

Vorlesung

Theoretische Informatik II

Bernhard Beckert

Institut für Informatik



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

- 1 Einführung
- 2 Primitiv rekursive Funktionen
- 3 $\wp = \text{LOOP}$
- 4 μ -rekursive Funktionen
- 5 $F_\mu = \text{TM} = \text{WHILE}$
- 6 Zusammenfassung

Teil III

Rekursive Funktionen

- 1 Einführung
- 2 Primitiv rekursive Funktionen
- 3 $\wp = \text{LOOP}$
- 4 μ -rekursive Funktionen
- 5 $F_\mu = \text{TM} = \text{WHILE}$
- 6 Zusammenfassung

Teil III

Rekursive Funktionen

1 Einführung

2 Primitiv rekursive Funktionen

3 $\wp = \text{LOOP}$

4 μ -rekursive Funktionen

5 $F_\mu = \text{TM} = \text{WHILE}$

6 Zusammenfassung

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?

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?

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?

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?

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?

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?

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?

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?

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

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

Nachfolger (Successor)

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

Projektion (Auswahl)

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

Atomare Funktionen

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

Konstante Null

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

Nachfolger (Successor)

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

Projektion (Auswahl)

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

Atomare Funktionen

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

Konstante Null

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

Nachfolger (Successor)

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

Projektion (Auswahl)

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

Schreibweise bei Argumententupeln

n für $n_1, \dots, n_k$ ($k \geq 0$)

Komposition (simultanes Einsetzen)

Sind

$$g : \mathbb{N}^r \rightarrow \mathbb{N} \quad (r \geq 1)$$

$$h_1 : \mathbb{N}^k \rightarrow \mathbb{N}, \dots, h_r : \mathbb{N}^k \rightarrow \mathbb{N} \quad (k \geq 0)$$

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

$$f : \mathbb{N}^k \rightarrow \mathbb{N} \quad \text{mit} \quad f(\mathbf{n}) = g(h_1(\mathbf{n}), \dots, h_r(\mathbf{n}))$$

primitiv rekursiv bzw. **μ -rekursiv**

Schreibweise ohne Argumente:

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

Komposition (simultanes Einsetzen)

Sind

$$g : \mathbb{N}^r \rightarrow \mathbb{N} \quad (r \geq 1)$$

$$h_1 : \mathbb{N}^k \rightarrow \mathbb{N}, \dots, h_r : \mathbb{N}^k \rightarrow \mathbb{N} \quad (k \geq 0)$$

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

$$f : \mathbb{N}^k \rightarrow \mathbb{N} \quad \text{mit} \quad f(\mathbf{n}) = g(h_1(\mathbf{n}), \dots, h_r(\mathbf{n}))$$

primitiv rekursiv bzw. **μ -rekursiv**

Schreibweise ohne Argumente:

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

Teil III

Rekursive Funktionen

1 Einführung

2 Primitiv rekursive Funktionen

3 $\wp = \text{LOOP}$

4 μ -rekursive Funktionen

5 $F_\mu = \text{TM} = \text{WHILE}$

6 Zusammenfassung

Teil III

Rekursive Funktionen

- 1 Einführung
- 2 Primitiv rekursive Funktionen**
- 3 $\wp = \text{LOOP}$
- 4 μ -rekursive Funktionen
- 5 $F_\mu = \text{TM} = \text{WHILE}$
- 6 Zusammenfassung

Primitiv rekursive Funktionen

Primitive Rekursion

Sind

$$g : \mathbb{N}^k \rightarrow \mathbb{N} \quad (k \geq 0)$$

$$h : \mathbb{N}^{k+2} \rightarrow \mathbb{N}$$

primitiv rekursiv,

dann ist auch

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

primitiv rekursiv

Schreibweise ohne Argumente:

