \forall xs :: [Int]$

► **I.A.:** Sei  $x_0 = []$   
           $\text{sum}(\text{foo } [])$

# Induktion über Listen

ZZ:  $\text{sum}(\text{foo } xs) = 2 * \text{sum } xs - \text{length } xs, \forall xs :: [Int]$

► **I.A.:** Sei  $x_0 = []$   
 $\text{sum}(\text{foo } [])$   
 $\stackrel{Z2}{=} \text{sum } []$

# Induktion über Listen

ZZ:  $\text{sum}(\text{foo } xs) = 2 * \text{sum } xs - \text{length } xs, \forall xs :: [Int]$

► **I.A.:** Sei  $x_0 = []$   
     $\text{sum}(\text{foo } [])$   
     $\xrightarrow{Z2} \text{sum } []$   
     $\xrightarrow{Z6} 0$

# Induktion über Listen

ZZ:  $\text{sum}(\text{foo } xs) = 2 * \text{sum } xs - \text{length } xs, \forall xs :: [Int]$

► **I.A.:** Sei  $x_0 = []$   
     $\text{sum}(\text{foo } [])$   
     $\stackrel{Z2}{=} \text{sum } []$   
     $\stackrel{Z6}{=} 0$   
     $= 2 * 0 - 0$

# Induktion über Listen

ZZ:  $\text{sum}(\text{foo } xs) = 2 * \text{sum } xs - \text{length } xs, \forall xs :: [Int]$

► **I.A.:** Sei  $x_0 = []$

$$\begin{aligned} & \text{sum}(\text{foo } []) \\ & \stackrel{Z2}{=} \text{sum } [] \\ & \stackrel{Z6}{=} 0 \\ & = 2 * 0 - 0 \\ & \stackrel{Z6,10}{=} 2 * \text{sum } [] - \text{length } [] \end{aligned}$$

# Induktion über Listen

ZZ:  $\text{sum}(\text{foo xs}) = 2 * \text{sum xs} - \text{length xs}, \forall \text{xs} :: [\text{Int}]$

► **I.S.:** Sei  $\text{xs} :: [\text{Int}]$  beliebig aber fest, sodass gilt:

$$\text{sum}(\text{foo xs}) = 2 * \text{sum xs} - \text{length xs} \quad (\text{I.V.})$$



# Induktion über Listen

ZZ:  $\text{sum}(\text{foo } xs) = 2 * \text{sum } xs - \text{length } xs, \forall xs :: [Int]$

► **I.S.:** Sei  $xs :: [Int]$  beliebig aber fest, sodass gilt:

$$\text{sum}(\text{foo } xs) = 2 * \text{sum } xs - \text{length } xs$$

dann gilt  $\forall x :: Int$

$$\text{sum}(\text{foo } (x : xs))$$

# Induktion über Listen

ZZ:  $\text{sum}(\text{foo } xs) = 2 * \text{sum } xs - \text{length } xs, \forall xs :: [Int]$

► **I.S.:** Sei  $xs :: [Int]$  beliebig aber fest, sodass gilt:

$$\text{sum}(\text{foo } xs) = 2 * \text{sum } xs - \text{length } xs$$

dann gilt  $\forall x :: Int$

$$\text{sum}(\text{foo } (x : xs))$$

$$\stackrel{Z3}{=} \text{sum}(x : x : (-1) : \text{foo}(xs))$$

# Induktion über Listen

ZZ:  $\text{sum}(\text{foo } xs) = 2 * \text{sum } xs - \text{length } xs, \forall xs :: [Int]$

► **I.S.:** Sei  $xs :: [Int]$  beliebig aber fest, sodass gilt:

$$\text{sum}(\text{foo } xs) = 2 * \text{sum } xs - \text{length } xs$$

dann gilt  $\forall x :: Int$

$$\text{sum}(\text{foo } (x : xs))$$

$$\stackrel{Z3}{=} \text{sum}(x : x : (-1) : \text{foo}(xs))$$

$$\stackrel{3*Z7}{=} x + x + (-1) + \text{sum}(\text{foo}(xs))$$

# Induktion über Listen

ZZ:  $\text{sum}(\text{foo } xs) = 2 * \text{sum } xs - \text{length } xs, \forall xs :: [Int]$

► **I.S.:** Sei  $xs :: [Int]$  beliebig aber fest, sodass gilt:

$$\text{sum}(\text{foo } xs) = 2 * \text{sum } xs - \text{length } xs$$

dann gilt  $\forall x :: Int$

$$\text{sum}(\text{foo } (x : xs))$$

$$\stackrel{Z3}{=} \text{sum}(x : x : (-1) : \text{foo}(xs))$$

$$\stackrel{3*Z7}{=} x + x + (-1) + \text{sum}(\text{foo}(xs))$$

$$\stackrel{I.V.}{=} x + x + (-1) + 2 * \text{sum } xs - \text{length } xs$$

# Induktion über Listen

ZZ:  $\text{sum}(\text{foo } xs) = 2 * \text{sum } xs - \text{length } xs, \forall xs :: [Int]$

