# PROGRAMMIERUNG

## ÜBUNG 6:
## STRUKTURELLE INDUKTION & λ-KALKÜL (TEIL 1)

Eric Kunze

`eric.kunze@tu-dresden.de`

TU Dresden, 16. Mai 2022

letzte Änderung:
18.05.2022, 16:35

# INHALT

# Induktionsbeweise

*Aufgabe 2*

# VOLLSTÄNDIGE INDUKTION AUF $\mathbb{N}$

**Definition:** natürliche Zahlen $\mathbb{N} := \{0, 1, \ldots\}$

$$\begin{array}{rll} \text{Basisfall:} & 0 \in \mathbb{N} & \\ \text{Rekursionsfall:} & x + 1 \in \mathbb{N} & \text{für} \quad x \in \mathbb{N} \end{array}$$

**Beweis von Eigenschaften:** Eigenschaft $=$ Prädikat $P$

$$\boxed{\text{zu zeigen:} \qquad \text{für alle } x \in \mathbb{N} \text{ gilt } P(x)}$$

**vollständige Induktion:**

- Induktionsanfang:
  zeige $P(x)$ für $x = 0$
- Induktionsvoraussetzung:
  Sei $x \in \mathbb{N}$, sodass $P(x)$ gilt. $P(x)$ gilt noch nicht für *alle* $x \in \mathbb{N}$
- Induktionsschritt:
  zeige $P(x + 1)$ unter Nutzung der Induktionsvoraussetzung

# INDUKTION AUF LISTEN

**Erinnerung:** Rekursion über Listen `xs`

Basisfall: `xs` = `[]`

Rekursionsfall: `xs` = `(y:ys)` für `ys :: [a]`

**Beweis von Programmeigenschaften:** Eigenschaft = Prädikat $P$

| zu zeigen: | für alle `xs :: [a]` gilt $P($`xs`$)$ |
|---|---|

**Induktion auf Listen:**

- Induktionsanfang:
  zeige $P($`xs`$)$ für `xs == []`
- Induktionsvoraussetzung:
  Sei `xs :: [a]` eine Liste für die $P($`xs`$)$ gilt.
- Induktionsschritt:
  zeige $P($`x:xs`$)$ für alle `x :: a` unter Nutzung der Induktionsvoraussetzung

*Allgemeiner Hinweis:* Es müssen immer alle Variablen quantifiziert werden!

# STRUKTURELLE INDUKTION

**Erinnerung:** Rekursion über Bäume

Basisfall: `Nil` oder `Leaf x` für `x :: a`

Rekursionsfall: `Branch x l r` für `x :: a` und `l,r :: BinTree a`

| | |
|---|---|
| zu zeigen: | für alle `t :: BinTree a` gilt $P$(`t`) |

**strukturelle Induktion:**

- Induktionsanfang:
  zeige $P$(`t`) für `t == Nil` oder `t == Leaf x` für alle `x :: a`
- Induktionsvoraussetzung:
  Seien `l, r :: BinTree a` zwei Bäume, sodass $P$(`l`) und $P$(`r`) gilt.
- Induktionsschritt:
  zeige $P$(`Branch x l r`) für alle `x :: a` unter Nutzung der Induktionsvoraussetzung

| |
|---|
| *Allgemeiner Hinweis:* Es müssen immer alle Variablen quantifiziert werden! |

# AUFGABE 1

```
data Tree a = Node a (Tree a) (Tree a) | Leaf a

mirror :: Tree a -> Tree a
mirror (Node x t1 t2) = Node x (mirror t2) (mirror t1)
mirror (Leaf x) = Leaf x

yield :: Tree a -> [a]
yield (Node _ t1 t2) = yield t1 ++ yield t2
yield (Leaf x) = [x]
```

verwendbare Eigenschaften:

$$\texttt{reverse [x] = [x]} \tag{E1}$$

$$\texttt{reverse (xs ++ ys) = reverse ys ++ reverse xs} \tag{E2}$$

Mittels struktureller Induktion ist folgende Aussage zu zeigen:

*Für jeden Typ $a$* und jeden Baum `t :: Tree a` gilt

$$\texttt{reverse (yield t) = yield (mirror t)}$$

# AUFGABE 1

**Induktionsanfang:** Sei `a` ein Typ und `x :: a` beliebig sowie `t = Leaf x`.

linke Seite: $\texttt{reverse (yield (Leaf x))} \overset{(9)}{=} \texttt{reverse [x]} \overset{(E1)}{=} \texttt{[x]}$

rechte Seite: $\texttt{yield (mirror (Leaf x))} \overset{(5)}{=} \texttt{yield (Leaf x)} \overset{(9)}{=} \texttt{[x]}$

**Induktionsvoraussetzung:** Seien `a` ein Typ und `l, r:: Tree a`, sodass

```
reverse (yield r) = yield (mirror l)        (IV1)
reverse (yield r) = yield (mirror r)        (IV2)
```

gilt.

