Informatik I

9. Nachweis von Programmeigenschaften

Jan-Georg Smaus

Albert-Ludwigs-Universität Freiburg

2. Dezember 2010

Informatik I

2. Dezember 2010 — 9. Nachweis von Programmeigenschaften

9.1 Nachweis von Programmeigenschaften

9.1 Nachweis von Programmeigenschaften

- Korrektheit: Ein verklausuliertes Programm
- Garantierte Terminierung
- Alternative Rekursion
- Listen

Ein verklausuliertes Programm

Was tut folgendes Programm?

```
(define h
  (lambda (n)
    (if (zero? n)
        0
        (+ (h (- n 1))
           (/ 1 (* n (+ n 1)))))))
```

Behauptung: $(\forall n)$ (h n) $= \frac{n}{n+1}$.

Zeige, dass Behauptung gilt.

Vorgehensweise

- Unsere Beweise sind eine Mischung aus Programmtext und mathematischer Syntax. Wir haben es nicht mit konkreten Ausdrücken zu tun, sondern Ausdrücken, in denen mathematische Variablen, also Variablen auf der Meta-Ebene vorkommen. Beispiel: `(+ 1 (n + 1))`.
- Wir werten diese abstrakten Ausdrücke gemäß dem Substitutionsmodell aus.
- Ohne Aufhebens gehen wir immer davon aus, dass die eingebauten mathematischen Operationen wie `+ * ...` das Übliche tun. Dies erlaubt uns z.B. darauf zu vertrauen dass `(+ 1 (n + 1))` zu $n+2$ reduziert wird.
- Auch „kleine“ Lemmata wie $n + m = m + n$ verwenden wir nach Belieben.
- Achten Sie sehr auf den Unterschied zwischen „`Programmtext () + *`“ und „*Mathematik* $() + \frac{i}{n}$“.

Nochmals das Programm

```
(define h
  (lambda (n)
    (if (zero? n)
        0
        (+ (h (- n 1))
           (/ 1 (* n (+ n 1)))))))
```

Behauptung: $(\forall n)$ `(h` n`)` $= \frac{n}{n+1}$.

Zeige, dass Behauptung gilt.

Induktionsanfang $P(0)$

$$\texttt{(h 0)} = \texttt{(if (zero? 0) 0 ...)} = 0 = \frac{0}{1}$$

gemäß Substitutionsmodell.

Induktionsschritt $P(n) \Rightarrow P(n+1)$

$$
\begin{aligned}
\texttt{(h } (n+1)\texttt{)} &= \texttt{(if (zero? } (n+1)\texttt{) 0 ...)} \\
&= \texttt{(+ (h (- } (n+1)\texttt{ 1))} \\
&\quad \texttt{(/ 1 (* } (n+1)\texttt{ (+ } (n+1)\texttt{ 1))))} \\
&= \texttt{(+ (h } n\texttt{)} \\
&\quad \texttt{(/ 1 (* } (n+1)\texttt{ } (n+2)\texttt{))))} \\
&\overset{\text{IH}}{=} \texttt{(+ } \frac{n}{n+1} \quad \frac{1}{(n+1)\cdot(n+2)}\texttt{)} \\
&= \frac{n\cdot(n+2)}{(n+1)\cdot(n+2)} + \frac{1}{(n+1)\cdot(n+2)} \\
&= \frac{n^2+2n+1}{(n+1)\cdot(n+2)} = \frac{(n+1)^2}{(n+1)\cdot(n+2)} \\
&= \frac{n+1}{n+2}
\end{aligned}
$$

Konstruktionsanleitung

Satz

Jede durch die Konstruktionsanleitung für natürliche Zahlen definierte Funktion liefert immer ein Ergebnis.

D.h. sei `(: f (natural -> natural))` definiert durch

```
(define f
  (lambda (n)
    (if (zero? n)
        basis
        (step (f (- n 1)) n))))
```

wobei *basis* ein Ausdruck mit Wert aus $\mathbb{N}$ und *step* eine totale (und berechenbare) Funktion `(natural natural -> natural)` ist.
Dann gilt: $(\forall n \in \mathbb{N})$ ist $(f\ n)$ definiert und $(f\ n) \in \mathbb{N}$.

