

# Vorlesung

# Theoretische Informatik II

**Bernhard Beckert**

**Institut für Informatik**



**Wintersemester 2007/2008**

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

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

# 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

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

# 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

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

# 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

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

# 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

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

# Teil III

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

# Teil III

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

# 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

## 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

## 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

## 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  $g_1, \dots, g_r, h_1, \dots, h_r$  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  $g_1, \dots, g_r, h_1, \dots, h_r$  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  $g_1, \dots, g_r, h_1, \dots, h_r$  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  $g_1, \dots, g_r, h_1, \dots, h_r$  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:  $x_2 := x_1 + 1$

$\pi_i^k$ :  $x_{k+1} := x_i$

# $\wp = \text{LOOP}$

## 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:  $x_2 := x_1 + 1$

$\pi_i^k$ :  $x_{k+1} := x_i$

# $\wp = \text{LOOP}$

## 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:  $x_2 := x_1 + 1$   
 $\pi_i^k$ :  $x_{k+1} := x_i$

# $\wp = \text{LOOP}$

## 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:  $x_2 := x_1 + 1$

$\pi_i^k$ :  $x_{k+1} := x_i$

# $\wp = \text{LOOP}$

## 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:  $x_2 := x_1 + 1$

$\pi_j^k$ :  $x_{k+1} := x_j$

# $\wp = \text{LOOP}$

## 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:  $x_2 := x_1 + 1$

$\pi_j^k$ :  $x_{k+1} := x_j$

## 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

$$P'_{g_1,1}; \dots; P'_{g_r,r}; P'_h$$

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

## 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.  $P \equiv x_j := x_j + 1$

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

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

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

## Beweisskizze

### Konstruktion von $f_P$

2.a.  $P \equiv x_j := x_j + 1$

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

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

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

## Beweisskizze

### Konstruktion von $f_P$

2.a.  $P \equiv x_j := x_j + 1$

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

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

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

## Beweisskizze

### Konstruktion von $f_P$

2.a.  $P \equiv x_j := x_j + 1$

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

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

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

## Beweisskizze

### Konstruktion von $f_P$

2.a.  $P \equiv x_j := x_j + 1$

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

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

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

## 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:

$$f_P = f_2 \circ f_1$$

## 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:

$$f_P = f_2 \circ f_1$$

## 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

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

# Teil III

1 Einführung

2 Primitiv rekursive Funktionen

3  $\wp = \text{LOOP}$

**4  $\mu$ -rekursive Funktionen**

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

6 Zusammenfassung

## $\mu$ -Operator

Ist

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

$\mu$ -rekursiv,

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

$$f(\mathbf{n}) = \mu i (g(\mathbf{n}, i) = 0) = \begin{cases} i_0 & \text{falls } g(\mathbf{n}, i_0) = 0 \\ & \text{und f\u00fcr } 0 \leq j < i_0: \\ & \quad g(\mathbf{n}, j) \text{ def. u. } g(\mathbf{n}, j) \neq 0 \\ \text{undef} & \text{sonst} \end{cases}$$

$\mu$ -rekursiv

## $\mu$ -Operator

Schreibweise ohne Argumente:

$$f = \mu g$$