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

Primitive Rekursion

Sind

$$g : \mathbb{N}^k \rightarrow \mathbb{N} \quad (k \geq 0)$$

$$h : \mathbb{N}^{k+2} \rightarrow \mathbb{N}$$

primitiv rekursiv,

dann ist auch

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

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

$\mathcal{P}$ = Menge der primitiv rekursiven Funktionen

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

$\mathcal{P}$ = Menge der primitiv rekursiven Funktionen

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

$\mathcal{P}$ = Menge der primitiv rekursiven Funktionen

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

$\mathcal{P}$ = Menge der primitiv rekursiven Funktionen

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

$\mathcal{P}$ = Menge der primitiv rekursiven Funktionen

Definition arithmetischer Funktionen

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

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

Identität

$$f = \pi_1^1$$

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

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

Definition arithmetischer Funktionen

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

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

Identität

$$f = \pi_1^1$$

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

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

Definition arithmetischer Funktionen

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

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

Identität

$$f = \pi_1^1$$

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

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

Definition arithmetischer Funktionen

$$f(n) = n - 1$$

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

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

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

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

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

Definition arithmetischer Funktionen

$$f(n) = n - 1$$

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

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

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

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

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

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

$$\mathbf{n}' = (\pi_{i_1}^k(\mathbf{n}), \dots, \pi_{i_r}^k(\mathbf{n}))$$

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

$$\mathbf{n}' = (\pi_{i_1}^k(\mathbf{n}), \dots, \pi_{i_r}^k(\mathbf{n}))$$

Fallunterscheidung

Lemma (Fallunterscheidung ist primitiv rekursiv)

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

dann ist

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

primitiv rekursiv.

Beweis

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

Fallunterscheidung

Lemma (Fallunterscheidung ist primitiv rekursiv)

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

dann ist

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

primitiv rekursiv.

Beweis

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

Fallunterscheidung

Lemma (Fallunterscheidung ist primitiv rekursiv)

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

dann ist

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

primitiv rekursiv.

Beweis

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

Fallunterscheidung

Lemma (Fallunterscheidung ist primitiv rekursiv)

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

dann ist

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

primitiv rekursiv.

Beweis

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

Teil III

Rekursive Funktionen

- 1 Einführung
- 2 Primitiv rekursive Funktionen**
- 3 $\wp = \text{LOOP}$
- 4 μ -rekursive Funktionen
- 5 $F_\mu = \text{TM} = \text{WHILE}$
- 6 Zusammenfassung

Teil III

Rekursive Funktionen

- 1 Einführung
- 2 Primitiv rekursive Funktionen
- 3 $\wp = \text{LOOP}$**
- 4 μ -rekursive Funktionen
- 5 $F_\mu = \text{TM} = \text{WHILE}$
- 6 Zusammenfassung