Beweis

Induktionsbasis:

$$\begin{aligned}(f\ 0) &= (\texttt{if}\ (\texttt{zero?}\ 0)\ \mathit{basis}\ (\mathit{step}\ \ldots)) \\ &= \mathit{basis} \in \mathbb{N}\end{aligned}$$

Induktionsschritt $P(n) \Rightarrow P(n+1)$:

$$\begin{aligned}(f\ (n+1)) &= (\texttt{if}\ (\texttt{zero?}\ (n+1))\ \ldots) \\ &= (\mathit{step}\ (f\ (\texttt{-}\ (n+1)\ 1))\ (n+1)) \\ &= (\mathit{step}\ (f\ n)\ (n+1))\end{aligned}$$

Nach Induktionshypothese ist $(f\ n)$ definiert und gleich $f(n) \in \mathbb{N}$, also

$$= (\mathit{step}\ f(n)\ (n+1)) \in \mathbb{N}$$

nach Voraussetzung über *step*.

Primitive Rekursion

Unsere Konstruktionsanleitung, so abstrakt unter Verwendung von *basis* und *step* formuliert . . .

```
(define f
  (lambda (n)
    (if (zero? n)
        basis
        (step (f (- n 1)) n))))
```

. . . hat in der Berechnungstheorie (3. Semester) einen Namen: f ist eine primitiv rekursive Funktion (vorausgesetzt, *basis* und *step* sind primitiv rekursiv).

Abweichung von der Konstruktionsanleitung

Wir haben am Beispiel von `factorial` gesehen, dass Funktionen auf den natürlichen Zahlen, die von der Konstruktionsanleitung abweichen, evtl. nicht terminieren.

Trotzdem können Programme, die von der Konstruktionsanleitung abweichen, terminieren.

Man muss nicht unbedingt von n nach $n-1$ gehen ...

Noch ein Programm

```
(define l
  (lambda (n)
    (if (zero? n)
        0
        (+ 1 (l (quotient n 2))))))
```

also $\mathtt{l}(0) = 0$, $\mathtt{l}(n) = 1 + \mathtt{l}(\lfloor \frac{n}{2} \rfloor)$, wobei $\lfloor x \rfloor$ die größte ganze Zahl z mit $x \geq z$ ist.

n	0	1	2	3	4	5	6	7	8	9...	15	16	17	...
$\mathtt{l}(n)$	0	1	2	2	3	3	3	3	4	4...	4	5	5	...

Vermutung

$$\mathtt{l}(n) = \lceil \log_2(n+1) \rceil$$

wobei ist $\lceil x \rceil$ die kleinste ganze Zahl z mit $x \leq z$ ist.

Vorabbemerkung

Warum präsentiere ich ein Programm sodass

$$\mathtt{l}(n) = \lceil \log_2(n+1) \rceil$$

anstatt des einfacheren

$$\mathtt{l}(n) = \lceil \log_2(n) \rceil \ ?$$

Das wäre das Programm

```
(define l
  (lambda (n)
    (if (= n 1)
        0
        (+ 1 (l (quotient n 2))))))
```

Dies wäre keine totale Prozedur der Sorte `(natural -> natural)`. Die Eingabe 0, für die `l` undefinert ist, müssten wir gesondert abfangen und eine Fehlermeldung generieren.

Induktion

Wir kennen

Induktionsschema

Falls $P(0)$ (Induktionsbasis, -anfang)
und
$(\forall n)\ P(n) \Rightarrow P(n')$ (Induktionsschritt)
dann
$(\forall n \in \mathbb{N})\ P(n)$. ($P(n)$ Induktionshypothese, -behauptung)

Das vorliegende Programm erfordert eine andere Art von Induktion. Wir nutzen aus, dass

$$\{(\lfloor \tfrac{n}{2} \rfloor, n) \mid n \in \mathbb{N}^+\} \subseteq <$$

und „$<$“ wohlfundiert ist.

Erinnerung: Induktion und wohlfundierte Relationen

1. Induktionsbasis: $(\forall x \in A)\ ((\nexists y \in A)\ y\ R\ x) \Rightarrow P(x)$
2. Induktionsschritt: $(\forall x \in A)\ ((\forall y \in A)\ y\ R\ x \Rightarrow P(y)) \Rightarrow P(x)$

also in unserem Fall

1. Induktionsbasis: $P(0)$
2. Induktionsschritt: $(\forall x \in \mathbb{N})\ ((\forall y \in \mathbb{N})\ y < x \Rightarrow P(y)) \Rightarrow P(x)$, d.h., zeige $P(x)$ wobei man für alle $y < x$ verwenden kann dass $P(y)$.

