Vorlesung

Theoretische Informatik II

Bernhard Beckert

Institut für Informatik

UNIVERSITÄT KOBLENZ · LANDAU

Wintersemester 2007/2008

Dank

Diese Vorlesungsmaterialien basieren zum Teil auf den Folien zu den Vorlesungen von

Katrin Erk (gehalten an der Universität Koblenz-Landau)

Jürgen Dix (gehalten an der TU Clausthal)

Christoph Kreitz (gehalten an der Universität Potsdam)

Ihnen gilt mein herzlicher Dank.

– Bernhard Beckert, Oktober 2007

Teil III

Rekursive Funktionen

Rekursive Funktionen

Motivation

- Funktionen als Berechnungsmodell
- Ohne darunterliegendes Maschinenmodell

Idee

- Einfache (atomare) Funktionen sind berechenbar
- Kombinationen berechenbarer Funktionen sind berechenbar
- Wir betrachten Funktionen $f : \mathbb{N}^k \rightarrow \mathbb{N}$ $(k \geq 0)$

Fragen

- Welches sind die atomaren Funktionen?
- Welche Art der Kombination ist zulässig?

Atomare Funktionen

Folgende Funktionen sind **primitiv rekursiv** und **μ-rekursiv**:

Konstante Null

$$0 : \mathbb{N}^0 \to \mathbb{N} \quad \text{mit} \quad 0() = 0$$

Nachfolger (Successor)

$$+1 : \mathbb{N}^1 \to \mathbb{N} \quad \text{mit} \quad +1(n) = n+1$$

Projektion (Auswahl)

$$\pi_i^k : \mathbb{N}^k \to \mathbb{N} \quad \text{mit} \quad \pi_i^k(n_1, \ldots, n_i, \ldots, n_k) = n_i \qquad (1 \leq i \leq k)$$

Schreibweise bei Argumententupeln

$\boldsymbol{n}$ für $n_1, \ldots, n_k \qquad (k \geq 0)$

Rekursive Funktionen

Komposition (simultanes Einsetzen)

Sind

$$g : \mathbb{N}^r \to \mathbb{N} \qquad (r \geq 1)$$

$$h_1 : \mathbb{N}^k \to \mathbb{N}, \ \ldots, \ h_r : \mathbb{N}^k \to \mathbb{N} \qquad (k \geq 0)$$

primitiv rekursiv bzw. **μ-rekursiv**,
dann ist auch

$$f : \mathbb{N}^k \to \mathbb{N} \quad \text{mit} \quad f(\boldsymbol{n}) = g(h_1(\boldsymbol{n}), \ldots, h_r(\boldsymbol{n}))$$

primitiv rekursiv bzw. **μ-rekursiv**

Schreibweise ohne Argumente:

$$f \ = \ g \circ (h_1, \ldots, h_r)$$

Teil III

Rekursive Funktionen

Primitiv rekursive Funktionen

Primitive Rekursion

Sind

$$g : \mathbb{N}^k \to \mathbb{N} \qquad (k \geq 0)$$
$$h : \mathbb{N}^{k+2} \to \mathbb{N}$$

primitiv rekursiv,
dann ist auch

$$f : \mathbb{N}^{k+1} \to \mathbb{N} \qquad \text{mit} \qquad \begin{aligned} f(\boldsymbol{n}, 0) &= g(\boldsymbol{n}) \\ f(\boldsymbol{n}, m+1) &= h(\boldsymbol{n}, m, f(\boldsymbol{n}, m)) \end{aligned}$$

primitiv rekursiv

Schreibweise ohne Argumente:

$$f = \mathcal{PR}[g, h]$$

Primitiv rekursive Funktionen

Definition 10.1 (Primitiv rekursive Funktionen)

Atomare Funktionen: Die konstanten Funktionen

- Null 0
- Nachfolger $+1$
- Projektion π_i^k $(1 \leq i \leq k)$

sind primitiv rekursiv

Komposition: Die durch Komposition aus primitiv rekursiven Funktionen gebildeten Funktionen sind primitiv rekursiv