Gödelisierung ist primitiv rekursiv

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

sind **primitiv rekursiv**

Genauer

Es gibt primitiv rekursive Funktionen

$$K^r : \mathbb{N}^r \rightarrow \mathbb{N} \quad (r \geq 1)$$

$$D_i : \mathbb{N}^1 \rightarrow \mathbb{N} \quad (1 \leq i \leq r)$$

mit

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

Gödelisierung ist primitiv rekursiv

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

sind **primitiv rekursiv**

Genauer

Es gibt primitiv rekursive Funktionen

$$K^r : \mathbb{N}^r \rightarrow \mathbb{N} \quad (r \geq 1)$$

$$D_i : \mathbb{N}^1 \rightarrow \mathbb{N} \quad (1 \leq i \leq r)$$

mit

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

Gödelisierung ist primitiv rekursiv

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

sind **primitiv rekursiv**

Genauer

Es gibt primitiv rekursive Funktionen

$$K^r : \mathbb{N}^r \rightarrow \mathbb{N} \quad (r \geq 1)$$

$$D_i : \mathbb{N}^1 \rightarrow \mathbb{N} \quad (1 \leq i \leq r)$$

mit

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

Gödelisierung ist primitiv rekursiv

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

sind **primitiv rekursiv**

Genauer

Es gibt primitiv rekursive Funktionen

$$K^r : \mathbb{N}^r \rightarrow \mathbb{N} \quad (r \geq 1)$$

$$D_i : \mathbb{N}^1 \rightarrow \mathbb{N} \quad (1 \leq i \leq r)$$

mit

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

Gödelisierung

Gödelisierung ist primitiv rekursiv

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

sind **primitiv rekursiv**

Genauer

Es gibt primitiv rekursive Funktionen

$$K^r : \mathbb{N}^r \rightarrow \mathbb{N} \quad (r \geq 1)$$

$$D_i : \mathbb{N}^1 \rightarrow \mathbb{N} \quad (1 \leq i \leq r)$$

mit

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

Gödelisierung ist primitiv rekursiv

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

sind **primitiv rekursiv**

Genauer

Es gibt primitiv rekursive Funktionen

$$K^r : \mathbb{N}^r \rightarrow \mathbb{N} \quad (r \geq 1)$$

$$D_i : \mathbb{N}^1 \rightarrow \mathbb{N} \quad (1 \leq i \leq r)$$

mit

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

Notation

$$K^r(n_1, \dots, n_r) = \langle n_1, \dots, n_r \rangle$$

$$D_i(n) = (n)_i$$

Simultane Rekursion

Ist

$$f_1(\mathbf{n}, 0) = g_1(\mathbf{n})$$

$$\vdots$$

$$f_r(\mathbf{n}, 0) = g_r(\mathbf{n})$$

$$f_1(\mathbf{n}, m+1) = h_1(\mathbf{n}, m, f_1(\mathbf{n}, m), \dots, f_r(\mathbf{n}, m))$$

$$\vdots$$

$$f_r(\mathbf{n}, m+1) = h_r(\mathbf{n}, m, f_1(\mathbf{n}, m), \dots, f_r(\mathbf{n}, m))$$

und sind $g_1, \dots, g_r, \quad h_1, \dots, h_r$ primitiv rekursiv,

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

Beweisidee: Gödelisierung

Simultane Rekursion

Ist

$$f_1(\mathbf{n}, 0) = g_1(\mathbf{n})$$

$$\vdots$$

$$f_r(\mathbf{n}, 0) = g_r(\mathbf{n})$$

$$f_1(\mathbf{n}, m+1) = h_1(\mathbf{n}, m, f_1(\mathbf{n}, m), \dots, f_r(\mathbf{n}, m))$$

$$\vdots$$

$$f_r(\mathbf{n}, m+1) = h_r(\mathbf{n}, m, f_1(\mathbf{n}, m), \dots, f_r(\mathbf{n}, m))$$

und sind $g_1, \dots, g_r, \quad h_1, \dots, h_r$ primitiv rekursiv,

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

Beweisidee: Gödelisierung

Simultane Rekursion

Ist

$$f_1(\mathbf{n}, 0) = g_1(\mathbf{n})$$

$$\vdots$$

$$f_r(\mathbf{n}, 0) = g_r(\mathbf{n})$$

$$f_1(\mathbf{n}, m+1) = h_1(\mathbf{n}, m, f_1(\mathbf{n}, m), \dots, f_r(\mathbf{n}, m))$$

$$\vdots$$

$$f_r(\mathbf{n}, m+1) = h_r(\mathbf{n}, m, f_1(\mathbf{n}, m), \dots, f_r(\mathbf{n}, m))$$

und sind $g_1, \dots, g_r, \quad h_1, \dots, h_r$ primitiv rekursiv,

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

Beweisidee: Gödelisierung

Simultane Rekursion

Ist

$$f_1(\mathbf{n}, 0) = g_1(\mathbf{n})$$

$$\vdots$$

$$f_r(\mathbf{n}, 0) = g_r(\mathbf{n})$$

$$f_1(\mathbf{n}, m+1) = h_1(\mathbf{n}, m, f_1(\mathbf{n}, m), \dots, f_r(\mathbf{n}, m))$$

$$\vdots$$

$$f_r(\mathbf{n}, m+1) = h_r(\mathbf{n}, m, f_1(\mathbf{n}, m), \dots, f_r(\mathbf{n}, m))$$

