

5. Induktive Strukturen und Induktion

Übersicht über den geplanten Inhalt:

- Motivation: weshalb braucht man Induktion?
- Induktive Strukturen und strukturelle Induktion
 - Gleichungstheorie und induktive Theorie
 - Rekursiv definierte Funktionen und Beweis ihrer Eigenschaften
- Fundierte Relationen und Noethersche Induktion
 - Termination rekursiv definierter Funktionen
- Beispiele

Warum braucht man Induktion?

1. Ein Induktionsaxiom/-schema schließt Nichtstandard-Modelle aus

(s.o. Logik höherer Stufe)

Ein *Induktionsschema* steht für eine (unendliche) Menge von (PL1-)Instanzen des Induktionsprinzips; ein Induktionsaxiom (in der Form einer PL2-Formel) drückt dasselbe etwas direkter (und sauberer) aus.

2. Mit Hilfe von *Induktion als Beweisprinzip* können Aussagen bewiesen werden, die anders (d.h. in PL1) nicht bewiesen werden können.

Z.B. Prinzip der *mathematischen Induktion* über den natürlichen Zahlen *Nat*:

$$P[0] \wedge (\forall x : Nat. P[x] \Rightarrow P[s(x)]) \Rightarrow (\forall x : Nat. P[x])$$

für beliebige Formel $P[x]$: Induktionsschema

Beispiel für Notwendigkeit von Induktion:

Sei R das folgenden Termersetzungssystem (über einer entsprechenden Signatur; vgl. frühere Gleichungssysteme im Kapitel über Termersetzung):

$$(a) \quad x + 0 \rightarrow x$$

$$(b) \quad x + s(y) \rightarrow s(x + y)$$

R ist terminierend und konfluent. Die Gleichung

$$(c) \quad (x + y) + z = x + (y + z)$$

läßt sich mit R nicht ableiten: Falls (c) in der Menge der aus (a), (b) ableitbaren Gleichungen läge, müßte gelten

$$(x + y) + z \leftrightarrow_R^* x + (y + z)$$

und damit auch

$$(x + y) + z \rightarrow_R^* t \quad x + (y + z) \rightarrow_R^* t$$

für ein t . Weder die linke noch die rechte Seite sind mit R reduzierbar, d.h. ein solches t gibt es nicht.

Andrerseits gilt: *Jede* Grundinstanz von (c) läßt sich mit R ableiten.

Beispiel: Peano-Arithmetik

Die Peano-Arithmetik (elementare Arithmetik; nach dem ital. Mathematiker Peano, 1889) ist die Theorie der natürlichen Zahlen mit Addition und Multiplikation, \mathcal{N}_{+*} , definiert durch das Axiomensystem PA :

$$\forall x. \neg(0 = s(x)) \quad (\text{constr})$$

$$\forall x, y. s(x) = s(y) \Rightarrow x = y \quad (\text{inj})$$

$$\forall x. x + 0 = x \quad (\text{add0})$$

$$\forall x, y. x + s(y) = s(x + y) \quad (\text{add1})$$

$$\forall x. x * 0 = 0 \quad (\text{mul0})$$

$$\forall x, y. x * s(y) = x * y + x \quad (\text{mul1})$$

$$\forall P : (\mathcal{N} \rightarrow \text{Bool}). \quad (\text{induct})$$

$$P(0) \wedge (\forall x. P(x) \Rightarrow P(s(x))) \Rightarrow \forall x. P(x)$$

plus Axiome über Gleichheit

Peano-Arithmetik (2)

Das Axiom (induct) definiert das *Induktionsprinzip* für natürliche Zahlen:

Um eine Formel $\forall x. Q(x)$ zu beweisen, muß gezeigt werden

- (i) $Q(0)$ (Induktions-Anfang)
- (ii) $Q(s(x))$ unter der Annahme $Q(x)$, für beliebiges x (Induktions-Schritt)

Man kann zeigen, daß obige Axiomatisierung \mathcal{N}_{+*} bis auf Isomorphie charakterisiert.

Die in PA gültigen Sätze sind nicht rekursiv aufzählbar.