Beweis

Induktionsbasis $n = 0$

$$
\begin{aligned}
\texttt{(l 0)} &= \texttt{(if (zero? 0) 0 ...)} \\
&= 0 \\
&= \lceil \log_2 1 \rceil \\
&= \lceil \log_2 (0 + 1) \rceil
\end{aligned}
$$

Beweis

Induktionsschritt: $n > 0$

$$
\begin{aligned}
\texttt{l}(n) &= \texttt{(if (zero? } n\texttt{) 0 ...)} \\
&= \texttt{(+ 1 (l (quotient } n\texttt{ 2)))} \\
&= \texttt{(+ 1 (l } \lfloor \tfrac{n}{2} \rfloor\texttt{))} \\
&\overset{\text{IH}}{=} \texttt{(+ 1 } \lceil \log_2(\lfloor \tfrac{n}{2} \rfloor + 1) \rceil\texttt{)} \\
&= 1 + \lceil \log_2(\lfloor \tfrac{n}{2} \rfloor + 1) \rceil \\
&= \lceil 1 + \log_2(\lfloor \tfrac{n}{2} \rfloor + 1) \rceil \\
&= \lceil \log_2(2 \cdot (\lfloor \tfrac{n}{2} \rfloor + 1)) \rceil \\
&= \lceil \log_2(2 \cdot \lfloor \tfrac{n}{2} \rfloor + 2) \rceil
\end{aligned}
$$

Es geht gleich weiter ...

Beweis

Induktionsschritt: $n > 0$ (Fortsetzung 1)

Wir haben bisher gezeigt:

$$1(n) = \lceil \log_2(2 \cdot \lfloor \tfrac{n}{2} \rfloor + 2) \rceil$$

1. Fall: $n = 2k$ mit $k > 0$

$$\begin{aligned} &= \lceil \log_2(\lfloor 2 \cdot \tfrac{2k}{2} \rfloor + 2) \rceil \\ &= \lceil \log_2(2k + 2) \rceil \\ &\overset{(*)}{=} \lceil \log_2(2k + 1) \rceil \\ &= \lceil \log_2(n + 1) \rceil \end{aligned}$$

Beweis

Nebenrechnung (*): $\lceil \log_2(2k+2) \rceil = \lceil \log_2(2k+1) \rceil$ für alle $k > 0$

Wie jede natürliche Zahl ≥ 3 liegt $2k+2$ zwischen zwei benachbarten Zweierpotenzen, d.h., es gibt ein $j \in \mathbb{N}^+$ so dass

$$2^j < 2k+2 \leq 2^{j+1}. \tag{1}$$

Da $2k+2 \in \mathbb{N}$, folgt hieraus $2^j \leq 2k+1 < 2^{j+1}$.
Da $2^j = 2k+1$ unmöglich ist, kann man verschärfen:

$$2^j < 2k+1 < 2^{j+1}. \tag{2}$$

Nun wenden wir auf (1) und (2) den Zweierlogarithmus an

$$j < \log_2(2k+2) \leq j+1$$
$$j < \log_2(2k+1) < j+1.$$

Hieraus folgt

$$\lceil \log_2(2k+2) \rceil = j+1$$
$$\lceil \log_2(2k+1) \rceil = j+1.$$

Beweis

Induktionsschritt: $n > 0$ (Fortsetzung 2)

Wir haben bisher gezeigt:

$$1(n) = \lceil \log_2(2 \cdot \lfloor \tfrac{n}{2} \rfloor + 2) \rceil$$

2. Fall: $n = 2k + 1$ mit $k \geq 0$

$$\begin{aligned} &= \lceil \log_2(2 \cdot \lfloor \tfrac{2k+1}{2} \rfloor + 2) \rceil \\ &= \lceil \log_2(2k + 2) \rceil \\ &= \lceil \log_2(n + 1) \rceil \end{aligned}$$

Erinnerung: Peano-Axiome

Definition (Peano-Axiome)

Die Menge $\mathbb{N}$ der natürlichen Zahlen ist definiert durch:

- P1 $0 \in \mathbb{N}$ (Null)
- P2 $(\forall n \in \mathbb{N})\ n' \in \mathbb{N}$ (Nachfolger)
- P3 $(\forall n \in \mathbb{N})\ n' \neq 0$ (Nachfolger ist nicht Null)
- P4 $(\forall m, n \in \mathbb{N})\ m \neq n \Rightarrow m' \neq n'$ (Nachfolger ist injektiv)
- P5 Für jede Menge $M \subseteq \mathbb{N}$ mit $0 \in M$ und $(\forall n)\ n \in M \Rightarrow n' \in M$ gilt $M = \mathbb{N}$ (Induktionsaxiom)

Listen

Definition

Die Menge M^* der Listen über M ist induktiv definiert (P3+P4) durch

1. $\texttt{empty} \in M^*$ (die leere Liste) (P1)
2. $(\forall a \in M)\ (\forall xs \in M^*)\ (\texttt{cons}\ \ a\ \ xs) \in M^*$
   (nichtleere Liste mit Kopf a und Rumpf (Rest) xs) (P2)
3. In M^* sind nur die durch (1) und (2) definierten Elemente enthalten (P5)

Dies ist analog zu den Peano-Axiomen.
Allgemein bezeichnet Σ^* die Menge der Wörter über dem Alphabet Σ.