Primitive Rekursion: Die durch primitive Rekursion aus primitiv rekursiven Funktionen gebildeten Funktionen sind primitiv rekursiv

Notation

$\wp =$ Menge der primitiv rekursiven Funktionen

Definition arithmetischer Funktionen

$f(n) = n + c$ für $c \in \mathbb{N}, c > 0$

$$f = \underbrace{+1 \circ \cdots \circ +1}_{c \text{ mal}}$$

Identität

$$f = \pi_1^1$$

$f(n, m) = n + m$

$$f = \mathcal{PR}[\, \pi_1^1,\ +1 \circ \pi_3^3 \,]$$

Definition arithmetischer Funktionen

$f(n) = n - 1$

$$f = \mathcal{PR}[\, 0,\, \pi_1^2 \,]$$

$f(n,m) = n - m$

$$f = \mathcal{PR}[\, \pi_1^1,\, -1 \circ \pi_3^3 \,]$$

$f(n,m) = n * m$

$$f = \mathcal{PR}[\, 0,\, + \circ (\pi_1^3, \pi_3^3) \,]$$

Umordnen, Weglassen, Verdoppeln von Argumenten

Lemma

Die Menge der primitiv rekursiven Funktionen ist abgeschlossen unter

- Umordnen
- Weglassen
- Verdoppeln

von Argumenten bei der Komposition von Funktionen

Beweis (Skizze)

Ein Argumenttupel $\boldsymbol{n}'$, das aus $\boldsymbol{n}$ durch Umordnung, Weglassen und Verdoppeln entsteht, kann dargestellt werden als:

$$\boldsymbol{n}' = (\pi^k_{i_1}(\boldsymbol{n}), \ldots, \pi^k_{i_r}(\boldsymbol{n}))$$

Fallunterscheidung

Lemma (Fallunterscheidung ist primitiv rekursiv)

- Sind g_i, h_i $(1 \leq i \leq r)$ primitiv rekursive Funktionen,
- und gibt es für jedes $\boldsymbol{n}$ genau ein i mit $h_i(\boldsymbol{n}) = 0$,

dann ist

$$f(\boldsymbol{n}) = \begin{cases} g_1(\boldsymbol{n}) & \text{falls } h_1(\boldsymbol{n}) = 0 \\ \vdots & \vdots \\ g_r(\boldsymbol{n}) & \text{falls } h_r(\boldsymbol{n}) = 0 \end{cases}$$

primitiv rekursiv.

Beweis

$$f(\boldsymbol{n}) = g_1(\boldsymbol{n}) * (1 - h_1(\boldsymbol{n})) + \cdots + g_r(\boldsymbol{n}) * (1 - h_r(\boldsymbol{n}))$$

Teil III

Rekursive Funktionen

Gödelisierung

Gödelisierung ist primitiv rekursiv

- Die Kodierung von Zahlenfolgen als einzelne Zahl,
- entsprechende Dekodierungsfunktionen (Projektionsfunktionen)

sind **primitiv rekursiv**

Genauer

Es gibt primitiv rekursive Funktionen

$$K^r : \mathbb{N}^r \to \mathbb{N} \qquad (r \geq 1)$$
$$D_i : \mathbb{N}^1 \to \mathbb{N} \qquad (1 \leq i \leq r)$$

