Mathematical Reasoning (mathematisches Begründen)

Prof. Dr. Josef F. Bürgler

Studiengang Informatik
Hochschule Luzern, Informatik

I.BA_DMATH

Zum Einstieg

Thema: Induktionsbeweis, Rekursion, Inferenzregeln

Ziele: Sie können den Induktionsbeweis auf verschiedene Beweis-Probleme anwenden. Sie verstehen rekursiv definierte Funktionen und rekursive Algorithmen. Sie können die Inferenzregeln praktisch anwenden.

Resultate: Dank der gelernten Methoden können Sie viele weitere mathematische Fragestellungen, wie sie im Informatikalltag auftreten, beweisen.

Vorgehen: Mittels verschiedener Beispiele wird die Theorie vermittelt und vertieft.

Inhalt

Übersicht

- **Direkter Beweis:** Man zeigt, dass $p \to q$ wahr ist, indem zeigt, dass aus $p = \texttt{WAHR}$ sofort $q = \texttt{WAHR}$ folgt, d.h. die Kombination $p = \texttt{WAHR}$ und $q = \texttt{FALSCH}$ kommt nicht vor (siehe Beweis, dass aus "n *ungerade*, sofort "n^2 *ungerade*" folgt).
- **Beweis durch Kontraposition:** Es handelt sich um einen indirekten Beweis bei welchem man verwendet, dass $p \to q$ äquivalent zur Kontraposition $\neg q \to \neg p$ ist (siehe Beweis, dass mit $3n + 2$ ungerade, auch n ungerade ist).
- **Beweis durch Widerspruch:** Wir möchten zeigen, dass p wahr ist. Nehmen wir an, wir finden einen Widerspruch q so, dass $\neg p \to q$ wahr ist. Weil q falsch ist, aber $\neg p \to q$ wahr, muss notwendigerweise $\neg p$ falsch sein (Wahrheitstabelle!), d.h. p muss wahr sein (siehe Beweis, dass $\sqrt{2} \notin \mathbb{Q}$).

Übersicht

Induktionsbeweis

Viele Sätze sagen aus, dass eine bestimmte propositionale Funktion $P(n)$ wahr ist für $n \in \mathbb{N}$, d.h. für alle positiven Zahlen n.

Der Beweis, dass $P(k)$ wahr ist $\forall k \in \mathbb{N}$ erfolgt in zwei Schritten:

Induktionsverankerung: Es wird gezeigt, dass $P(1)$ wahr ist.

Induktionsschritt: Es wird gezeigt, dass die Implikation $P(k) \to P(k+1)$ wahr ist $\forall k \geqslant 1$.

Man nennt $P(k)$ auch **Induktionshypothese**.

Die Kurzform des Induktionsbeweises heisst:

$$[P(1) \wedge \forall k\,(P(k) \to P(k+1))] \to \forall n\, P(n)$$

Einige Beispiele zum Induktionsbeweis

Example

Beweisen Sie die folgenden Aussagen durch Induktion.

1. $\forall n \in \mathbb{N} \left(1 + 2 + 3 + \cdots + n = \frac{n(n+1)}{2}\right)$
2. $\forall n \in \mathbb{N} \left(7^0 + 7^1 + \cdots + 7^{n-1} = \frac{7^n - 1}{7 - 1}\right)$
3. Eine ungerade Anzahl Personen stehen in jeweils paarweise unterscheidlicher Distanz zueinander. Exakt zum Zeitpunkt $t_0 = 0$ bewirft jeder seinen nächsten Nachbarn mit einer Torte. Zeigen Sie, dass dabei eine Person *verschont* wird. Bei einer geraden Anzahl Personen kann man das nicht zeigen!
4. Entfernt man in einem $2^n \times 2^n$ Schachbrett ein Feld, dann lässt sich der Rest mit (Rechts-) Triominos [a] vollständig zudecken.

[a] Ein Triomino (od. Trimino) ist ein L-förmiges Gebiet bestehend aus 3 Quadraten.

