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**: Nimm an, Behauptung gilt für ein "beliebiges aber festes" Element "n"
- **Induktionsschritt**: Zeige, dass dann Behauptung für dessen Nachfolger "n+1" gilt

  $\Rightarrow$ Behauptung muss für alle gelten, die "größer" sind als "0"

Induktion über Listen

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

- **I.A.**:

  $\texttt{sum (foo [ ])}$

Induktion über Listen

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

- **I.A.**:

  $\texttt{sum (foo [ ])}$
  $\overset{Z2}{=\!=} sum\ [\ ]$

Induktion über Listen

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

- **I.A.**:

$$\begin{aligned} &\texttt{sum (foo [ ])} \\ &\overset{Z2}{=\!=} sum\ [\ ] \\ &\overset{Z6}{=\!=} 0 \end{aligned}$$

Induktion über Listen

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

- **I.A.**:

$$\begin{aligned} &\texttt{sum (foo [ ])} \\ &\overset{Z2}{=\!=} sum\ [\,] \\ &\overset{Z6}{=\!=} 0 \\ &=\!= 2 * 0 - 0 \end{aligned}$$

Induktion über Listen

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

- **I.A.**:

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

Induktion über Listen

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

- **I.V.**: Sei $\texttt{xs}$ :: $[\texttt{Int}]$ beliebig aber fest und gelte:
  $\texttt{sum (foo xs)} \ = \ 2 * \texttt{sum xs} - \texttt{length xs}$

Induktion über Listen

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

- **I.V.**: Sei $\texttt{xs}$ :: $[\texttt{Int}]$ beliebig aber fest und gelte:

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

- **I.S.**: Dann gilt $\forall x :: Int$

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

Induktion über Listen

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

- **I.V.**: Sei $\texttt{xs} :: \texttt{[Int]}$ beliebig aber fest und gelte:

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

- **I.S.**: Dann gilt $\forall x :: Int$

  $$\texttt{sum (foo (x : xs))}$$
  $$\overset{Z3}{=\!=} sum\ (\ x\ :x\ :(-1):\ foo(xs))$$

Induktion über Listen

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

- **I.V.**: Sei $\texttt{xs} :: [\texttt{Int}]$ beliebig aber fest und gelte:

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

- **I.S.**: Dann gilt $\forall x :: Int$

  $$\begin{aligned} &\texttt{sum (foo (x : xs))} \\ &\overset{Z3}{=\!=} sum\ (\ x\ \ : x\ \ : (-1) :\ \ foo(xs)) \\ &\overset{3*Z7}{=\!=} x + x + (-1) + sum\ (foo(xs)) \end{aligned}$$

Induktion über Listen

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

- **I.V.**: Sei $\texttt{xs} :: [\texttt{Int}]$ beliebig aber fest und gelte:

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

- **I.S.**: Dann gilt $\forall x :: Int$

  $$\begin{aligned}&\texttt{sum (foo (x : xs))}\\ &\overset{Z3}{=\!=}\ sum\ (\ x\ :x\ :(-1):\ foo(xs))\\ &\overset{3*Z7}{=\!=}\ x + x + (-1) + sum\ (foo(xs))\\ &\overset{I.V.}{=\!=}\ x + x + (-1) + 2 * sum\ xs - length\ xs\end{aligned}$$

Induktion über Listen

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

- **I.V.**: Sei $\texttt{xs} :: \texttt{[Int]}$ beliebig aber fest und gelte:

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

- **I.S.**: Dann gilt $\forall x :: Int$

  $$\begin{aligned}
  &\texttt{sum (foo (x : xs))} \\
  &\overset{Z3}{=\!=} sum\ (\ x\ \ : x\ \ : (-1) :\ \ foo(xs)) \\
  &\overset{3*Z7}{=\!=} x + x + (-1) + sum\ (foo(xs)) \\
  &\overset{I.V.}{=\!=} x + x + (-1) + 2 * sum\ xs - length\ xs \\
  &=\!= 2 * x + 2 * sum\ xs - 1 - length\ xs
  \end{aligned}$$

Induktion über Listen

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

- **I.V.**: Sei $\texttt{xs} :: [\texttt{Int}]$ beliebig aber fest und gelte:

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

- **I.S.**: Dann gilt $\forall x :: Int$

  $$\begin{aligned}
  &\texttt{sum (foo (x : xs))} \\
  &\overset{Z3}{=\!=} sum\ (\ x\ :x\ :(-1):\ foo(xs)) \\
  &\overset{3*Z7}{=\!=} x + x + (-1) + sum\ (foo(xs)) \\
  &\overset{I.V.}{=\!=} x + x + (-1) + 2 * sum\ xs - length\ xs \\
  &=\!= 2 * x + 2 * sum\ xs - 1 - length\ xs \\
  &\overset{Z7,11}{=\!=} 2 * sum\ (x : xs) - length\ (x : xs)
  \end{aligned}$$

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 λ -Kalkül...?

- Axiome $\Rightarrow$ λ-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 λ -Kalkül...?

- Regeln:

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

  α-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 - Vegleich mit Funktionsapplikation

$\mathrm{f}\ (\mathrm{x}) = \mathrm{x} + \mathrm{x} \Rightarrow \mathrm{f}(3)$

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

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

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

$\Rightarrow (+3)\ 4$

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:
   $\lambda x.xy = \lambda x.(xy)$ 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

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

Normalisierung-Beispiel

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

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

$\rightarrow_\alpha (\lambda x.(\lambda o.xz(oz))\ (\lambda x.y(\lambda y.y))$ ...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

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