Rekursions- und Lerntheorie

WiSe 2010/11; Univ. Trier

Henning Fernau
Universität Trier
fernau@uni-trier.de

Rekursions- und Lerntheorie Gesamtübersicht

1. Einführung: Grundsätzliche Betrachtungen
2. Elementare Berechenbarkeitstheorie
3. Ausblicke auf weitere Ergebnisse der Rekursionstheorie
4. Lerntheorie: Modelle und Aussagen

Organisatorisches

Vorlesung Montag 12 bis 14 im HZ 201
Vorschlag: Nach diesem Termin findet die Veranstaltung von 12.25-13.55 statt; das gestattet eine längere Mittagspause

Übungen (von Stefan Gulan) Freitag 12 bis 14 im F 59 (bei der Mensa)
Vorschlag: von 12.25-13.55 statt; das gestattet eine längere Mittagspause

Ausnahmen bei den nächsten Malen. . .
1. In dieser Woche werde ich auch am Übungstermin eine Vorlesung halten.
2. In der nächsten Woche bin ich bis auf Freitag nicht da. Montag ist aber sowieso Feiertag. Deshalb werde ich dann am Übungstermin eine Vorlesung halten.
3. In der dritten Woche bin ich gar nicht da. Daher wird Stefan dann am VL- und am Übungstermin eine Übung halten.

Im Folgenden:
Anderer Zugang zu Berechenbarkeit von $f : \mathbb{N}^k \to \mathbb{N}$ für $k \in \mathbb{N}$

Dabei auch $k = 0$ erlaubt! $\Rightarrow$ Nullstellige Funktion, Konstante

Beachte Unterschied z.B bei

$$f : \mathbb{N}^0 \to \mathbb{N} \quad \text{mit} \quad f() = 42$$
$$g : \mathbb{N}^2 \to \mathbb{N} \quad \text{mit} \quad (\forall x, y)\ g(x, y) = 42$$

In Programmiersprachen:
`int f()` und `int g(int,int)` haben verschiedenen Typ!

Grundstock sehr einfacher Funktionen:

- $Z : \mathbb{N}^0 \to \mathbb{N}$, die nullstellige Funktion (=Konstante) mit $Z() = 0$
- $S : \mathbb{N}^1 \to \mathbb{N}$, die einstellige Nachfolgerfunktion mit $S(n) = n + 1$
- $pr^k_j : \mathbb{N}^k \to \mathbb{N}$, die k-stellige Projektion auf die j-te Komponente, also $pr^k_j(x_1, ..., x_k) = x_j$, definiert für $k \geq 1$ und $1 \leq j \leq k$.

Zwei Schemata zum Aufbau komplexerer Funktionen mittels Rekursion

- Das *Kompositionsschema* $\mathrm{Komp} : (g_1, ..., g_m, h) \mapsto f$ erzeugt aus m k-stelligen Funktionen g_1,...,g_m und einer m-stelligen Funktion h eine k-stellige Funktion f mit

$$f(x_1, ..., x_k) := h(g_1(x_1, ..., x_k), ..., g_m(x_1, ..., x_k))$$

  Dabei ist $m \geq 1$ und $k \geq 0$.

- Das *Rekursionsschema* $\mathrm{PrRek} : (g, h) \mapsto f$ der *primitiven Rekursion* erzeugt aus einer k-stelligen Verankerungsfunktion g, $k \geq 0$, und einer $(k+2)$-stelligen Funktion h eine neue (rekursiv definierte) $(k+1)$-stellige Funktion f mit

$$\begin{aligned} f(x_1, ..., x_k, 0) &:= g(x_1, ..., x_k) \\ f(x_1, ..., x_k, y+1) &:= h(x_1, ..., x_k, y, f(x_1, ..., x_k, y)) \end{aligned}$$

Primitiv rekursive Funktionen

1. Alle Grundfunktionen sind primitiv rekursiv.
2. Sind m k-stellige Funktionen g_1,...,g_m primitiv rekursiv
   und eine m-stellige Funktion h ebenfalls primitiv rekursiv,
   so ist auch die im Kompositionsschema definierte k-stellige Funktion
   $f = \mathrm{Komp}(g_1, ..., g_m, h)$ primitiv rekursiv.
3. Sind eine k-stellige Funktion g und
   eine $(k+2)$-stellige Funktion h primitiv rekursiv,
   so ist auch die im Rekursionsschema definierte $(k+1)$-stellige Funktion
   $f = \mathrm{PrRek}(g, h)$ primitiv rekursiv.
4. Weitere primitiv rekursive Funktionen gibt es nicht.