und sind $g_1, \dots, g_r, \quad h_1, \dots, h_r$ primitiv rekursiv,

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

Beweisidee: Gödelisierung

$\wp = \text{LOOP}$

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

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

Beweisskizze

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

1.a. Die atomaren Funktionen sind LOOP-berechenbar

0: $x_1 := x_1 - 1$ // NOP
+1: $x_2 := x_1 + 1$
 π_i^k : $x_{k+1} := x_i$

$\wp = \text{LOOP}$

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

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

Beweisskizze

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

1.a. Die atomaren Funktionen sind LOOP-berechenbar

0: $x_1 := x_1 - 1$ // NOP
+1: $x_2 := x_1 + 1$
 π_i^k : $x_{k+1} := x_i$

$\wp = \text{LOOP}$

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

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

Beweisskizze

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

1.a. Die atomaren Funktionen sind LOOP-berechenbar

0: $x_1 := x_1 - 1$ // NOP
+1: $x_2 := x_1 + 1$
 π_j^k : $x_{k+1} := x_j$

$\wp = \text{LOOP}$

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

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

Beweisskizze

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

1.a. Die atomaren Funktionen sind LOOP-berechenbar

0: $x_1 := x_1 - 1$ // NOP
+1: $x_2 := x_1 + 1$
 π_i^k : $x_{k+1} := x_i$

$\wp = \text{LOOP}$

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

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

Beweisskizze

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

1.a. Die atomaren Funktionen sind LOOP-berechenbar

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

+1: $x_2 := x_1 + 1$

π_j^k : $x_{k+1} := x_j$

$\wp = \text{LOOP}$

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

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

Beweisskizze

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

1.a. Die atomaren Funktionen sind LOOP-berechenbar

0: $x_1 := x_1 - 1$ // NOP
+1: $x_2 := x_1 + 1$
 π_j^k : $x_{k+1} := x_j$

Beweisskizze (Forts.)

1.b. LOOP ist abgeschlossen unter Komposition von Funktionen

Gegeben

$$f : \mathbb{N}^k \rightarrow \mathbb{N} \quad \text{mit} \quad f(\mathbf{n}) = h(g_1(\mathbf{n}), \dots, g_r(\mathbf{n}))$$

- P_h berechnet h
- $P_{g,i}$ berechnet g_i ($1 \leq i \leq r$)

Dann wird f berechnet von dem LOOP-Programm

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

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.b. LOOP ist abgeschlossen unter Komposition von Funktionen

Gegeben

$$f : \mathbb{N}^k \rightarrow \mathbb{N} \quad \text{mit} \quad f(\mathbf{n}) = h(g_1(\mathbf{n}), \dots, g_r(\mathbf{n}))$$

- P_h berechnet h
- $P_{g,i}$ berechnet g_i ($1 \leq i \leq r$)

Dann wird f berechnet von dem LOOP-Programm

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

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.b. LOOP ist abgeschlossen unter Komposition von Funktionen

Gegeben

$$f : \mathbb{N}^k \rightarrow \mathbb{N} \quad \text{mit} \quad f(\mathbf{n}) = h(g_1(\mathbf{n}), \dots, g_r(\mathbf{n}))$$

- P_h berechnet h
- $P_{g,i}$ berechnet g_i ($1 \leq i \leq r$)

Dann wird f berechnet von dem LOOP-Programm

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

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.b. LOOP ist abgeschlossen unter Komposition von Funktionen

Gegeben

$$f : \mathbb{N}^k \rightarrow \mathbb{N} \quad \text{mit} \quad f(\mathbf{n}) = h(g_1(\mathbf{n}), \dots, g_r(\mathbf{n}))$$

- P_h berechnet h
- $P_{g,i}$ berechnet g_i ($1 \leq i \leq r$)

Dann wird f berechnet von dem LOOP-Programm

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

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.b. LOOP ist abgeschlossen unter Komposition von Funktionen

Gegeben

$$f : \mathbb{N}^k \rightarrow \mathbb{N} \quad \text{mit} \quad f(\mathbf{n}) = h(g_1(\mathbf{n}), \dots, g_r(\mathbf{n}))$$

- P_h berechnet h
- $P_{g,i}$ berechnet g_i ($1 \leq i \leq r$)

Dann wird f berechnet von dem LOOP-Programm

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

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

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

P_h berechnet h und P_g berechnet g

Beweisskizze (Forts.)