Insbesondere ist also PA nicht entscheidbar.

Es gibt entscheidbare Untertheorien von PA , z.B. die *Presburger-Arithmetik*, die nur lineare Arithmetik (keine Multiplikation) enthält.

Peano-Arithmetik: Beispielbeweis

Beweise die Formel $Q := \forall x. (x + 1 = 1 + x)$ aus den Axiomen PA :

Beweisstruktur:

$$\frac{\frac{\frac{PA \vdash IA \quad PA \vdash IS}{PA \vdash IA \wedge IS}}{PA, (IA \wedge IS \Rightarrow Q) \vdash Q}}{PA, \forall P. P(0) \wedge (\forall x. Px \Rightarrow P(s(x))) \Rightarrow \forall x. P(x) \vdash Q}^{(*)}$$

Bei dem Schritt (*) ist für P einzusetzen die Funktion

$$\sigma := \lambda z. (z + 1 = 1 + z)$$

Dadurch wird $P(0)$ zum Induktions-Anfang:

$$IA := 0 + 1 = 1 + 0$$

$\forall x. P(x) \Rightarrow P(s(x))$ wird zum Induktions-Schritt

$$IS := \forall x. (x + 1 = 1 + x \Rightarrow s(x) + 1 = 1 + s(x))$$

$\forall x. P(x)$ wird gleich der zu beweisenden Formel Q :

$$\forall x. x + 1 = 1 + x$$

Peano-Arithmetik: Beispielbeweis (2)

Die einzigen verbleibenden Beweisziele sind:

$$PA \vdash IA$$

und

$$PA \vdash IS$$

Sie lassen sich mit Hilfe der übrigen Axiome beweisen, d.h. ohne Anwendung von Induktion.

Bemerkung: Formal gesehen ist der Beweis nur in PL2 möglich, aber Elemente der PL2 werden nur für die Instantiierung der Induktionsformel (und für die zugehörigen Inferenzen) benötigt; der Rest des Beweises ist dann wie in einem “normalen” PL1-Beweis.

Induktive Theorie

Unterschied zwischen der *Gleichungstheorie* und der *induktiven Theorie* für eine Menge E von Gleichungen über einer Signatur Σ :

Gleichungstheorie $Th(\Sigma, E)$: Gleichungen, die sich aus E ableiten lassen (z.B. mit Hilfe eines vollständigen TES)

Induktive Theorie $ITh(\Sigma, E)$: Menge der Gleichungen, für die jede *Grundinstanz* in der Gleichungstheorie liegt:

$$ITh(\Sigma, E) := \{ l = r \mid \sigma(l) = \sigma(r) \in Th(\Sigma, E) \\ \text{für alle Grundsubstitutionen } \sigma \}$$

Gleichungen, die in der induktiven Theorie, aber nicht in der Gleichungstheorie liegen, lassen sich mit Hilfe von *Induktion* beweisen, sofern ein geeignetes Induktionsprinzip gefunden (und bewiesen) werden kann.

Induktive Theorie (2)

Für jede induktiv definierte Struktur gibt es ein Induktionsschema analog zu dem für Peano-Arithmetik, dem Schema der “mathematischen Induktion”:

Das Schema folgt dem induktiven Aufbau der Struktur, daher: *strukturelle Induktion*

Induktion als Beweisprinzip ist im allgemeinen notwendig für Beweise über induktiv definierte (oder “rekursive”) Datenstrukturen und rekursive Funktionen.

Abstrakte Datentypen in PVS

Abstrakte Datentypen (ADT's) drücken die *Struktur* von Daten aus, ohne daß deren Implementierung festgelegt wird. Wir interessieren uns hier nur für induktiv definierte Datenstrukturen.

(Im folgenden wird ein Typ *Bool* vorausgesetzt, entsprechend der Vereinbarung in der mehrsortigen Logik.)

Statt für jeden einzelnen ADT ein den Peano-Axiomen entsprechendes Axiomensystem angeben zu müssen, reicht es in PVS, eine Datentyp-Deklaration zu geben, aus der das System die weiteren notwendigen Deklarationen und Axiome generiert.