Beispiele (a)

Die nullstellige Funktion Z mit Wert 0 ist eine Grundfunktion.

Betrachte $c_1^{(0)} := \mathrm{Komp}(Z, S)$: Stelligkeiten passen zusammen,
Resultat $c_1^{(0)}$ ist nullstellig, mit $c_1^{(0)}() = 1$
d.h. $c_1^{(0)}$ ist nullstellige Konstante mit Wert 1

Analog: nullstellige Konstante $c_m^{(0)}$ mit Wert m durch

$$c_m^{(0)} = \underbrace{\mathrm{Komp}(\mathrm{Komp}(\ldots \mathrm{Komp}}_{m}(Z, S), \ldots, S), S)$$

Formal: Induktionsbeweis hierfür

Beispiele (b)

Betrachte $c_m^{(1)} := \mathrm{PrRek}(c_m^{(0)}, pr_2^2)$.
Stelligkeiten passen wieder zusammen,
Resultat $c_m^{(1)}(0) = c_m^{(0)}() = m$, $c_m^{(1)}(n+1) = c_m^{(1)}(n) = \ldots = m$

In der “Schemaschreibweise”:
$c_m^{(1)}(0) := c_m^{(0)}()$ (Verankerung)
$c_m^{(1)}(y+1) := pr_2^2(y, c_m^{(1)}(y))$ (eigentl. Rekursion)

d.h.: $c_m^{(1)}$ ist einstellige Konstante mit Wert m.

Beispiele

(c) Betrachte $c_m^{(k)} := \mathrm{Komp}(pr_1^k, c_m^{(1)})$:
k-stellige konstante Funktion mit Wert m

(d) Identität $id : \mathbb{N} \to \mathbb{N}$ ist primitiv rekursiv:
$id = pr_1^1$, d.h. id ist Grundfunktion.

(e) Rekursive Definition der Addition:

$$\begin{aligned} add(x,0) &= x = id(x) \\ add(x,y+1) &= S(add(x,y)) \\ &= S(pr_3^3(x,y,add(x,y))) \end{aligned}$$

Damit $add = \mathrm{PrRek}(id, \mathrm{Komp}(pr_3^3, S))$.

Beispiele

(f) Rekursive Definition der Multiplikation:

$$\begin{aligned} \mathrm{mult}(x,0) &= 0 \\ \mathrm{mult}(x,y+1) &= \mathrm{add}(x,\mathrm{mult}(x,y)) \\ &= \mathrm{add}(pr^3_1(x,y,\mathrm{mult}(x,y)),pr^3_3(x,y,\mathrm{mult}(x,y))) \end{aligned}$$

Damit $\mathrm{mult} = \mathrm{PrRek}(c^{(1)}_0, \mathrm{Komp}(pr^3_1, pr^3_3, \mathrm{add}))$.

(g) Vorgängerfunktion $V : \mathbb{N} \to \mathbb{N}$ mit

$$\begin{aligned} V(0) &= 0 \\ V(n+1) &= n \end{aligned}$$

Damit $V = \mathrm{PrRek}(Z, pr^2_1)$.

Beispiele

(h) modifizierte Subtraktion $\mathrm{sub} : \mathbb{N} \to \mathbb{N}$ mit

$$\begin{aligned} \mathrm{sub}(x,y) &= \begin{cases} x-y, & \text{falls } x \geq y \\ 0, & \text{falls } x < y \end{cases} \\ \text{d.h.} \quad \mathrm{sub}(x,0) &= x \\ \mathrm{sub}(x,y+1) &= V(\mathrm{sub}(x,y)) \end{aligned}$$

Damit

$$\mathrm{sub} = \mathrm{PrRek}(\mathrm{id}, \mathrm{Komp}(\mathrm{pr}^3_3, V))$$

(i) Vorzeichenfunktion $\mathrm{sg} : \mathbb{N} \to \mathbb{N}$ mit

$$\begin{aligned} \mathrm{sg}(0) &= 0 \\ \mathrm{sg}(n+1) &= 1 \end{aligned}$$

Damit $\mathrm{sg} = \mathrm{PrRek}(Z, c^{(2)}_1)$.