1.c. LOOP ist abgeschlossen gegen primitive Rekursion

Gegeben

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

P_h berechnet h und P_g berechnet g

Beweisskizze (Forts.)

1.c. LOOP ist abgeschlossen gegen primitive Rekursion

Gegeben

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

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 (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

$\wp = \text{LOOP}$

Beweisskizze

2. $\text{LOOP} \subseteq \wp$

Gegeben ein LOOP-Programm P

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

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

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

gdw.

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

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

Beweisskizze

2. $\text{LOOP} \subseteq \wp$

Gegeben ein LOOP-Programm P

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

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

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

gdw.

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

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

Beweisskizze

2. $\text{LOOP} \subseteq \wp$

Gegeben ein LOOP-Programm P

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

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

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

gdw.

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

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

Beweisskizze

2. $\text{LOOP} \subseteq \wp$

Gegeben ein LOOP-Programm P

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

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

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

gdw.

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

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

Beweisskizze

2. $\text{LOOP} \subseteq \wp$

Gegeben ein LOOP-Programm P

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

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

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

gdw.

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

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

Beweisskizze

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

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

Beweisskizze

Konstruktion von f_P

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

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

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

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

Beweisskizze

Konstruktion von f_P

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

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

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

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

Beweisskizze

Konstruktion von f_P

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

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

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

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

Beweisskizze

Konstruktion von f_P

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

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

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

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

Beweisskizze

Konstruktion von f_P

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

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

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

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

$\wp = \text{LOOP}$

Beweisskizze

2.c. $P \equiv \text{loop } x_j \text{ do } P_1 \text{ end}$

Zur Initialisierung der Schleife (die j -te):

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

Schleife laufen lassen:

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

Dann:

$$f_P = f_2 \circ f_1$$

$\wp = \text{LOOP}$

Beweisskizze

2.c. $P \equiv \text{loop } x_j \text{ do } P_1 \text{ end}$

Zur Initialisierung der Schleife (die j -te):

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

Schleife laufen lassen:

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

Dann:

$$f_P = f_2 \circ f_1$$

$\wp = \text{LOOP}$

Beweisskizze

2.c. $P \equiv \text{loop } x_j \text{ do } P_1 \text{ end}$

Zur Initialisierung der Schleife (die j -te):

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

Schleife laufen lassen:

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

Dann:

$$f_P = f_2 \circ f_1$$

Beweisskizze

2.c. $P \equiv \text{loop } x_j \text{ do } P_1 \text{ end}$

Zur Initialisierung der Schleife (die j -te):

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

Schleife laufen lassen:

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

Dann:

$$f_P = f_2 \circ f_1$$

Teil III

Rekursive Funktionen

- 1 Einführung
- 2 Primitiv rekursive Funktionen
- 3 $\wp = \text{LOOP}$**
- 4 μ -rekursive Funktionen
- 5 $F_\mu = \text{TM} = \text{WHILE}$
- 6 Zusammenfassung

Teil III

Rekursive Funktionen

- 1 Einführung
- 2 Primitiv rekursive Funktionen
- 3 $\wp = \text{LOOP}$
- 4 μ -rekursive Funktionen**
- 5 $F_\mu = \text{TM} = \text{WHILE}$
- 6 Zusammenfassung

μ -Operator

Ist

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

μ -rekursiv,

dann ist auch $f : \mathbb{N}^k \rightarrow \mathbb{N}$ mit

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

μ -rekursiv

μ -Operator

Schreibweise ohne Argumente:

$$f = \mu g$$

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

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

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

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

Notation

F_μ = Menge der totalen μ -rekursiven Funktionen

F_μ^{part} = Menge aller μ -rekursiven Funktionen

Theorem 12.2

$$F_\mu \subseteq \mathbf{WHILE} \quad \text{und} \quad F_\mu^{part} \subseteq \mathbf{WHILE}^{part}$$

Beweis (Skizze)

Wir haben schon bewiesen:

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

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

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

kann berechnet werden durch:

$$x_i = 0; \quad \text{while } g(\mathbf{n}, x_i) \neq 0 \text{ do } x_i := x_i + 1 \text{ end}$$

(informelle Notation!)

