

## Primitiv und $\mu$ -rekursive Funktionen

Prof. Dr. David Sabel

LFE Theoretische Informatik



Letzte Änderung der Folien: 2. Juli 2022

## Primitiv rekursive Funktionen

### Definition

Eine Funktion  $f : \mathbb{N}^k \rightarrow \mathbb{N}$  ist **primitiv rekursiv**, wenn sie der folgenden induktiven Definition genügt:

- Jede **konstante Funktion**  $f(x_1, \dots, x_k) = c \in \mathbb{N}$  ist primitiv rekursiv.
- Die **Projektionsfunktionen**  $\pi_i^k(x_1, \dots, x_k) = x_i$  sind primitiv rekursiv.
- Die **Nachfolgerfunktion**  $\text{succ}(x) = x + 1$  ist primitiv rekursiv.
- **Komposition / Einsetzung**: Wenn  $g : \mathbb{N}^m \rightarrow \mathbb{N}$  und für  $i = 1, \dots, m$ :  $h_i : \mathbb{N}^k \rightarrow \mathbb{N}$  primitiv rekursiv sind, dann ist auch  $f$  mit  $f(x_1, \dots, x_k) = g(h_1(x_1, \dots, x_k), \dots, h_m(x_1, \dots, x_k))$  primitiv rekursiv.
- **Primitive Rekursion**: Wenn  $g : \mathbb{N}^{k-1} \rightarrow \mathbb{N}$  und  $h : \mathbb{N}^{k+1} \rightarrow \mathbb{N}$  primitiv rekursiv, dann ist auch  $f$  mit

$$f(x_1, \dots, x_k) = \begin{cases} g(x_2, \dots, x_k), & \text{wenn } x_1 = 0 \\ h(f(x_1 - 1, x_2, \dots, x_k), x_1 - 1, x_2, \dots, x_k), & \text{sonst} \end{cases}$$

primitiv rekursiv.

## Rekursive Funktionen

Weitere Formalismen zur [Definition der Berechenbarkeit](#)

- Primitiv rekursive Funktionen und
- $\mu$ -rekursive Funktionen

Wir werden schließlich sehen:

- Primitiv rekursive Funktionen entsprechen genau den LOOP-berechenbaren Funktionen
- $\mu$ -rekursive Funktionen entsprechen genau den Turing-berechenbaren (WHILE-, GOTO-berechenbaren) Funktionen

## Konstruktionen

### Komponenten eines Tupels entfernen / vertauschen / vervielfachen

Wenn  $g : \mathbb{N}^4 \rightarrow \mathbb{N}$  primitiv rekursiv, dann ist auch  $f : \mathbb{N}^3 \rightarrow \mathbb{N}$  mit

$$f(n_1, n_2, n_3) = g(n_2, n_3, n_3, n_2),$$

denn:

$$f(n_1, n_2, n_3) = g(\pi_2^3(n_1, n_2, n_3), \pi_3^3(n_1, n_2, n_3), \pi_3^3(n_1, n_2, n_3), \pi_2^3(n_1, n_2, n_3))$$

## Konstruktionen (2)

### Rekursion durch das $i$ . Argument

Für  $1 \leq i \leq k$  kann man

$$f(x_1, \dots, x_k) = \begin{cases} g(x_1, \dots, x_{i-1}, x_{i+1}, x_k), & \text{falls } x_i = 0 \\ h(f(x_1, \dots, x_{i-1}, x_i - 1, x_{i+1}, \dots, x_k), x_1, \dots, x_{i-1}, x_i - 1, x_{i+1}, \dots, x_k), & \text{sonst} \end{cases}$$

durch  $f(x_1, \dots, x_k) = f'(x_i, x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_k)$  darstellen, wobei

$$\begin{aligned} f'(x_1, \dots, x_k) &= \begin{cases} g'(x_2, \dots, x_k), & \text{falls } x_1 = 0 \\ h'(f'(x_1 - 1, x_2, \dots, x_k), x_1 - 1, x_2, \dots, x_k), & \text{sonst} \end{cases} \\ g'(x_1, \dots, x_{k-1}) &= g(x_2, \dots, x_i, x_1, x_{i+1}, \dots, x_k) \\ h'(x_1, \dots, x_{k+1}) &= h(x_1, x_2, \dots, x_i, x_1 - 1, x_{i+1}, \dots, x_k) \end{aligned}$$

## Beispiele (1)

### Additionsfunktion

$add(x_1, x_2) = x_1 + x_2$  ist primitiv rekursiv:

$$add(x_1, x_2) = \begin{cases} x_2, & \text{falls } x_1 = 0 \\ succ(add(x_1 - 1, x_2)), & \text{sonst} \end{cases}$$

Bemerkung:

