TECHNISCHE UNIVERSITÄT DRESDEN

PROGRAMMIERUNG

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

Eric Kunze

eric.kunze@tu-dresden.de

TU Dresden, 16. Mai 2022

letzte Änderung: 15.05.2022, 14:02

INHALT

Induktionsbeweise

Aufgabe 2

VOLLSTÄNDIGE INDUKTION AUF $\mathbb{N}$

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

Basisfall:	$0 \in \mathbb{N}$		
Rekursionsfall:	$x + 1 \in \mathbb{N}$	für	$x \in \mathbb{N}$

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

```haskell
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`.

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

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

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

$$\texttt{reverse (yield r) = yield (mirror l)} \qquad \text{(IV1)}$$

$$\texttt{reverse (yield r) = yield (mirror r)} \qquad \text{(IV2)}$$

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

λ-KALKÜL

- weitere funktionale Programmiersprache
- Programme = λ-Terme
- Vorstellung: *anonyme* Funktionen

Sei X eine Menge mit Variablen, Σ eine Menge mit Symbolen. Die gültigen λ-Terme sind *induktiv* definiert:

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

BEISPIELE (INFORMELL)

Vorstellung: Jeder λ-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:

$$\begin{aligned} \text{quadriere } &= \lambda x.\ x \cdot x \\ ((\lambda x.x \cdot x)\ 2) &= 2 \cdot 2 = 4 \end{aligned}$$

$\nearrow$ β-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 $\mathsf{FV}(t)$ und $\mathsf{GV}(t)$ geben frei bzw. gebunden *vorkommende* Variablen von t an — induktive Definition

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

β–Reduktion

Seien $s, t \in \lambda(\Sigma)$ gültige λ-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 λx entfällt

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

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

α – 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 $\Sigma = \{3, a\}$.

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

AUFGABE 2 — TEIL (A)

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

AUFGABE 2 — TEIL (B)

Term 1

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

Term 2

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

Term 3

$$(\lambda x.(\lambda y.x\ (\lambda z.y\ z)))\ (((\lambda x.\underbrace{(\lambda y.y)}_{\text{GV}=\{y\}})\ \underbrace{8}_{\text{FV}=\emptyset})\ (\lambda x.(\lambda y.y)\ x))$$

$$\Rightarrow_\beta \quad (\lambda x.(\lambda y.x\ (\lambda z.y\ z)))\ ((\lambda y.y))\ (\lambda x.(\lambda y.y)\ x))$$

$$\Rightarrow_\beta \quad (\lambda x.(\lambda y.x\ (\lambda z.y\ z)))\ ((\lambda y.y)\ (\lambda x.(\lambda y.\underbrace{y}_{\text{GV}=\emptyset})\ \underbrace{x}_{\{x\}}))$$

$$\Rightarrow_\beta \quad (\lambda x.(\lambda y.x\ (\lambda z.y\ z)))\ ((\lambda y.\underbrace{y}_{\text{GV}=\emptyset})\ \underbrace{(\lambda x.x)}_{\text{FV}=\emptyset})$$

$$\Rightarrow_\beta \quad (\lambda x.\underbrace{(\lambda y.x\ (\lambda z.y\ z)))}_{\text{GV}=\{y,z\}}\ \underbrace{(\lambda x.x)}_{\text{FV}=\emptyset}$$

$$\Rightarrow_\beta \quad (\lambda y.(\lambda x.\underbrace{x}_{\text{GV}=\emptyset})\ \underbrace{(\lambda z.y\ z)}_{\text{FV}=\{y\}})$$

$$\Rightarrow_\beta \quad (\lambda y.(\lambda z.y\ z)) \quad = \quad (\lambda yz.y\ z)$$

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

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

AUFGABE 2 — TEIL (B)

Term 5

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

AUFGABE 3

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

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

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

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 $D \Rightarrow^* D$: $\quad D = (C\ C)$

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

denn:

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