Beweis (Skizze)

Wir haben schon bewiesen:

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

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

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

kann berechnet werden durch:

$x_i = 0$; while $g(\mathbf{n}, x_i) \neq 0$ do $x_i := x_i + 1$ end

(informelle Notation!)

Beweis (Skizze)

Wir haben schon bewiesen:

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

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

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

kann berechnet werden durch:

$x_i = 0; \text{ while } g(\mathbf{n}, x_i) \neq 0 \text{ do } x_i := x_i + 1 \text{ end}$

(informelle Notation!)

Beweis (Skizze)

Wir haben schon bewiesen:

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

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

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

kann berechnet werden durch:

$$x_i = 0; \quad \text{while } g(\mathbf{n}, x_i) \neq 0 \text{ do } x_i := x_i + 1 \text{ end}$$

(informelle Notation!)

Ackermann-Funktion

Wilhelm Ackermann ★ 1896, † 1962

- **Mathematiker und Logiker**
- Promovierte bei D. Hilbert in Göttingen
(Koautor von Hilberts 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

Wilhelm Ackermann ★ 1896, † 1962

- Mathematiker und Logiker
- Promovierte bei D. Hilbert in Göttingen
(Koautor von Hilberts 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

Wilhelm Ackermann ★ 1896, † 1962

- Mathematiker und Logiker
- Promovierte bei D. Hilbert in Göttingen
(Koautor von Hilberts 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

Wilhelm Ackermann ★ 1896, † 1962

- Mathematiker und Logiker
- Promovierte bei D. Hilbert in Göttingen
(Koautor von Hilberts 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

Wilhelm Ackermann ★ 1896, † 1962

- Mathematiker und Logiker
- Promovierte bei D. Hilbert in Göttingen
(Koautor von Hilberts 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 . . .



Definition 12.3 (Ackermann-Funktion A)

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

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

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

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

Definition 12.3 (Ackermann-Funktion A)

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

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

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

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

Definition 12.3 (Ackermann-Funktion A)

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

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

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

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

Wachstumsverhalten

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

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

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

Wachstumsverhalten

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

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

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

Wachstumsverhalten

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

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

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

Theorem 12.4

Die Ackermann-Funktion ist

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

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 $F_\mu = \mathbf{WHILE}$ (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)

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 $F_\mu = \mathbf{WHILE}$ (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)

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 $F_\mu = \mathbf{WHILE}$ (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)

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 $F_\mu = \mathbf{WHILE}$ (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)

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 $F_\mu = \mathbf{WHILE}$ (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

- 1 Einführung
- 2 Primitiv rekursive Funktionen
- 3 $\wp$ = LOOP
- 4 μ -rekursive Funktionen**
- 5 F_μ = TM = WHILE
- 6 Zusammenfassung

Teil III

Rekursive Funktionen

- 1 Einführung
- 2 Primitiv rekursive Funktionen
- 3 $\wp = \text{LOOP}$
- 4 μ -rekursive Funktionen
- 5 $F_\mu = \text{TM} = \text{WHILE}$
- 6 Zusammenfassung

Aus dem Vorlesungsteil „Registermaschinen“ wissen wir:

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

In diesem Abschnitt schon gezeigt:

- $\text{LOOP} = \emptyset$
- $F_\mu \subseteq \text{WHILE}$ und $F_\mu^{\text{part}} \subseteq \text{WHILE}$

Noch zu zeigen:

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

Aus dem Vorlesungsteil „Registermaschinen“ wissen wir:

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

In diesem Abschnitt schon gezeigt:

- $\text{LOOP} = \emptyset$
- $F_\mu \subseteq \text{WHILE}$ und $F_\mu^{\text{part}} \subseteq \text{WHILE}$