Die verwendeten Funktionen  $g$  und  $h$  aus der Definition der primitiv rekursiven Funktionen sind hier:

- $g = \pi_1^1$
- $h(x_1, x_2, x_3) = succ(\pi_1^3(x_1, x_2, x_3))$

## Beispiele (2)

### Multiplikationsfunktion

$$mult(x_1, x_2) = \begin{cases} 0, & \text{falls } x_1 = 0 \\ add(mult(x_1 - 1, x_2), x_2), & \text{sonst} \end{cases}$$

Idee:  $x_1$ -mal  $x_2$  zu 0 addieren:

## Beispiele (3)

### Differenz

Allgemein  $x_1 - x_2$  nicht primitiv rekursiv, undefinierter Fall  $x_1 < x_2$  nicht darstellbar.

### Angepasste Differenz

liefert 0 falls  $x_1 < x_2$ , ist primitiv rekursiv:

$$\begin{aligned} sub(x_1, x_2) &= \begin{cases} x_1, & \text{falls } x_2 = 0 \\ pred(sub(x_1, x_2 - 1)), & \text{sonst} \end{cases} \\ \text{wobei} \\ pred(x_1) &= \begin{cases} 0, & \text{falls } x_1 = 0 \\ x_1 - 1, & \text{sonst} \end{cases} \end{aligned}$$

## Nächstes Ziel

Wir wollen zeigen:

Primitiv rekursive Funktionen sind genau die LOOP-berechenbaren Funktionen

Benötigt:

- Darstellung der Variablenbelegung  $\rho$  als **eine einzige Zahl**, um sie der primitiv rekursiven Funktion als Argument zu übergeben.
- D.h. eindeutige Darstellung eines Tupels natürlicher Zahlen als eine einzige Zahl.
- Operationen zum Konvertieren in beide Richtungen

Eine solches Verfahren nennt man auch „Gödelisierung“ (nach Kurt Gödel)

## Gödelisierung (1)

- Tupel von natürlichen Zahlen  $(x_0, \dots, x_k)$  bijektiv in die natürlichen Zahlen abbilden
- Für festes  $k$
- mit **primitiv rekursiven** Funktionen

$$c(x, y) = \binom{x + y + 1}{2} + x$$

Werte von  $c(x, y)$  für  $x, y \in \{0, \dots, 5\}$ :

$y \backslash x$	0	1	2	3	4	5
0	0	1	3	6	10	15
1	2	4	7	11	16	22
2	5	8	12	17	23	30
3	9	13	18	24	31	39
4	14	19	25	32	40	49
5	20	26	33	41	50	60

## Gödelisierung (2)

Funktion  $c$  ist primitiv rekursiv, da

$$\binom{0}{2} = 0 \text{ und } \binom{n+1}{2} = \binom{n}{2} + n$$

Für  $k+1$ -Tupel definieren wir:

$$\langle x_0, \dots, x_k \rangle = c(x_0, c(x_1, \dots, c(x_k, 0) \dots))$$

Beachte:  $\langle \cdot \rangle$  ist primitiv rekursiv

(da  $c$  primitiv rekursiv und Komposition primitiv rekursiv)

## Gödelisierung (3)

Rückgewinnung der Komponenten:

Seien  $left$  und  $right$  Funktionen mit

- $left(c(x, y)) = x$  und
- $right(c(x, y)) = y$ .

Im Skript wird gezeigt:

- $left$  und  $right$  existieren
- $left$  und  $right$  sind primitiv rekursiv

## Gödelisierung (5)

Zugriff auf beliebige Komponenten:

Programmiere  $d_i(\langle x_0, \dots, x_k \rangle) = x_i$  durch:

$$\begin{aligned}d_0(x) &= \text{left}(x) \\d_1(x) &= \text{left}(\text{right}(x)) \\d_i(x) &= \text{left}(\underbrace{\text{right}(\text{right} \dots \text{right}(x) \dots)}_{i \text{ mal}})\end{aligned}$$

Damit sind auch die  $d_i$ -Funktionen primitiv rekursiv.

## Von LOOP-Programm berechnete Funktion

- Sei  $P$  ein LOOP-Programm,  $\rho$  eine Variablenbelegung mit

$$(\rho, P) \xrightarrow[\text{LOOP}]{*} (\rho', \varepsilon)$$

- Seien  $x_0, x_1, \dots, x_n$  alle vom Programm  $P$  verwendeten Variablen (auch solche, die nicht in  $\rho$  vorkommen)
- Die von  $P$  berechnete Funktion:

$$g_P(\langle x_0, \dots, x_n \rangle) = \langle x'_0, \dots, x'_n \rangle$$

(wobei  $x'_i = \rho'(x_i)$ )

## LOOP-Programme berechnen primitiv rekursive Funktionen

### Lemma

Für jedes LOOP-Programm  $P$  ist die zugehörige Funktion  $g_P$  primitiv rekursiv.