Abgeleitetes Erzeugungsschema: beschränkte Nullstellensuche

- Gegeben totale Funktion $f : \mathbb{N}^{k+1} \to \mathbb{N}$
- Definiere totale Funktion $g : \mathbb{N}^{k+1} \to \mathbb{N}$ durch

$$g(x_1, ..., x_k, x_{k+1}) = \min(\{x_{k+1}\} \cup \{n \in \mathbb{N} \mid n < x_{k+1} \text{ und } f(n, x_1, ..., x_k) = 0\})$$

Dann gilt

$$\begin{aligned} g(x_1, ..., x_k, 0) &= 0 \\ g(x_1, ..., x_k, y+1) &= g(x_1, ..., x_k, y) \\ &\quad + sg(f(g(x_1, ..., x_k, y), x_1, ..., x_k)) \end{aligned}$$

Warum? Betrachte 3 Fälle: 1. $y' < y$ ist Nst., 2. y ist kleinste Nst., 3. keine Nst. in $0..y$.
Daher gilt: wenn f primitiv rekursiv ist, ist auch g primitiv rekursiv.

Beispiel zur Anwendung der beschränkten Minimalisierung:

- Sei $f : \mathbb{N}^2 \to \mathbb{N}$ definiert durch

$$f(n, x) := \mathrm{sub}(x, \mathrm{mult}(n, n)) \quad = x - n^2$$

- Sei g die nach dem eben beschriebenen Verfahren zu f definierte primitiv rekursive Funktion, d.h.

$$g(x, y) = \min(\ \{y\} \cup \{n \in \mathbb{N} \mid n < y \text{ und } n^2 \geq x\}\ )$$

- Sei $h : \mathbb{N} \to \mathbb{N}$ definiert durch $h(x) := g(x, x+1)$.

  Dann ist auch h primitiv rekursiv, und es gilt:

$$h(x) = \min\{n \in \mathbb{N} \mid n^2 \geq x\} = \lceil \sqrt{x} \rceil$$

Verschlüsselung von Zahlenvektoren durch eine Zahl

Betrachte

$$\begin{aligned} & & c_2(n) &:= \frac{n \cdot (n+1)}{2} = \sum_{i=0}^{n} i \\ \text{also} & & c_2(0) &= 0 \\ & & c_2(n+1) &= c_2(n) + n + 1 = S(\mathrm{add}(n, c_2(n))) \end{aligned}$$

Damit

$$c_2 = \mathrm{PrRek}(Z, \mathrm{Komp}(\mathrm{add}, S))$$

Daraus primitiv rekursive *Cantorsche Bijektion* c:

$$c(x, y) := c_2(x + y) + x$$

Die Cantorsche Bijektion

$c(x,y)$	0	1	2	3	4	...
0	0	1	3	6	10	...
1	2	4	7	11	16	...
2	5	8	12	17	23	...
3	9	13	18	24	31	...
4	14	19	25	32	40	...
⋮	⋮	⋮	⋮	⋮	⋮	

- Werte der Nebendiagonalen von rechts oben nach links unten:
  Konstante Summe $x + y$
- Offensichtlich ist c bijektiv!
  Während dann der zweite Summand weiterzählt, läuft man die Nebendiagonale abwärts.

Die Cantorsche Bijektion (Forts.)

Umkehrung der Funktion c:

- Sei z gegeben
- Gesucht x und y mit $z = c(x, y)$
- Bestimme n mit $c_2(n) \leq z < c_2(n+1)$
- Dann $x = z - c_2(n)$ und $y = n - x$.

Beispiel: $z = c(x, y) = 18 \Rightarrow n = 5, x = 3, y = 2$

Die Cantorsche Bijektion (Forts.)

Betrachte Komponenten der Umkehrung c^{-1} von c, d.h.

$$p_1 := pr_1^2 \circ c^{-1} \text{ und } p_2 := pr_2^2 \circ c^{-1}$$

D.h. $p_1(c(x,y)) = x$, $p_2(c(x,y)) = y$ und $c(p_1(z), p_2(z)) = z$.

p_1 und p_2 sind primitiv rekursiv:

- Zu z bestimme kleinstes n mit $z < c_2(n+1)$:

  Dazu beschränkte Nullstellensuche mit

$$\begin{array}{rcl} f(n,z) & := & sub(1, sub(c_2(n+1), z)) = 1 - (c_2(n+1) - z) \\ g(z,t) & := & \min(\{t\} \cup \{n \in \mathbb{N} \mid n < t \text{ und } f(n,z) = 0\}) \\ & = & \min(\{t\} \cup \{n \in \mathbb{N} \mid n < t \text{ und } z < c_2(n+1)\}) \end{array}$$

  gesuchte Zahl n ist nicht größer als z, also $n := g(z,z)$.