# AUFGABE 1

**Induktionsschritt:** Sei `a` ein Typ und `x :: a` beliebig. Es gilt

$$
\begin{aligned}
&\texttt{reverse (yield (Node x l r))} \\
&\overset{(8)}{=} \texttt{reverse (yield l ++ yield r)} \\
&\overset{(E2)}{=} \texttt{reverse (yield r) ++ reverse (yield l)} \\
&\overset{(IV1)}{=} \texttt{reverse (yield r) ++ yield (mirror l)} \\
&\overset{(IV2)}{=} \texttt{yield (mirror r) ++ yield (mirror l)} \\
&\overset{(8)}{=} \texttt{yield (Node x (mirror r) (mirror l))} \\
&\overset{(4)}{=} \texttt{yield (mirror (Node x l r))}
\end{aligned}
$$

# Der λ-Kalkül

- weitere funktionale Programmiersprache
- Programme = $\lambda$-Terme
- Vorstellung: *anonyme* Funktionen

Sei $X$ eine Menge mit Variablen, $\Sigma$ eine Menge mit Symbolen. Die gültigen $\lambda$-Terme sind *induktiv* definiert:

1. **Atome** (Variablen oder Symbole) sind gültige $\lambda$-Terme.
2. **Abstraktion**: Ist $t$ ein gültiger $\lambda$-Term und $x \in X$ eine Variable, dann ist auch $(\lambda x.t)$ ein gültiger $\lambda$-Term.
3. **Applikation**: Sind $t_1$ und $t_2$ gültige $\lambda$-Terme, dann ist auch $(t_1\ t_2)$ ein gültiger $\lambda$-Term.

# BEISPIELE (INFORMELL)

Vorstellung: Jeder $\lambda$-Term beschreibt eine anonyme Funktion.

- Abstraktion gibt das Argument an:

$$\lambda x.t \quad \leftrightarrow \quad f(x) = t$$

- Applikation beschreibt Funktionsanwendung (Einsetzen):

$$((\lambda x.t)\ 2) \quad \leftrightarrow \quad f(2)$$

**Beispiel:**

$$\text{quadriere } = \lambda x.\ x \cdot x$$
$$((\lambda x.x \cdot x)\ 2) = 2 \cdot 2 = 4$$

$\nearrow$ $\beta$-Reduktion

# VERABREDUNGEN

- Applikation ist linksassoziativ:

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

- mehrfache Abstraktion:

$$(\lambda x_1.(\lambda x_2.(\lambda x_3.t))) = \lambda x_1 x_2 x_3.t$$

- Applikation vor Abstraktion:

$$\begin{aligned}(\lambda x.\ x\ y) &= (\lambda x.(x\ y)) \\ &\neq ((\lambda x.x)\ y)\end{aligned}$$

# GEBUNDENE UND FREIE VORKOMMEN

Mengen $\mathrm{FV}(t)$ und $\mathrm{GV}(t)$ geben frei bzw. gebunden *vorkommende* Variablen von $t$ an — induktive Definition

- einzelne **Variablen** sind immer frei:
  $x \in X \quad \Rightarrow \quad \mathrm{FV}(x) = \{x\}\,, \mathrm{GV}(x) = \emptyset$
- **Symbole** sind weder frei noch gebunden
- **Applikation**: Sei $t = (t_1\ t_2)$. Dann
  $\Rightarrow \quad \mathrm{FV}(t) = \mathrm{FV}(t_1) \cup \mathrm{FV}(t_2), \quad \mathrm{GV}(t) = \mathrm{GV}(t_1) \cup \mathrm{GV}(t_2)$
- **Abstraktion**: $t = \lambda x.t'$
  $\Rightarrow \quad \mathrm{FV}(t) = \mathrm{FV}(t') \setminus \{x\}\,, \quad \mathrm{GV}(t) = \mathrm{GV}(t') \cup \{x\}$