Noch zu zeigen:

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

Aus dem Vorlesungsteil „Registermaschinen“ wissen wir:

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

In diesem Abschnitt schon gezeigt:

- $\text{LOOP} = \emptyset$
- $F_\mu \subseteq \text{WHILE}$ und $F_\mu^{\text{part}} \subseteq \text{WHILE}$

Noch zu zeigen:

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

Gödelisierung von Turing-Maschinen

Gödelisierung von Turing-Maschinen

Man kann jeder Turing-Maschine

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

eindeutig eine Gödelnummer

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

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

Man kann jeder Turing-Maschine

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

eindeutig eine Gödelnummer

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

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.

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.

Lemma 13.1 (Simulationslemma)

Es gibt eine **primitiv rekursive** Funktion

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

so dass für jede Turing-Maschine M gilt:

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

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

dann gilt

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

Lemma 13.1 (Simulationslemma)

Es gibt eine **primitiv rekursive** Funktion

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

so dass für jede Turing-Maschine M gilt:

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

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

dann gilt

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

Lemma 13.1 (Simulationslemma)

Es gibt eine **primitiv rekursive** Funktion

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

so dass für jede Turing-Maschine M gilt:

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

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

dann gilt

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

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])

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])

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, Prie])

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, Prie])

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:

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

Beweisskizze

Sei

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

eine TM-berechnebare Funktion.

Sei M eine TM, die f berechnet.

TM-berechenbare Funktionen sind μ -rekursiv

Theorem 13.2

Jede TM-berechenbare Funktion ist μ -rekursiv:

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

Beweisskizze

Sei

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

eine TM-berechnebare Funktion.

Sei M eine TM, die f berechnet.

TM-berechenbare Funktionen sind μ -rekursiv

Theorem 13.2

Jede TM-berechenbare Funktion ist μ -rekursiv:

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

Beweisskizze

Sei

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

eine TM-berechnebare Funktion.

Sei M eine TM, die f berechnet.

TM-berechenbare Funktionen sind μ -rekursiv

Beweisskizze

Dann

$$f(n_1, \dots, n_k) = (f_U(\langle M \rangle, \text{start}, \mu_i((f_U(\langle M \rangle, \text{start}, i))_{\text{Zust}} = \langle h \rangle)))_w$$

wobei

$$\text{start} = \langle s, \# \underbrace{|\dots|}_{n_1} \# \dots \# \underbrace{|\dots|}_{n_k} \# \rangle$$

$\langle h \rangle$ = Gödelisierung des Haltezustands

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

$(\cdot)_w$ = Dekodierung des Wortes links vom Schreib-/Lesekopf

TM-berechenbare Funktionen sind μ -rekursiv

Beweisskizze

Dann

$$f(n_1, \dots, n_k) = (f_U(\langle M \rangle, \text{start}, \mu_i((f_U(\langle M \rangle, \text{start}, i))_{\text{Zust}} = \langle h \rangle)))_w$$

wobei

$$\text{start} = \langle s, \# \underbrace{|\dots|}_{n_1} \# \dots \# \underbrace{|\dots|}_{n_k} \# \rangle$$

$\langle h \rangle$ = Gödelisierung des Haltezustands

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

$(\cdot)_w$ = Dekodierung des Wortes links vom Schreib-/Lesekopf

Korollar (Kleenesche Normalform)

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

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

also

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

Teil III

Rekursive Funktionen

- 1 Einführung
- 2 Primitiv rekursive Funktionen
- 3 $\wp = \text{LOOP}$
- 4 μ -rekursive Funktionen
- 5 $F_\mu = \text{TM} = \text{WHILE}$
- 6 Zusammenfassung

Teil III

Rekursive Funktionen

- 1 Einführung
- 2 Primitiv rekursive Funktionen
- 3 $\wp = \text{LOOP}$
- 4 μ -rekursive Funktionen
- 5 $F_\mu = \text{TM} = \text{WHILE}$
- 6 **Zusammenfassung**

Klassen berechenbarer Funktionen

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

Klassen berechenbarer Funktionen

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