mit

$$D_i(K^r(n_1, \ldots, n_r) = n_i$$

Gödelisierung

Notation

$$K^r(n_1, \ldots, n_r) = \langle n_1, \ldots, n_r \rangle$$
$$D_i(n) = (n)_i$$

Gödelisierung

Simultane Rekursion

Ist

$$
\begin{aligned}
f_1(\boldsymbol{n},0) &= g_1(\boldsymbol{n}) \\
&\vdots \\
f_r(\boldsymbol{n},0) &= g_r(\boldsymbol{n}) \\
f_1(\boldsymbol{n},m+1) &= h_1(\boldsymbol{n},m,f_1(\boldsymbol{n},m),\ldots,f_r(\boldsymbol{n},m)) \\
&\vdots \\
f_r(\boldsymbol{n},m+1) &= h_r(\boldsymbol{n},m,f_1(\boldsymbol{n},m),\ldots,f_r(\boldsymbol{n},m))
\end{aligned}
$$

und sind $g_1,\ldots,g_r$, $h_1,\ldots,h_r$ primitiv rekursiv,

Dann sind $f_1,\ldots,f_r$ **primitiv rekursiv**

Beweisidee: Gödelisierung

$\wp = \mathbf{LOOP}$

Theorem 11.1 ($\wp = \mathbf{LOOP}$)

Die Menge der LOOP-berechenbaren Funktionen ist gleich der Menge der primitiv rekursiven Funktionen.

Beweisskizze

1. $\wp \subseteq \mathbf{LOOP}$

1.a. Die atomaren Funktionen sind LOOP-berechenbar

0: $x_1 := x_1 - 1$ *// NOP*

$+1$: $x_2 := x_1 + 1$

π_i^k: $x_{k+1} := x_i$

Beweisskizze (Forts.)

1.b. LOOP ist abgeschlossen unter Komposition von Funktionen

Gegeben

$$f : \mathbb{N}^k \to \mathbb{N} \qquad \text{mit} \qquad f(\boldsymbol{n}) = h(g_1(\boldsymbol{n}), \ldots, g_r(\boldsymbol{n}))$$

– P_h berechnet h

– $P_{g,i}$ berechnet $g_i \quad (1 \leq i \leq r)$

Dann wird f berechnet von dem LOOP-Programm

$$P'_{g,1}; \ldots; P'_{g,r}; \; P'_h$$

Dabei unterscheiden sich P'_h von P_h und $P'_{g,i}$ von $P_{g,i}$ durch geeignete Umbenennung von Registern und Umkopierung von Registerninhalten

Beweisskizze (Forts.)

1.c. LOOP ist abgeschlossen gegen primitive Rekursion

Gegeben

$$f : \mathbb{N}^k \to \mathbb{N} \qquad \text{mit} \qquad \begin{aligned} f(\boldsymbol{n}, 0) &= g(\boldsymbol{n}) \\ f(\boldsymbol{n}, m+1) &= h(\boldsymbol{n}, m, f(\boldsymbol{n}, m)) \end{aligned}$$

P_h berechnet h und P_g berechnet g

Beweisskizze (Forts.)

Dann wird f berechnet von dem LOOP-Programm

$x_{store\_m} := x_{k+1}$; // *Anzahl Durchläufe*
$x_{k+1} := 0$; // *Aktueller Wert von m*
P'_g;
loop x_{store_m} do
  P'_h
  $x_{k+1} := x_{k+1} + 1$;
end;
$x_{store\_m} := 0$

Dabei unterscheiden sich P'_h von P_h und P'_g von P_g durch geeignete Umbenennung von Registern und Umkopierung von Registerninhalten

Beweisskizze

2. LOOP $\subseteq \wp$

Gegeben ein LOOP-Programm P

- das Register $x_1, \ldots, x_l$ benutzt
- m `loop`-Anweisungen hat

Wir konstruieren eine primitiv rekursive Funktion f_P, die P „simuliert“

$$f_p(\langle n_1, \ldots, n_l,\ h_1, \ldots, h_m \rangle) = \langle n'_1, \ldots, n'_l,\ h_1, \ldots, h_m \rangle$$

gdw.