- Dann

$$\begin{array}{rcl} x & = & p_1(z) = z - c_2(g(z,z)) \\ y & = & p_2(z) = n - x = g(z,z) - (z - c_2(g(z,z))) \end{array}$$

Damit sowohl p_1 als auch p_2 primitiv rekursiv.

Die Cantorsche Bijektion (Forts.)

Definiere primitiv rekursive Bijektion $< \cdot > \; : \; \mathbb{N}^k \to \mathbb{N}$ wie folgt:

$$<n_1, n_2, ..., n_k> := c(n_1, c(n_2, ..., c(n_{k-1}, n_k)...))$$

Dabei sei $k \geq 1$.

Betrachte Komponenten $d_i^{(k)}$ der Umkehrfunktion, d.h.

$$d_i^{(k)}(<n_1, n_2, ..., n_k>) = n_i$$

Die Cantorsche Bijektion (Forts.)

Alle $d_i^{(k)}$ sind ebenfalls primitiv rekursiv:

$$\begin{aligned}
d_1^{(k)}(n) &= p_1(n) \\
d_2^{(k)}(n) &= p_1(p_2(n)) \\
&\ldots \\
d_{k-1}^{(k)}(n) &= p_1(p_2(p_2(\ldots p_2(n)\ldots)) \\
d_k^{(k)}(n) &= p_2(p_2(\ldots p_2(n)\ldots))
\end{aligned}$$

Zentraler Satz dieser Vorlesung

Satz: Eine Funktion ist primitiv rekursiv gdw. sie ist LOOP-berechenbar.

Beweis '$\Rightarrow$'

Durch Induktion über den Aufbau der primitiv rekursiven Funktionen:

(a) Grundfunktionen sind LOOP-berechenbar:

Funktion	LOOP-Programm
Z	$x_0 := x_0 + 0$
S	$x_0 := x_1 + 1$
pr_j^k	$x_0 := x_j + 0$

Zentraler Satz dieser Vorlesung (Forts. Beweis '$\Rightarrow$')

(b) Kompositionsschema $f = h(g_1, ..., g_m)$ mit k-stelligen g_i:
Setze Programme H, G_1,..., G_m für h, g_1,..., g_m
zu Programm F für f zusammen durch:
ν sei maximaler Index der Variablen in H, G_1,..., G_m
Dann arbeite F wie folgt:

- Sicherung der Startwerte $n_1, ..., n_k$ von $x_1, ..., x_k$ in $x_{\nu+1}, ..., x_{\nu+k}$
- Ausführung von G_1, d.h. Berechnung von $j_1 = g_1(n_1, ..., n_k)$
- Sicherung des Resultates j_1 durch $x_{\nu+k+1} := x_0$
- Alle Variablen mit Index $\leq \nu$ wieder auf 0 setzen
- Werte n_i für $1 \leq i \leq k$ wieder in x_i speichern
- Ausführung von G_2, d.h. Berechnung von $j_2 = g_2(n_1, ..., n_k)$
- usw...
- Am Ende: Berechnung von $h(j_1, ..., j_m)$

Zentraler Satz dieser Vorlesung (Forts. Beweis '$\Rightarrow$')

(c) Rekursionsschema $f = \mathrm{PrRek}(g, h)$, d.h.

$$\begin{aligned} f(x_1, ..., x_k, 0) &:= g(x_1, ..., x_k) \\ f(x_1, ..., x_k, y+1) &:= h(x_1, ..., x_k, y, f(x_1, ..., x_k, y)) \end{aligned}$$

Gegeben seien also LOOP-berechenbare g und h

Arbeitsweise eines LOOP-Programms für f:

$x_0 := g(x_1, ..., x_k)$;
$t := 0$;
LOOP x_{k+1} DO $x_0 := h(x_1, ..., x_k, t, x_0); t := t + 1$ END;

Zentraler Satz dieser Vorlesung (Forts. Beweis '$\Rightarrow$')

Umsetzung in LOOP-Programm (analog zur Komposition):

- Werte $x_1, ..., x_k$ werden gesichert (in geeigneten Variablen)
- Variablen, die bei g oder h genutzt werden, werden wieder gelöscht
- Damit: Keine 'Seiteneffekte' bei den Berechnungen von g und h
- t ist Variable, die ansonsten ungenutzt ist

LOOP-Programm berechnet offenbar f !

Zentraler Satz dieser Vorlesung (Beweis '$\Leftarrow$')

- P sei LOOP-Programm für r-stellige Funktion f.