Übersicht

Rekursiv definierte Funktionen

Definition

Ist f eine Funktion mit Definitionsbereich $D(f) = \mathbb{Z}^+ \cup \{0\}$, für die $f(0)$ definiert ist und für die eine Vorschrift existiert, die den Wert $f(k)$ aus $f(k-1)$, $f(k-2)$, ..., $f(1)$, $f(0)$ berechnet, dann wird diese Definition **rekursive** oder **induktive Definition** genannt.

Example (Die Fibonacci Zahlen)

Die Fibonacci Zahlen f_0, f_1, f_2, ... werden rekursiv definiert durch

$$f_0 = 0 \text{ und } f_1 = 1$$
$$f_k = f_{k-1} + f_{k-2}, \text{ für } k = 2, 3, 4, \ldots.$$

Man berechne die ersten 6 Glieder dieser Zahlenfolge.

Rekursiv definierte Funktionen (Fort.)

Example

Beweisen Sie die folgende Behauptung über die Fibonacci-Zahlen durch Induktion: Für alle $n \geqslant 1$ gilt

$$f_1^2 + f_2^2 + \cdots + f_n^2 = f_n \cdot f_{n+1}$$

Übersicht

Rekursive Algorithmen

Definition

Ein Algorithmus heisst **rekursiv**, wenn er ein Problem löst, indem er dazu ein (oder mehrere) gleiche, aber kleinere Probleme löst.

Wir betrachten folgende rekursive Algorithmen:

- Fakultät $n!$ sowohl rekursiv, wie auch iterativ
- n. Potenz a^n
- ggT
- Fibonacci Zahlen ebenfalls rekursiv und iterativ

Example (Iterative Fakultät)

Wir nehmen an, der Funktion `factorial` werde eine Zahl $n \in \mathbb{N}_0$ übergeben!

```
int factorial(int n) {
  int fact = 1;
  for (int i = 1; i <= n; i++) {
    fact = fact*i;
  }
  return fact;
}
```

Beachte: die Funktion ruft sich nicht selber auf. Zudem kommt auch für $n = 0$ das richtige Resultat raus.

Example (Rekursive Fakultät)

Wir nehmen an, der Funktion `factorial` werde eine Zahl $n \in \mathbb{N}_0$ übergeben!

```
int factorial(int n) {
  if ( n == 0 ) {
    return 1;
  } else {
    return n*factorial(n-1);
  }
}
```

Beachte: die Funktion ruft sich immer wieder selber auf; allerdings nicht unendlich lange, sondern nur bis das Argument Null wird!

Übersicht

Schlussregeln (Inferenzregeln, rules of inference)

Bei mathematischen Beweisen geht es darum mit gültigen Argumenten die Richtigkeit einer mathematischen Aussage zu begründen.

Ein **Argument** ist eine Folge von Aussagen, die mit einer **Folgerung** (Konklusion) enden.

gültig heisst, dass die Folgerung (Konklusion) aus der Wahrheit der vorhergehenden Aussagen (den **Prämissen**) des Arguments folgt.

Ein Argument ist also dann und nur dann gültig, wenn es unmöglich ist, dass alle Prämissen wahr sind, die Folgerung aber falsch ist.

Frage: Was sind gültige Argumente in der propositionalen Logik?

Schlussregeln (Fort.)

Example

Betrachte folgendes Argument welches Propositionen einbezieht

- "Falls Sie ein aktuelles Passwort haben, dann können Sie sich ins Netzwerk einloggen"
- "Sie haben eine aktuelles Passwort"
- Demzufolge
- "Sie können sich ins Netzwerk einloggen"

Ist das ein gültiges Argument? Wir möchten bestimmen, ob die Folgerung "Sie können sich ins Netzwerk einloggen" wahr ist, wenn die Prämissen "Falls Sie ein aktuelles Passwort haben, dann können Sie sich ins Netzwerk einloggen" und "Sie haben eine aktuelles Passwort" beide wahr sind!

Bevor wir die Gültigkeit dieses bestimmten Argumentes diskutieren wollen wir eine Kurzschreibweise einführen.

Schlussregeln (Fort.)

Example (Fort.)