## β–Reduktion

Seien $s, t \in \lambda(\Sigma)$ gültige $\lambda$-Terme und es gilt $\mathrm{GV}(t) \cap \mathrm{FV}(s) = \emptyset$.

$$(\lambda x.t)\ s \quad \longrightarrow_\beta \quad t[x/s]$$

- Bedeutung von $t[x/s]$: Ersetze jedes *freie* Vorkommen von $x$ in $t$ durch $s$.
- Erinnerung: Vorstellung der Applikation als „Einsetzen“ in Funktionen
- beachte: Abstraktion $\lambda x$ entfällt

**Bsp.:** Seien die Symbole gegeben durch $\Sigma = \{3, a\}$.

$$(\lambda x.\underbrace{+x3}_{\mathrm{GV}=\emptyset})\ (\underbrace{\lambda z.a}_{\mathrm{FV}=\emptyset}) \quad \longrightarrow_\beta \quad +\ (\lambda z.a)3$$

# $\alpha$ – KONVERSION

- Was machen wir, wenn Voraussetzung $\mathrm{FV}(t) \cap \mathrm{GV}(t) = \emptyset$ für β-Reduktion nicht erfüllt ist?
- einfacher Ausweg: entsprechende Variablen umbenennen, sodass Bedingung erfüllt ist

## α–Konversion

Sei $t \in \lambda(\Sigma)$ und $z \notin \mathrm{GV}(t) \cup \mathrm{FV}(t)$.

$$(\lambda x.t) \quad \longrightarrow_\alpha \quad \lambda z.t[x/z]$$

**Bsp.:** Seien die Symbole gegeben durch $\Sigma = \{3, a\}$.

$$(\lambda x.(\underbrace{\lambda y. + xy}_{\mathrm{GV}=\{y\}}))\;(\underbrace{\;y\;}_{\mathrm{FV}=\{y\}}) \quad \longrightarrow_\alpha \quad (\lambda x.(\underbrace{\lambda z. + xz}_{\mathrm{GV}=\{z\}}))\;(\underbrace{\;y\;}_{\mathrm{FV}=\{y\}})$$

# NORMALFORM

NEW

Nutzt man sowohl $\alpha$-Konversionen $\Rightarrow_\alpha$ als auch $\beta$-Reduktionen $\Rightarrow_\beta$, so spricht man von der *Rechenvorschrift* $\Rightarrow$ des $\lambda$-Kalküls.

Führt man mehrere Schritte direkt aus, so schreibt man $\Rightarrow^*$ statt $\Rightarrow$.

Wenn $t \Rightarrow^* s$ und es gibt keine $\lambda$-Terme $s_1, s_2$ mit $s \Rightarrow_\alpha^* s_1 \Rightarrow_\beta^* s_2$, dann heißt $s$ ($\beta$-)**Normalform** von $t$.

**Anschauung:**

- möglichst einfache Form eines Lambda-Terms
- „fertig ausgerechnete" Funktion

Die Rechenvorschrift $\Rightarrow$ ist *konfluent*, d. h. für alle $\lambda$-Terme $t, t_1, t_2$ gilt: wenn $t \Rightarrow^* t_1$ und $t \Rightarrow^* t_2$, dann gibt es einen $\lambda$-Term $s$ mit $t_1 \Rightarrow^* s$ und $t_2 \Rightarrow^* s$.

Damit gilt auch: wenn eine Normalform existiert, dann ist sie *eindeutig* (egal welche Schritte man zwischendurch auswählt).

## NORMALFORM — VORGEHEN

NEW

**Vorgehen:**

1. Bestimmung von Stellen, an denen reduziert werden kann
   Anforderungen: (Teil-)Term der Form $(\underbrace{(\lambda x.t)}_{=t'}\, s)$
   - ▷ Applikation $(t' s)$: es muss ein Term $s$ zum Einsetzen vorhanden sein
   - ▷ Abstraktion in $t' = (\lambda x.t)$: der erste Term braucht eine „Variable" $x$, für die etwas eingesetzt werden kann
2. Bestimmung gebundener und freier Vorkommen: $((\lambda x.\underbrace{t}_{\text{GV}})\underbrace{s}_{\text{FV}})$
3. Falls $\text{GV}(t) \cap \text{FV}(s) \neq \emptyset$: $\alpha$-Konversion
   - ▷ Umbenennung der gebunden Vorkommen in $t$
4. Falls $\text{GV}(t) \cap \text{FV}(s) = \emptyset$: $\beta$-Reduktion
   - ▷ Streichen der Abstraktion $\lambda x.$
   - ▷ Setze für jedes Vorkommen von $x$ in $t$ den Term $s$ sein