- Wähle $k \geq r$ so, dass P nur Variablen x_i mit $i \leq k$ nutzt.

Nun Induktion über den Aufbau von LOOP-berechenbaren Funktionen mit Variablen aus $\{x_0, ..., x_k\}$:

Zentraler Satz dieser Vorlesung (Forts. Beweis '$\Leftarrow$')

- Kodiere x_0,x_1,..., x_k in einer einzelnen Zahl
  mit Bijektion $< \cdot > : \mathbb{N}^{k+1} \to \mathbb{N}$ und Umkehrungen $d_i^{(k+1)}$
- Konstruiere (induktiv) primitiv rekursive Funktion $g_P : \mathbb{N} \to \mathbb{N}$,
  die das Verhalten von P auf allen Variablen x_0, x_1,..., x_k simuliert
- d.h. zu gegebenen Anfangswerten a_0, a_1,..., a_k und Endwerten b_0, b_1,..., b_k (nach Ablauf von P) der Variablen x_0, x_1,..., x_k gelte

$$g_P(<a_0, a_1, ..., a_k>) = <b_0, b_1, ..., b_k> \quad (*)$$

Zentraler Satz dieser Vorlesung (Beweis '$\Leftarrow$' Ind.schritt)

(a) Falls P die Form $x_i := x_j \pm c$ hat, so setze

$$\begin{aligned} g_P(z) \;:= <\; & d_1^{(k+1)}(z), \ldots, d_i^{(k+1)}(z), \\ & d_{j+1}^{(k+1)}(z) \pm c, \\ & d_{i+2}^{(k+1)}(z), \ldots, d_{k+1}^{(k+1)}(z) \;> \end{aligned}$$

(Achtung: Variable x_i entspricht $i{+}1$-ter Komponente!)

Damit ist g_P primitiv rekursiv und hat Eigenschaft $(*)$.

Zentraler Satz dieser Vorlesung (Beweis '$\Leftarrow$' Ind.schritt)

(b) Falls P die Form $\mathrm{Q}; \mathrm{R}$ hat:

- Programme Q und R sind kürzer als P
- mit Induktionsannahme:
  es gibt primitiv rekursive Funktionen g_{Q} für Q und g_{R} für R
- Definiere g_{P} durch $g_{\mathrm{P}}(z) = g_{\mathrm{R}}(g_{\mathrm{Q}}(z))$.

Damit ist g_{P} primitiv rekursiv und hat Eigenschaft $(*)$.

Zentraler Satz dieser Vorlesung (Beweis '$\Leftarrow$' Ind.schritt)

(c) Falls P die Form `LOOP` x_i `DO` Q `END` hat:

- mit Induktionsannahme:
  es gibt primitiv rekursive Funktion g_Q für Q mit $(*)$.
- Definiere mit primitiver Rekursion zweistellige Funktion h:

$$h(0,x) = x; \qquad h(n+1,x) = g_Q(h(n,x))$$

- $h(n,x)$ simuliert in x (für $<x_0, x_1, ..., x_k>$) n Anwendungen von Q.
- Setze $g_P(x) := h(d_{i+1}^{(k+1)}(x), x)$.

Damit ist wieder g_P primitiv rekursiv mit Eigenschaft $(*)$.

Zentraler Satz dieser Vorlesung (Beweis '$\Leftarrow$' Fazit)

Also: Für alle LOOP-Programme P existiert primitiv rekursives g_P mit

$$g_P(<a_0, a_1, ..., a_k>) = <b_0, b_1, ..., b_k>$$

Wird $f : \mathbb{N}^r \to \mathbb{N}$ durch P berechnet, so ergibt sich

$$f(n_1, ..., n_r) = d_1^{(k+1)}(g_P(<0, n_1, ..., n_r, \underbrace{0, ..., 0}_{k-r}>)),$$

d.h. f ist auch primitiv rekursiv.

Abschließende Hinweise

— “Rekursion” zentrales Konzept in der Informatik.
— Daher ist “Induktion” das zentrale Beweisprinzip in der Informatik.
— im Titel der Vorlesung: “Rekursionstheorie”.

— Als Programmiersprachenparadigma gestattet / erzwingt es “seiteneffektfreies Programmieren”.

— Über charakteristische Funktionen lassen sich auch “primitv rekursive Mengen” einführen, und über die “Äquivalenz” zwischen Zahlen und Zeichenketten auch “primitiv rekursive Sprachen”.
— Wir werden beim nächsten Mal sehen: primitive Rekursion ist “nicht alles”.