P gestartet mit n_i in x_i terminiert mit n'_i in $x_i \quad (1 \leq i \leq l)$

In h_j ist „gespeichert“, wie oft the j-te Schleife noch laufen muss

Beweisskizze

Die von P LOOP-berechnete Funktion g ist dann primitiv rekursiv, denn

$$g(n_1, \ldots, n_k) = \left(f_P(\langle n_1, \ldots, n_k, 0, 0, 0, \ldots \rangle) \right)_{k+1}$$

Beweisskizze

Konstruktion von f_P

2.a. $P \equiv \quad x_i := x_i + 1$

$$f_P(n) = \langle (n)_1, \ldots, (n)_{i-1}, (n)_i + 1, (n)_{i+1}, \ldots \rangle$$

2.b. $P \equiv \quad P_1; P_2$

$$f_P = f_{P_2} \circ f_{P_1}$$

Beweisskizze

2.c. $P \equiv$ `loop` x_i `do` P_1 `end`

Zur Initialisierung der Schleife (die j-te):

$$f_1(n) = \langle (n)_1, \ldots, (n)_l, (n)_{l+1}, \ldots, (n)_{l+j-1}, (n)_i, (n)_{l+j+1}, \ldots \rangle$$

Schleife laufen lassen:

$$f_2(n) = \begin{cases} n & \text{falls } (n)_{l+j} = 0 \\ f_{P_1}(f_2(\langle \ldots, (n)_{l+j} - 1, \ldots \rangle)) & \text{sonst} \end{cases}$$

Dann:

$$f_P = f_2 \circ f_1$$

Teil III

Rekursive Funktionen

μ-rekursive Funktionen

μ-Operator

Ist

$$g : \mathbb{N}^{k+1} \to \mathbb{N} \qquad (k \geq 0)$$

μ-rekursiv,
dann ist auch $f : \mathbb{N}^k \to \mathbb{N}$ mit

$$f(\boldsymbol{n}) = \mu i(g(\boldsymbol{n}, i) = 0) = \begin{cases} i_0 & \text{falls } g(\boldsymbol{n}, i_0) = 0 \\ & \text{und für } 0 \leq j < i_0\text{:} \\ & \quad g(\boldsymbol{n}, j) \text{ def. u. } g(\boldsymbol{n}, j) \neq 0 \\ \text{undef} & \text{sonst} \end{cases}$$

μ-rekursiv

μ-rekursive Funktionen

μ-Operator

Schreibweise ohne Argumente:

$$f = \mu g$$

μ-rekursive Funktionen

Definition 12.1 (μ-rekursive Funktionen)

Atomare Funktionen: Die konstanten Funktionen

- Null 0
- Nachfolger $+1$
- Projektion π_i^k $(1 \leq i \leq k)$

sind μ-rekursiv

Komposition: Die durch Komposition aus μ-rekursiven Funktionen gebildeten Funktionen sind μ-rekursiv

Primitive Rekursion: Die durch primitive Rekursion aus μ-rekursiven Funktionen gebildeten Funktionen sind μ-rekursiv

μ-Operator: Die durch Anwendung des μ-Operators aus μ-rekursiven Funktionen gebildeten Funktionen sind μ-rekursiv

μ-rekursive Funktionen

Notation

$$F_\mu = \text{Menge der totalen } \mu\text{-rekursiven Funktionen}$$
$$F_\mu^{part} = \text{Menge aller } \mu\text{-rekursiven Funktionen}$$

μ-rekursive Funktionen

Theorem 12.2

$$F_\mu \subseteq \textbf{WHILE} \quad \textit{und} \quad F_\mu^{\textit{part}} \subseteq \textbf{WHILE}^{\textit{part}}$$

μ-rekursive Funktionen

Beweis (Skizze)

Wir haben schon bewiesen:

$$\wp = \textbf{LOOP} \subset \textbf{WHILE}$$

Bei Verwendung des entsprechenden Beweises bleibt zu zeigen, dass der μ-Operator in WHILE nachgebildet werden kann.

$$f(\boldsymbol{n}) = \mu i(g(\boldsymbol{n}, i) = 0)$$

kann berechnet werden durch:

$$\mathtt{x}_i = 0; \quad \mathtt{while}\ g(\boldsymbol{n}, \mathtt{x}_i) \neq 0\ \mathtt{do}\ \mathtt{x}_i := \mathtt{x}_i + 1\ \mathtt{end}$$

(informelle Notation!)