Listeninduktion

Definition (Das Schema der Listeninduktion)

Sei $P(xs)$ eine Eigenschaft von Listen über M. Es gilt $(\forall xs \in M^*)\ P(xs)$ genau dann, wenn

1. Induktionsbasis:
   $P(\texttt{empty})$.
2. Induktionsschritt:
   $(\forall a \in M)\ (\forall xs \in M^*)\ P(xs) \Rightarrow P(\texttt{(cons}\ a\ xs\texttt{)})$.

Beispiel: Listenverkettung

```
(: append ((list-of %a) (list-of %a) -> (list-of %a)))
(define append
  (lambda (xs ys)
    (cond
      ((empty? xs)
       ys)
      ((cons? xs)
       (cons (first xs) (append (rest xs) ys))))))
```

Die Prozedur ist auf dem ersten Argument rekursiv.

Eigenschaften von append

1. (append empty *bs*) = *bs*
2. (append *as* empty) = *as*
3. (append (append *as* *bs*) *cs*) = (append *as* (append *bs* *cs*))

Beweise

1. `(append empty bs)` $=$ *bs*

`(append empty bs)` $=$ *bs*
folgt direkt aus dem Substitutionsmodell.

Beweise

2. (append *as* empty) = *as*

(append *as* empty) = *as*
Listeninduktion über *as*.

Induktionsbasis: *as* = empty
(append empty empty) = empty
folgt aus dem Substitutionsmodell.

Induktionsschritt: *as* = (cons *a* *as*′)

$$
\begin{array}{ll}
 & \texttt{(append (cons } a \; as'\texttt{) empty)} \\
\overset{\text{Sub.}}{=} & \texttt{(cons (first (cons } a \; as'\texttt{))} \\
 & \qquad \texttt{(append (rest (cons } a \; as'\texttt{)) empty))} \\
\overset{\text{Sub.}}{=} & \texttt{(cons } a \; \texttt{(append } as' \; \texttt{empty))} \\
\overset{\text{IH}}{=} & \texttt{(cons } a \; as'\texttt{)}
\end{array}
$$

Beweise

3. `(append (append as bs) cs) = (append as (append bs cs))`

Listeninduktion über *as*.

Induktionsbasis: *as* = `empty`

$$
\begin{aligned}
& \texttt{(append (append empty } bs\texttt{) } cs\texttt{)} \\
\overset{2}{=} \; & \texttt{(append } bs \; cs\texttt{)} \\
\overset{2}{=} \; & \texttt{(append empty (append } bs \; cs\texttt{))}
\end{aligned}
$$

Beweise

3. (append (append *as* *bs*) *cs*) = (append *as* (append *bs* *cs*))

Listeninduktion über *as*.

Induktionsschritt: *as* = (cons *a* *as*′)

$$
\begin{aligned}
& \texttt{(append (append (cons } a \texttt{ } as'\texttt{) } bs\texttt{) } cs\texttt{)} \\
\overset{\text{Sub.}}{=} \quad & \texttt{(append (cons } a \texttt{ (append } as' \texttt{ } bs\texttt{)) } cs\texttt{)} \\
\overset{\text{Sub.}}{=} \quad & \texttt{(cons } a \texttt{ } \underline{\texttt{(append (append } as' \texttt{ } bs\texttt{) } cs\texttt{)}}\texttt{)} \\
\overset{\text{IH}}{=} \quad & \texttt{(cons } a \texttt{ (append } as' \texttt{ (append } bs \texttt{ } cs\texttt{)))} \\
\overset{\text{Sub.}^{-1}}{=} \quad & \texttt{(append (cons } a \texttt{ } as'\texttt{) (append } bs \texttt{ } cs\texttt{))} \\
= \quad & \texttt{(append } as \texttt{ (append } bs \texttt{ } cs\texttt{))}
\end{aligned}
$$

Zusammenfassung

- Nachweis, was ein Programm auf den natürlichen Zahlen ausrechnet
- Nachweis der Terminierung
- Rekursion einmal nicht über den direkten Vorgänger einer Zahl
- Listen und Beweise für Prozeduren, die darauf rechnen