Es gibt keine einheitliche Notation bzw. Syntax für die Spezifikation von ADT's; im Rahmen dieser Vorlesung orientieren wir uns an der PVS-Syntax.

ADTs in PVS: Beispiel Nat

Beispiel: abstrakter Datentyp Nat
(entsprechend der oben gegebenen Axiomatisierung)

```
Nat:  DATATYPE
      BEGIN    nul: nul?
              suc(prd: Nat): suc?
      END Nat
```

Die intendierte Bedeutung einer solchen Deklaration wird in PVS durch den hieraus automatisch generierten Theorie-Modul Nat_adt ausgedrückt.

Abstrakte Datentypen in PVS (2)

Deklaration von Typ und Signaturen:

```
Nat_adt: THEORY
  BEGIN
    Nat: TYPE
    nul?, suc?: [Nat -> boolean]           % Erkenner
    nul: (nul?)                             % Konstruktor
    suc: [Nat -> (suc?)]                   % Konstruktor
    prd: [(suc?) -> Nat]                   % Selektor
```

Gleichheit (=) muss nicht deklariert werden – = ist in PVS für alle Typen vordefiniert.

Die Typ-Spezifikation nutzt eine PVS-Konvention aus: Prädikate über einem Basistyp werden mit *Teilmengen* der Trägermenge des Basistyps identifiziert.

Entsprechend wird ein Prädikat wie `suc?` in die Bezeichnung eines Untertyps des Basistyps `Nat` durch die Notation “`(suc?)`” verwandelt.

Abstrakte Datentypen in PVS (3)

- `nul` und `suc` sind *Konstruktoren* oder *Generatoren*: sie erzeugen die Elemente des Datentyps. Intuitiv:

$$\begin{array}{lcl} \text{nul} & \rightsquigarrow & 0 \\ \text{suc}(\text{nul}) & \rightsquigarrow & 1 \\ \text{suc}(\text{suc}(\text{nul})) & \rightsquigarrow & 2 \\ & \dots & \\ \text{suc}^{(n)}(\text{nul}) & \rightsquigarrow & n \end{array}$$

- `prd` ist ein *Selektor* oder *Akzessor*: für den Zugriff auf eine Komponente eines Datenelements.
- `nul?` und `suc?` sind *Klassifikatoren* (*recognizers*) für die Unterscheidung zwischen verschiedenen Alternativen von (Klassen von) Elementen des Datentyps.

Abstrakte Datentypen in PVS (4)

Von der Datentyp-Deklaration implizierte Axiome:

- Ein Element vom Typ `Nat` ist entweder `nu1` oder von der Form `suc(n)`, d.h. `nu1` und `suc` konstruieren *alle* Elemente von `Nat`.
`Nat` ist die disjunkte Vereinigung der durch die Alternativen beschriebenen Teilmengen (Teiltypen).

```
Nat_inclusive: AXIOM
  (FORALL (Nvar: Nat): nu1?(Nvar) OR suc?(Nvar));
```

- Die Konstruktoren sind injektiv:

```
nu1_injektiv: AXIOM
  (FORALL (nvar: (nu1?), nvar2: (nu1?)):
    nvar = nvar2);
```

```
suc_injektiv: AXIOM
  (FORALL (svar: (suc?), svar2: (suc?)):
    prd(svar) = prd(svar2) IMPLIES svar = svar2);
```

Abstrakte Datentypen in PVS (5)

- Zusammen bedeutet dies, daß Nat *frei erzeugt* wird von nu1 und suc und daß alle Werte durch die Terme nu1 , $\text{suc}(\text{nu1})$, . . . repräsentiert werden; diese entsprechen dem intendierten Standardmodell, der Menge der natürlichen Zahlen.
- Axiom Nat_prd_suc drückt die übliche Beziehung zwischen Konstruktor und zugehörigem Selektor aus.

```
Nat_prd_suc: AXIOM
  (FORALL (var: Nat): prd(suc(var)) = var);
```

Frage: was ist $\text{prd}(\text{nu1})$?