- $t_1 = (\lambda x.xy)\,(\lambda y.y)$:
  - $\mathrm{GV}(t_1) = \{x, y\}$
  - $\mathrm{FV}(t_1) = \{y\}$
- $t_2 = (\lambda x.(\lambda y.z(\lambda z.z(\lambda x.y))))$
  - $\mathrm{GV}(t_2) = \{x, y, z\}$
  - $\mathrm{FV}(t_2) = \{z\}$
- $t_3 = (\lambda x.(\lambda y.z(yz)))(\lambda x.y(\lambda y.y))$
  - $\mathrm{GV}(t_3) = \{x, y\}$
  - $\mathrm{FV}(t_3) = \{y, z\}$

Term 1

$$
\begin{aligned}
& (\lambda x.(\underbrace{\lambda y.x\ z\ (y\ z)}_{\mathrm{GV}=\{y\}}))\ \underbrace{(\lambda x.y\ (\lambda y.y))}_{\mathrm{FV}=\{y\}} \\
\Rightarrow_\alpha \quad & (\lambda x.(\underbrace{\lambda y_1.x\ z\ (y_1\ z))}_{\mathrm{GV}=\{y\}})\ \underbrace{(\lambda x.y\ (\lambda y.y))}_{\mathrm{FV}=\{y\}} \\
\Rightarrow_\beta \quad & (\lambda y_1.(\lambda x.\underbrace{y\ (\lambda y.y)}_{\mathrm{GV}=\{y\}})\ \ \underbrace{z}_{\mathrm{FV}=\{z\}}\ \ (y_1\ z)) \\
\Rightarrow_\beta \quad & (\lambda y_1.(y\ (\lambda y.y))\ (y_1\ z)) \\
= \ & (\lambda y_1.y\ (\lambda y.y)\ (y_1\ z))
\end{aligned}
$$

# AUFGABE 2 — TEIL (B)

Term 2

$$
\begin{aligned}
& (\lambda x.(\underbrace{\lambda y.(\lambda z.z)}_{\mathsf{GV}=\{y,z\}}))\ \underbrace{x}_{\mathsf{FV}=\{x\}}\ (+\ y\ 1) \\
\Rightarrow_\beta \quad & (\lambda y.\ \underbrace{(\lambda z.z)}_{\mathsf{GV}=\{z\}})\ \underbrace{(+\ y\ 1)}_{\mathsf{FV}=\{y\}} \\
\Rightarrow_\beta \quad & (\lambda z.z)
\end{aligned}
$$

Term 3

$$
\begin{aligned}
& (\lambda x.(\lambda y.x\ (\lambda z.y\ z)))\ (((\lambda x.\underbrace{(\lambda y.y)}_{\mathrm{GV}=\{y\}})\ \underbrace{8}_{\mathrm{FV}=\emptyset})\ (\lambda x.(\lambda y.y)\ x)) \\
\Rightarrow_\beta\quad & (\lambda x.(\lambda y.x\ (\lambda z.y\ z)))\ ((\lambda y.y))\ (\lambda x.(\lambda y.y)\ x)) \\
\Rightarrow_\beta\quad & (\lambda x.(\lambda y.x\ (\lambda z.y\ z)))\ ((\lambda y.y)\ (\lambda x.(\lambda y.\underbrace{y}_{\mathrm{GV}=\emptyset})\ \underbrace{x}_{\{x\}})) \\
\Rightarrow_\beta\quad & (\lambda x.(\lambda y.x\ (\lambda z.y\ z)))\ ((\lambda y.\underbrace{y}_{\mathrm{GV}=\emptyset})\ \underbrace{(\lambda x.x)}_{\mathrm{FV}=\emptyset}) \\
\Rightarrow_\beta\quad & (\lambda x.\underbrace{(\lambda y.x\ (\lambda z.y\ z)))}_{\mathrm{GV}=\{y,z\}}\ \underbrace{(\lambda x.x)}_{\mathrm{FV}=\emptyset} \\
\Rightarrow_\beta\quad & (\lambda y.(\lambda x.\underbrace{x}_{\mathrm{GV}=\emptyset})\ \underbrace{(\lambda z.y\ z)}_{\mathrm{FV}=\{y\}}) \\
\Rightarrow_\beta\quad & (\lambda y.(\lambda z.y\ z)) \quad = \quad (\lambda yz.y\ z)
\end{aligned}
$$