Ackermann-Funktion

Wilhelm Ackermann ⋆ 1896, † 1962

- Mathematiker und Logiker
- Promovierte bei D. Hilbert in Göttingen
  (Koautor von Hiberts Buch
  Grundzüge der Theoretischen Logik)
- Mathematiklehrer in Lüdenscheid

Aus seinem Nachruf:

Oberstudienrat Dr. Ackermann war aber nicht nur ein allseits geschätzter und beliebter Lehrer, sondern auch ein weltbekannter Wissenschaftler . . .

Ackermann-Funktion

Definition 12.3 (Ackermann-Funktion A)

$$A_0(x) = \begin{cases} 1 & \text{falls } x = 0 \\ 2 & \text{falls } x = 1 \\ x+2 & \text{sonst} \end{cases}$$

$$A_{n+1}(0) = 1$$

$$A_{n+1}(x+1) = A_n(A_{n+1}(x))$$

$$A(x) = A_x(x)$$

Ackermann-Funktion

Wachstumsverhalten

$$A_1(x) = 2 * x$$

$$A_2(x) = 2^x$$

$$A_3(x) = \underbrace{2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}}_{x \text{ mal}}$$

Ackermann-Funktion

Wachstumsverhalten

$$A_4(3) = 2^{2^{2^2}} = 65536$$

$$A_4(4) = \underbrace{2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}}_{65536 \text{ mal}}$$

$$A_4(5) = \underbrace{2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}}_{A_4(4) \text{ mal}}$$

Ackermann-Funktion

Theorem 12.4

Die Ackermann-Funktion ist

- *total*
- *μ-rekursiv*
- ***nicht primitiv-rekursiv***

Ackermann-Funktion

Beweis

Die Funktionen A_n sind total, da in jedem Rekursionsschritt eines der Argumente kleiner wird.

A ist TM-berechenbar, weil man den Rekursions-Stack auf dem Band speichern kann.

Wegen $F_\mu = \textbf{WHILE}$ (Beweis später)
ist A auch μ-rekursiv

Bei einer primitiv rekursiven Funktion f ist die zur Berechnung von $f(n)$ notwendige Funktions-Schachtelungstiefe für alle n gleich.

A kann mit konstanter Funktions-Schachtelungstiefe nicht berechnet werden. (Detaillierter Beweis sehr aufwendig)

Teil III

Rekursive Funktionen

Rück- und Ausblick

Aus dem Vorlesungsteil „Registermaschinen“ wissen wir:

- **LOOP** $\subsetneq$ **WHILE** $=$ **GOTO** $\subseteq$ **TM**
- **WHILE**^{part} $=$ **GOTO**^{part} $\subseteq$ **TM**^{part}

In diesem Abschnitt schon gezeigt:

- **LOOP** $= \wp$
- $F_\mu \subseteq$ **WHILE** und $F_\mu^{part} \subseteq$ **WHILE**

Noch zu zeigen:

TM $\subseteq F_\mu$ und **TM**^{part} $\subseteq F_\mu^{part}$

Gödelisierung von Turing-Maschinen

Man kann jeder Turing-Maschine

$$M \quad = \quad (K, \Sigma, \delta, s)$$

eindeutig eine Gödelnummer

$$\langle M \rangle \quad \in \mathbb{N}$$

so zuordnen, dass

- die Kodierungsfunktion
  (Berechnung von $\langle M \rangle$ aus den Bestandteilen von M)
- die Dekodierungsfunktionen
  (Berechnung der Bestandteile von M aus $\langle M \rangle$)

primitiv rekursiv sind.