Abstrakte Datentypen in PVS (6)

Disjunktheit der Alternativen. d.h. das Äquivalent der Formel

```
(FORALL (Nvar: Nat): NOT(nul?(Nvar) AND suc?(Nvar)))
```

ist in PVS aus Effizienzgründen in den Beweiser eingebaut. Damit ist z.B. die in der ADT-Theorie vordefinierte Funktion `ord` wohldefiniert:

```
ord(x: Nat): upto(1) =  
  CASES x OF nul: 0, suc(s1_var): 1 ENDCASES
```

Induktion über natürlichen Zahlen:

```
Nat_induction: AXIOM  
  (FORALL (p: [Nat -> boolean]):  
    p(nul) AND  
    (FORALL (var: Nat): p(var) IMPLIES p(suc(var))))  
  IMPLIES (FORALL (Nat_var: Nat): p(Nat_var));
```

Das Induktionsaxiom wird eingesetzt, indem es für ein geeignetes Prädikat über `Nat` instantiiert wird.

Beispielbeweise mit Induktion über Nat

1. Beispielbeweis: Assoziativität von +

Definition von + durch die Axiome

add0: AXIOM $x + \text{nul} = x$

add1: AXIOM $x + \text{suc}(y) = \text{suc}(x + y)$

Zu zeigen:

add_assoc: LEMMA (FORALL (x,y,z: Nat):
 $(x + y) + z = x + (y + z)$)

Anwendung des Induktionsschemas (der Induktionsformel) zur Generierung von Unterzielen.

Problem: welche der 3 Variablen soll als “Induktionsvariable”, d.h. als diejenige Variable, für die die Induktionsformel instantiiert wird, ausgewählt werden?

Heuristik: rekursive Definitionen der beteiligten Funktionen legt die Induktionsvariable nahe – in diesem Fall: die Variable z.

Beispiel: Assoziativität von + (2)

↪ Induktionsformel wird instantiiert mit dem Prädikat

$$P := (\text{lambda } z:\text{Nat}. (\text{FORALL } (x,y: \text{Nat}): \\ (x + y) + z = x + (y + z)))$$

Unterziele, wie sie von PVS generiert werden:

(i) Induktionsanfang:

add_assoc.1 :

$$\begin{array}{l} |----- \\ \{1\} \quad (\text{FORALL } (x, y: \text{Nat}): \\ \quad \quad x + (y + \text{nul}) = (x + y) + \text{nul}) \end{array}$$

Beispiel: Assoziativität von + (3)

(ii) Induktionsschritt:

add_assoc.2 :

```
|-----  
{1} (FORALL (suc1_var: Nat):  
      (FORALL (x, y: Nat):  
          x + (y + suc1_var) = (x + y) + suc1_var)  
      IMPLIES  
      (FORALL (x, y: Nat):  
          x + (y + suc(suc1_var))  
          = (x + y) + suc(suc1_var))))
```

Weitere Beweisschritte über Umformungen der Gleichungen, insbesondere durch Termersetzung mit Hilfe der Gleichungen, die + definieren.

Heuristiken für Induktionsbeweise

1. Auswahl des Induktionsarguments
2. Bei Teilbeweis für Induktionsschritt: Induktionsannahme wird als Voraussetzung angenommen.
3. Teilbeweis für Induktionsschritt muss so strukturiert werden, dass die Induktionsannahme benutzt werden kann
 \rightsquigarrow Schlußfolgerung (d.h. die zu beweisende Formel) so manipulieren, dass die Induktionsannahme “paßt”.

Erforderlich: Techniken der Umformung von Formeln

Ziel: syntaktischen “Abstand” zwischen Induktionsannahme und Induktionsschlußfolgerung verringern.

Techniken bzw. Heuristiken für die Umformung der Schlußfolgerung: z.B.

- Symbolische Auswertung
- Verschieben von Kontexten (“*rippling*”)

Heuristiken für Induktionsbeweise (2)

Symbolische Auswertung:

z.B. Anwendung der die Funktionen definierenden (bedingten) Gleichungen
“*unfolding*” einer rekursiven Definition

“*Rippling*”:

Verschieben von Kontexten nach außen oder seitwärts

Allgemeine Form von Termersetzungsregeln:

$$F(S(U)) \rightarrow T(F(U))$$

F , S , T sind Terme mit einem “Loch” (Argument);
 T kann auch leer (d.h. die Identität) sein.