Term 4

$$(\lambda h.(\lambda x.h\,(x\,x))\,(\lambda x.h\,(x\,x)))((\lambda x.\underbrace{x}_{\text{GV}=\emptyset})\ \underbrace{(+\,1\,5))}_{\text{FV}=\emptyset}$$

$$\Rightarrow_\beta \quad (\lambda h.(\lambda x.\underbrace{h\,(x\,x))}_{\text{GV}=\emptyset}\ \underbrace{(\lambda x.h\,(x\,x)))}_{\text{FV}=\{h\}}(+\,1\,5)$$

$$\Rightarrow_\beta \quad (\lambda h.h\,((\lambda x.\underbrace{h\,(x\,x))}_{\text{GV}=\emptyset}\ \underbrace{(\lambda x.h\,(x\,x)))}_{\text{FV}=\{h\}}))\,(+\,1\,5)$$

$$\Rightarrow_\beta \quad (\lambda h.h\,(h\,((\lambda x.\underbrace{h\,(x\,x))}_{\text{GV}=\emptyset}\ \underbrace{(\lambda x.h\,(x\,x)))}_{\text{FV}=\{h\}})))\,(+\,1\,5)$$

$\longrightarrow$ endlose Rekursion, bei der $h$ durch $(+\,1\,5)$ noch reduziert werden könnte

$$\Rightarrow_\beta \quad (+\,1\,5)\,((+\,1\,5)\,((\lambda x.(+\,1\,5)\,(x\,x))\,(\lambda x.(+\,1\,5)\,(x\,x))))$$

Term 5

$$
\begin{aligned}
& (\lambda f.\underbrace{(\lambda a.(\lambda b.f\ a\ b))}_{\mathsf{GV}=\{a,b\}})\ \underbrace{(\lambda x.(\lambda y.x))}_{\mathsf{FV}=\emptyset}) \\
\Rightarrow_\beta \quad & (\lambda a.(\lambda b.(\lambda x.\underbrace{(\lambda y.x)}_{\mathsf{GV}=\{y\}})\ \underbrace{a}_{\mathsf{FV}=\{a\}}\ b)) \\
\Rightarrow_\beta \quad & (\lambda a.(\lambda b.(\lambda y.\underbrace{a}_{\mathsf{GV}=\emptyset})\ \underbrace{b}_{\mathsf{FV}=\{b\}}\ )) \\
\Rightarrow_\beta \quad & (\lambda a.(\lambda b.a)) \\
= {} & (\lambda ab.a)
\end{aligned}
$$

# AUFGABE 3

*(a)* $A$ mit $A\ t\ s\ u \Rightarrow^* s$: $A = (\lambda xyz\ .\ y)$

*(b)* $B$ mit $B\ t\ s \Rightarrow^* s\ t$: $B = (\lambda xy\ .\ yx)$

*(c)* $C$ mit $C\ C \Rightarrow^* C\ C$: $C = (\lambda x\ .\ xx)$

denn: $(\lambda x.\ \underbrace{xx}_{GV=\emptyset})\ \underbrace{(\lambda x.xx)}_{FV=\emptyset} \quad \Rightarrow^\beta \quad (\lambda x.xx)(\lambda x.xx)$

*(d)* $D$ mit $D \Rightarrow^* D$: $D = (C\ C)$

*(e)* $E$ mit $E\ E\ t \Rightarrow^* E\ t\ E$: $E = (\lambda xy\ .\ xyx)$

denn:

$$\begin{aligned}
(\lambda x\ \underbrace{y\ .\ xyx}_{GV=\{y\}})\underbrace{(\lambda xy\ .\ xyx)}_{FV=\emptyset}\ t \quad &\Rightarrow^\beta \quad (\lambda y\ .\ (\lambda xy\ .\ xyx)\ y\ (\lambda xy\ .\ xyx))\ t \\
&\Rightarrow^\beta \quad \underbrace{(\lambda xy\ .\ xyx)}_{=E}\ t\ \underbrace{(\lambda xy\ .\ xyx)}_{=E}
\end{aligned}$$