► **I.S.:** Sei  $xs :: [Int]$  beliebig aber fest, sodass gilt:

$$\text{sum}(\text{foo } xs) = 2 * \text{sum } xs - \text{length } xs$$

dann gilt  $\forall x :: Int$

$$\text{sum}(\text{foo } (x : xs))$$

$$\stackrel{Z3}{=} \text{sum}(x : x : (-1) : \text{foo}(xs))$$

$$\stackrel{3*Z7}{=} x + x + (-1) + \text{sum}(\text{foo}(xs))$$

$$\stackrel{I.V.}{=} x + x + (-1) + 2 * \text{sum } xs - \text{length } xs$$

$$= 2 * x + 2 * \text{sum } xs - 1 - \text{length } xs$$

# Induktion über Listen

ZZ:  $\text{sum}(\text{foo } xs) = 2 * \text{sum } xs - \text{length } xs, \forall xs :: [Int]$

► **I.S.:** Sei  $xs :: [Int]$  beliebig aber fest, sodass gilt:

$$\text{sum}(\text{foo } xs) = 2 * \text{sum } xs - \text{length } xs$$

dann gilt  $\forall x :: Int$

$$\text{sum}(\text{foo } (x : xs))$$

$$\stackrel{Z3}{=} \text{sum}(x : x : (-1) : \text{foo}(xs))$$

$$\stackrel{3*Z7}{=} x + x + (-1) + \text{sum}(\text{foo}(xs))$$

$$\stackrel{I.V.}{=} x + x + (-1) + 2 * \text{sum } xs - \text{length } xs$$

$$= 2 * x + 2 * \text{sum } xs - 1 - \text{length } xs$$

$$\stackrel{Z7,11}{=} 2 * \text{sum}(x : xs) - \text{length}(x : xs)$$

# Lambda-Kalkül

Was ist ein Kalkül...?

# Lambda-Kalkül

Was ist ein Kalkül...?

- ▶ Formales System von **Regeln** zur Ableitung von Aussagen aus **Axiomen**



# Lambda-Kalkül

## Was ist der $\lambda$ -Kalkül...?

- ▶ Axiome  $\Rightarrow$   $\lambda$ -Terme:

$$f(x) = x + x$$

$$f\ x = x + x$$

$$\lambda x. (+x)x$$

- ▶ 3 Typen von Termen:

Variable  $x$

Applikation  $(term1)(term2)$

Abstraktion  $\lambda x. term$

# Lambda-Kalkül

## Was ist der $\lambda$ -Kalkül...?

- ▶ Regeln:

$\beta$ -Reduktion: Applikation  $(\lambda x. (+x)x) y \Rightarrow (+y)y$

$\alpha$ -Konversion: Umbenennen  $(\lambda x. (+x)x) x \Rightarrow (\lambda x_1. (+x_1)x_1) x$

- ▶ Normalform ist erreicht, wenn keine Regel mehr (sinnvoll) anwendbar ist

# Lambda-Kalkül

Applikation - Vergleich mit Funktionsapplikation

$$f(x) = x + x \Rightarrow f(3)$$

$$\lambda x. (+x)x \Rightarrow (\lambda x. (+x)x) 3$$

$$\Rightarrow (\lambda x. (\lambda y. (+x)y)) 3 4$$

$$\Rightarrow (\lambda y. (+3)y) 4$$

$$\Rightarrow (+3) 4$$

# Lambda-Terme

## Schreibkonventionen

1. Applikation ist linksassoziativ:

$$t_1 t_2 t_3 = (t_1 t_2) t_3$$

2. Abstraktionen können zusammengefasst werden:

$$\lambda x. (\lambda y. term) = \lambda xy. term$$

3. Applikation vor Abstraktion:

$$\lambda x. xy = \lambda x. (xy) \text{ und nicht } (\lambda x. x)y$$

# Lambda-Terme

Freie und gebundene Vorkommen von Variablen

$((\lambda x. (\lambda y. (+ x) y)) y)$

$\Rightarrow GV = \{x, y\}$

$\Rightarrow FV = \{y\}$

$\Rightarrow$  Applikation term1 term2

$\beta$ -Reduktion möglich gdw.  $GV(term1) \cap FV(term2) = \emptyset$

# Normalisierung-Beispiel

$(\lambda x.(\lambda y.xz(yz)) (\lambda x.y(\lambda y.y)))$

$GV_{\text{term1}}\{x, y\} \cap FV_{\text{term2}} = \{y\} \neq \emptyset$

$\rightarrow_{\alpha} (\lambda x.(\lambda o.xz(oz)) (\lambda x.y(\lambda y.y)))$  ...Umbenennung y zu o

$\rightarrow_{\beta} \lambda o.(\lambda x.y(\lambda y.y)) z (oz)$  ...Applikation vor Abstraktion

$\rightarrow_{\beta} \lambda o.(y(\lambda y.y)) (oz)$  ...nichts geht mehr

$= \lambda o.(y(\lambda y.y)) (oz)$