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

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

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

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

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

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

μ-rekursiv

μ-rekursive Funktionen

μ-Operator

Schreibweise ohne Argumente:

$$f = \mu g$$