Seien also

- p ="Sie haben ein aktuelles Passwort"
- q ="Sie können sich ins Netzwerk einloggen"

Dann kann man das Argument in folgender Form schreiben

$$\begin{array}{ll} & p \to q \\ & p \\ \hline \therefore & q \end{array}$$

Das Symbol $\therefore$ steht für "**demzufolge**" oder "**folglich**".

Schlussregeln (Fort.)

Wir wissen, dass $((p \to q) \wedge p) \to q$ eine Tautologie ist (überprüfen Sie das). Wenn also $p \to q$ und p wahr sind, dann muss auch q wahr sein!

Sobald also alle Prämissen (die Aussagen des Arguments ausser dem letzten) wahr sind, dann muss auch die Folgerung bzw. die Konklusion (die letzte Aussage im Argument) wahr sein. Eine solche Form des Arguments nennt man gültig.

Man hat also folgende Schlussregeln (Inferenzregeln):

- **Modus ponens** (Abtrennungsregel)

$$\begin{array}{l} p \\ p \to q \\ \hline \therefore \quad q \end{array} \qquad \text{Tautologie}: \ [p \wedge (p \to q)] \to q$$

Schlussregeln (Fort.)

- **Modus tollens** (Aufhebender Modus)

$$\begin{array}{r} \neg q \\ p \to q \\ \hline \therefore \quad \neg p \end{array}$$

Tautologie : $[\neg q \wedge (p \to q)] \to \neg p$

- **Hypothetischer Syllogismus**

$$\begin{array}{r} p \to q \\ q \to r \\ \hline \therefore \quad p \to r \end{array}$$

Tautologie : $[(p \to q) \wedge (q \to r)] \to (p \to r)$

- **Disjunktiver Syllogismus**

$$\begin{array}{r} p \vee q \\ \neg p \\ \hline \therefore \quad q \end{array}$$

Tautologie : $[(p \vee q) \wedge \neg p] \to q$

Schlussregeln (Fort.)

- **Addition**

$$\frac{p}{\therefore \; p \vee q}$$

Tautologie : $p \to (p \vee q)$

- **Simplifikation**

$$\frac{p \wedge q}{\therefore \; q}$$

Tautologie : $(p \wedge q) \to p$

- **Konjunktion**

$$\frac{\begin{array}{l} p \\ q \end{array}}{\therefore \; p \wedge q}$$

Tautologie : $[(p) \wedge (q)] \to (p \wedge q)$

Schlussregeln (Fort.)

- **Resolution**

$$\begin{array}{l} p \vee q \\ \neg p \vee r \\ \hline \therefore \quad q \vee r \end{array}$$

Tautologie : $[(p \vee q) \wedge (\neg p \vee r)] \rightarrow (q \vee r)$

Example (Anwendungsbeispiel)

Zeige dass die Hypothesen "*es ist heute Nachmittag nicht sonnig und es ist kälter als gestern*", "*wir gehen nur schwimmen falls es sonnig ist*", "*wenn wir nicht schwimmen gehen, machen wir eine Kanufahrt*" und "*wenn wir eine Kanufahrt machen, werden wir bis Sonnenuntergang zu Hause sein*" zum Schluss führen: "*wir werden bis Sonnenuntergang zu Hause sein*"!

Schlussregeln (Fort.)

Example (Anwendungsbeispiel (Fort.))

Wir verwenden:

$p =$ "Es ist heute Nachmittag sonnig" $\quad q =$ "Es ist kälter als gestern"

$r =$ "Wir gehen schwimmen" $\quad s =$ "Wir machen eine Kanufahrt"

$t =$ "Wir sind bis Sonnenuntergang zu Hause"

Dann lauten die Hypothesen

$\neg p \wedge q =$ "es ist heute Nachmittag nicht sonnig und es ist kälter als gestern"

$r \to p =$ "wir gehen nur schwimmen falls es sonnig ist"

$\neg r \to s =$ "wenn wir nicht schwimmen gehen, machen wir eine Kanufahrt"

$s \to t =$ "wenn wir eine Kanufahrt machen, werden wir bis Sonnenuntergang zu Hause sein"

Schlussregeln (Fort.)

Example (Anwendungsbeispiel (Fort.))

Wir müssen zeigen, dass aus diesen Hypothesen durch Anwendung der Schlussregeln, der Schluss (bzw. die Konklusion) t folgt.

