PROGRAMMIERUNG

ÜBUNG 5: INDUKTION

Eric Kunze

eric.kunze@[redacted]tu-dresden.de

INHALT

Induktionsbeweise

Aufgaben 1 und 2

VOLLSTÄNDIGE INDUKTION AUF $\mathbb{N}$

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

$0 \mapsto 1 \mapsto 2 \mapsto \ldots$

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. $\qquad$ $P(x)$ 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`

$2 : 3 : [\,] = [2,3]$

Basisfall: $\mathtt{xs} = \underline{\underline{\mathtt{[]}}}$ 0

Rekursionsfall: $\mathtt{xs} = (\underline{\mathtt{y}}:\underline{\mathtt{ys}})$ für $\mathtt{ys}$ $\boxed{\mathtt{::\ [a]}}$

$x+1$ $x \in \mathbb{N}$

INDUKTION AUF LISTEN

Erinnerung: Rekursion über Listen `xs`

Basisfall: `xs = []`

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

Beweis von Programmeigenschaften: Eigenschaft = Prädikat P

$$\boxed{\text{zu zeigen:} \qquad \text{für alle } \texttt{xs :: [a]} \text{ gilt } P(\texttt{xs})}$$

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 == []`

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.

INDUKTION AUF LISTEN

Erinnerung: Rekursion über Listen `xs`

Basisfall: `xs` = `[]`

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

N

Beweis von Programmeigenschaften: Eigenschaft = Prädikat P

$$\boxed{\text{zu zeigen: \quad für alle } \texttt{xs :: [a]} \text{ gilt } P(\texttt{xs})}$$

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

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`

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`

$\boxed{\text{zu zeigen:} \quad \text{für alle } \texttt{t :: BinTree a} \text{ gilt } P(\texttt{t})}$

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`

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 $P(\texttt{t})$

strukturelle Induktion:

- Induktionsanfang:
  zeige $P(\texttt{t})$ für `t == Nil` oder `t == Leaf x` für alle `x :: a`
- Induktionsvoraussetzungen
  Seien `l, r :: BinTree a` zwei Bäume, sodass $P(\texttt{l})$ und $P(\texttt{r})$ gilt.

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 Induktionsvoraussetzungen

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!

FEHLERQUELLEN

- kein Induktionsprinzip
- IV wird im Induktionsschritt nicht verwendet
- fehlende Quantifizierung (nur Gleichungen bringen kaum Punkte)
- *Missachtung freier Variablen*

FEHLERQUELLEN

- kein Induktionsprinzip
- IV wird im Induktionsschritt nicht verwendet
- fehlende Quantifizierung (nur Gleichungen bringen kaum Punkte)
- *Missachtung freier Variablen*
- zu beweisende Eigenschaft P wird für `xs` angenommen, um sie dann im Induktionsschritt nochmal für `xs` zu beweisen — eine Tautologie
- Annahme, dass P bereits für alle Listen gilt, um es dann für `x:xs` nochmal zu zeigen

ENDE

Fragen?