Beispiel: $s(x) + y \rightarrow s(x + y)$

$$U \doteq x, \quad S \doteq T \doteq s(-), \quad F \doteq - + y$$

2. Beispiel: Kommutativität von $+$

mit vereinfachter Notation: 0 für `nu1`, s für `suc` usw.

$$\forall x, y : Nat. x + y = y + x$$

Induktionsvariable: beliebig, da symmetrisch

(i) *Induktionsanfang*: $0 + y = y + 0$

vereinfacht zu $0 + y = y$

keine vorhandene Regel zur Vereinfachung/Umformung anwendbar

\rightsquigarrow neue Induktion mit Induktionsvariable y :

Ind.-Anfang: $0 + 0 = 0$, reduziert zu $0 = 0$

Induktionsannahme: $0 + y_1 = y_1$

zu zeigen $0 + s(y_1) = s(y_1)$

mit *rippling* und Ind.-Annahme: $0 + s(y_1) = s(0 + y_1) = s(y_1)$

(ii) *Induktionsannahme*: $x_1 + y = y + x_1$

zu zeigen: $s(x_1) + y = y + s(x_1)$

mit *rippling* und Ind.-Annahme folgt für die rechte Seite

$$y + s(x_1) = s(y + x_1) = s(x_1 + y)$$

Kommutativität von $+$ (Forts.)

\rightsquigarrow neue Induktion mit Induktionsvariable y für

$$s(x_1) + y = s(x_1 + y)$$

Ind.-Anf.: $s(x_1) + 0 = s(x_1 + 0)$, reduziert zu $s(x_1) = s(x_1)$

Ind.-Ann.: $s(x_1) + y_1 = s(x_1 + y_1)$

zu zeigen: $s(x_1) + s(y_1) = s(x_1 + s(y_1))$

linke Seite: $s(x_1) + s(y_1) = s(s(x_1) + y_1)$

rechte Seite: $s(x_1 + s(y_1)) = s(s(x_1 + y_1))$

mit Ind.-Annahme ergibt sich die gesuchte Gleichung.

Bemerkung: Die beiden “inneren” Induktionen sind für Gleichungen der Form

$$0 + x = x \quad s(y) + x = s(y + x)$$

die auch für sich allein nützlich sind und als separate Lemmata (Vereinfachungsregeln) bewiesen und abgelegt werden können.

(Orientierung als Ersetzungsregel?)

3. Beispiel: Distributivität von $*$ über $+$

$$x * (y + z) = x * y + x * z$$

Definitionen:

$$0 * v = 0 \quad s(u) * v = v + u * v$$

außerdem als Lemmata:

$$x + (y + z) = (x + y) + z$$

$$x + y = y + x$$

Induktion mit Induktionsvariable x (warum?)

1. *Induktionsanfang*: $0 * (y + z) = 0 * y + 0 * z$
durch einfache Vereinfachung gezeigt.

2. *Induktionsschritt*:

Induktionsannahme:

$$x_1 * (y + z) = x_1 * y + x_1 * z$$

zu zeigen: $s(x_1) * (y + z) = s(x_1) * y + s(x_1) * z$

Beispiel: Distributivität von $*$ über $+$ (Forts.)

linke Seite:

$$\begin{aligned} & s(x_1) * (y + z) \\ &= (y + z) + x_1 * (y + z) \\ &= (y + z) + (x_1 * y + x_1 * z) \quad \text{mit Ind. Ann.} \end{aligned}$$

rechte Seite:

$$\begin{aligned} & s(x_1) * y + s(x_1) * z \\ &= (y + x_1 * y) + (z + x_1 * z) \\ &= (y + z) + (x_1 * y + x_1 * z) \\ & \text{mit Assoz. und Kommut. von } + \text{ (Seitwärtsverschiebung)} \end{aligned}$$