1.	$\neg p \wedge q$	Hypothese
2.	$\neg p$	Simplifikation unter Verwendung von (1)
3.	$r \to p$	Hypothese
4.	$\neg r$	Modus tollens unter Verwendung von (2) und (3)
5.	$\neg r \to s$	Hypothese
6.	s	Modus ponens unter Verwendung von (4) und (5)
7.	$s \to t$	Hypothese
8.	t	Modus ponens unter Verwendung von (6) und (7)

Beachte: Man hätte das Problem auch mit einer Wahrheitstabelle für die fünf Variablen p, q, r, s und t lösen können. Diese Tabelle hätte aber $2^5 = 32$ Zeilen gehabt!

Übersicht

Definition (korrekt)

Ein Programm heisst **korrekt**, falls es den korrekten Output für jeden möglichen Input liefert. Ein Programm heisst **teilweise korrekt** (engl. **partially correct**), wenn es den korrekten Output liefert, falls es terminiert. Wenn das Programm zudem immer terminiert, dann heisst das Programm **korrekt**.

Definition (teilweise korrekt)

Ein Programmsegment S heisst **teilweise korrekt** bezüglich der Anfangsbedingung p und der Endbedingung q falls gilt: Ist p wahr für alle Eingaben von S und terminiert S, dann ist q wahr für die Ausgaben von S: man bezeichnet dies mit $\mathrm{p}\{S\}\mathrm{q}$ (*Hoare Tripel*, nach Tony Hoare).

Example

Zeigen Sie, dass das Programmsegment

$$y := 2,\ z := x + y$$

bezüglich der Anfangsbedingung $x = 1$ und der Endbedingung $z = 3$ teilweise korrekt ist.

Regeln des Schliessens

Hat man zwei Programmsegmente S_1 und S_2, dann gilt die folgende Regel für das zusammengesetzte Programmsegment S_1; S_2:

$$\frac{p\,\{S_1\}\,q \qquad q\,\{S_2\}\,r}{\therefore p\,\{S_1;\ S_2\}\,r}$$

Conditionale Statements

Das Programmsegment S_1 in

if (condition) **then**
S_1

wird ausgeführt, falls condition wahr ist. Also hat man:

$$\frac{(p \wedge \texttt{condition})\{S_1\}q \qquad (p \wedge \neg\texttt{condition}) \rightarrow q}{\therefore p\{\textbf{if } \texttt{condition } \textbf{then } S_1\}q}$$

Example

Zeigen Sie, dass das Programmsegment

$$\textbf{if } x > y \textbf{ then } y := x$$

bezüglich der Anfangsbedingung $\mathbf{T}$ und der Endbedingung $y \geqslant x$ teilweise korrekt ist.

Conditionale Statements (Fort.)

Das Programmsegment S_1 im folgenden Code

if `(condition)` **then**
  S_1
else
  S_2

wird ausgeführt, falls `condition` wahr ist; ansonsten S_2. Also gilt:

$$\frac{(p \wedge \texttt{condition})\,\{S_1\}\,q \qquad (p \wedge \neg\texttt{condition})\,\{S_2\}\,q}{\therefore p\,\{\textbf{if } \texttt{condition } \textbf{then } S_1 \textbf{ else } S_2\}\,q}$$

Conditionale Statements (Fort.)

Example

Zeigen Sie, dass das Programmsegment

$$\textbf{if } x < 0 \textbf{ then } abs := -x \textbf{ else } abs := x$$

bezüglich der Anfangsbedingung $\mathbf{T}$ und der Endbedingung $abs := |x|$ (der Betrag von x) teilweise korrekt ist.

Zusammenfassung

- Die StudentIn kennt grundlegende Prinzipien von Beweisen.
- Die StudentIn kann mit Folgen und endlichen Reihen umgehen und einfache, endliche Summen berechnen.
- Die StudentIn kann einfache Induktionsbeweise führen.
- Die StudentIn versteht rekursiv definitierte Funktionen, Mengen, Strukturen und Algorithmen.
- Die StudentIn kann einfach Programme auf Korrektheit untersuchen.