Gödelisierung von Turing-Maschinen

Gödelisierung der Konfigurationen von Turing-Maschinen

Man kann jeder Konfiguration

$$q, w\underline{a}u$$

einer gegebenen Turing-Maschine M eindeutig eine Gödelnummer

$$\langle C \rangle \in \mathbb{N}$$

so zuordnen, dass

- die Kodierungsfunktion
  (Berechnung von $\langle C \rangle$ aus den Bestandteilen von C)
- die Dekodierungsfunktionen
  (Berechnung der Bestandteile von C aus $\langle C \rangle$)

primitiv rekursiv sind.

Simulationslemma

Lemma 13.1 (Simulationslemma)

Es gibt eine ***primitiv rekursive*** *Funktion*

$$f_U : \mathbb{N}^3 \to \mathbb{N} \ ,$$

so dass für jede Turing-Maschine M gilt:

Sind $C_0, \ldots, C_t$ $(t \geq 0)$ *Konfigurationen von M mit*

$$C_i \vdash_M C_{i+1} \qquad (0 \leq i < t) \ ,$$

dann gilt

$$f_U(\langle M \rangle, \langle C_0 \rangle, t) \ = \ \langle C_t \rangle$$

Simulationslemma

Beweisidee

- Die Kodierungs- und Dekodierungsfunktionen für Turing-Maschinen und Konfigurationen sind primitiv rekursiv.
- Ein einzelner Schritt der Turing-Maschine ist primitiv rekursiv
- Eine vorgegebene Anzahl t von Schritten ist primitiv rekursiv

Also ist f_U primitiv rekursiv.

(Detaillierter, konstruktiver Beweis durch explizite Angabe der Funktionen:
4 Seiten in [Erk, Priese])

TM-berechenbare Funktionen sind μ-rekursiv

Theorem 13.2

Jede TM-berechenbare Funktion ist μ-rekursiv:

$$\mathbf{TM} \subseteq F_\mu \quad \textit{und} \quad \mathbf{TM}^{part} \subseteq F_\mu^{part}$$

Beweisskizze

Sei

$$f : \mathbb{N}^k \to \mathbb{N}$$

eine TM-berechnebare Funktion.

Sei M eine TM, die f berechnet.

TM-berechenbare Funktionen sind μ-rekursiv

Beweisskizze

Dann

$$\begin{aligned} f(n_1,\ldots,n_k) \quad = \quad & (f_U(\ \langle M\rangle, \\ & \quad start, \\ & \quad \mu i((f_U(\langle M\rangle, start, i))_{\text{Zust}} = \langle h\rangle)) \\ &)_{\text{w}} \end{aligned}$$

wobei

$$start = \langle s, \#\underbrace{|\cdots|}_{n_1}\#\ldots\#\underbrace{|\cdots|}_{n_k}\underline{\#}\rangle$$

$\langle h\rangle$ = Gödelisierung des Haltezustands

$(\cdot)_{\text{Zust}}$ = Dekodierung des Zustands in einer Konfiguration

$(\cdot)_{\text{w}}$ = Dekodierung des Wortes links vom Schreib-/Lesekopf

Kleenesche Normalform

Korollar (Kleenesche Normalform)

Für jede μ-rekursive Funktion f gibt es primitiv rekursive Funktionen g, h, so dass

$$f(\boldsymbol{n}) = g(\mu i(h(\boldsymbol{n}) = 0))$$

also

$$f = g \circ \mu h$$

Teil III

Rekursive Funktionen

Zusammenfassung

Klassen berechenbarer Funktionen

- $\mathbf{LOOP} = \wp \subsetneq \mathbf{WHILE} = \mathbf{GOTO} = \mathbf{TM} = F_\mu$
- $\mathbf{WHILE}^{part} = \mathbf{GOTO}^{part} = \mathbf{TM}^{part} = F_\mu^{part}$