Beweis: Strukturelle Induktion über jedes Teilprogramm  $Q$  und die zugeh. Funktion  $g_Q$ .

- Basis:  $Q$  ist Zuweisung  $x_i = x_j \pm c$ .  
Für  $g_Q$  muss gelten:

$$g_Q(\langle m_0, \dots, m_n \rangle) = \langle m_0, \dots, m_{i-1}, m_j \pm c, m_i, \dots, m_n \rangle$$

Primitiv rekursive Implementierung:

$$g_Q(x) = \langle d_0(x), \dots, d_{i-1}(x), d_j(x) + c, d_{i+1}(x), \dots, d_n(x) \rangle$$

## LOOP-Programme berechnen primitiv rekursive Funktionen (2)

Induktionsschritt:

- $Q$  ist eine Sequenz  $Q_1; Q_2$ .  
Induktionshypothese: primitiv rekursive Funktionen  $g_{Q_1}, g_{Q_2}$ .  
Funktion  $g_Q(x) = g_{Q_2}(g_{Q_1}(x))$  ist primitiv rekursiv.

- $Q$  ist **LOOP**  $x_i$  **DO**  $P$  **END**.  
Induktionshypothese liefert primitiv rekursive Funktion  $g_P$ .  
Konstruktion von  $g_Q$ :  $x_i$ -mal wird  $g_P$  angewendet.

$$\begin{aligned}g_Q(x) &= \text{run}(d_i(x), x) \\ \text{run}(n, x) &= \begin{cases} x, & \text{falls } n = 0 \\ g_P(\text{run}(n-1, x)) & \text{sonst} \end{cases}\end{aligned}$$

$\text{run}$  ist primitiv rekursiv. Damit auch  $g_Q$  primitiv rekursiv.

## LOOP-berechenbare Funktionen sind primitiv rekursiv

### Satz 11.1.6

Jede LOOP-berechenbare Funktion ist primitiv rekursiv.

Beweis:

- Sei  $f : \mathbb{N}^k \rightarrow \mathbb{N}$  LOOP-berechenbar.
- Sei  $P$  LOOP-Programm mit  $\rho = \{x_1 \mapsto n_1, \dots, x_k \mapsto n_k\}$ ,  $(\rho, P) \xrightarrow[\text{LOOP}]^* (\rho', \varepsilon)$  und  $\rho'(x_0) = f(n_1, \dots, n_k)$
- Es gilt  $f(n_1, \dots, n_k) = d_0(g_P(\langle 0, n_1, \dots, n_k, 0, \dots, 0 \rangle))$ .
- Da  $\langle \cdot \rangle$ ,  $d_0$  und  $g_P$  primitiv rekursiv sind, ist auch  $f$  primitiv rekursiv.

## Primitiv rekursive Funktionen sind LOOP-berechenbar

### Satz 11.1.7

Jede primitiv rekursive Funktion ist LOOP-berechenbar.

Beweis: Induktion über Struktur der primitiv rekursiven Funktion:

- Wenn  $f(x) = c$ ,  $f = succ$ , oder  $f = \pi_n^k$ , dann gibt es auch LOOP-Programm dazu.
- Wenn  $f(x_1, \dots, x_k) = h(g_1(x_1, \dots, x_k), \dots, g_n(x_1, \dots, x_k))$ : Induktionshypothese liefert LOOP-Programme  $P_h$  und  $P_{g_1}, \dots, P_{g_n}$ , die  $h, g_1, \dots, g_n$  berechnen. Konstruiere Programm für  $f$  nach dem **Schema**:

$$y_1 := g_1(x_1, \dots, x_k); \dots; y_n := g_n(x_1, \dots, x_k); x_0 := h(y_1, \dots, y_n)$$

Genauer:  $P_{g_1}, \dots, P_{g_n}, P_h$  abändern, so dass sie auf disjunkten Variablenmengen arbeiten, entsprechende Variableninhalte für  $x_1, \dots, x_k$  verdoppeln.

- ...

## Primitiv rekursive Funktionen sind LOOP-berechenbar (2)

- ...
- $f$  ist primitiv rekursiv:

$$f(x_1, \dots, x_k) = \begin{cases} g(x_2, \dots, x_k) & \text{wenn } x_1 = 0 \\ h(f(x_1 - 1, x_2, \dots, x_k), x_1 - 1, x_2, \dots, x_k) & \text{sonst} \end{cases}$$

wobei  $g : \mathbb{N}^{k-1} \rightarrow \mathbb{N}$  und  $h : \mathbb{N}^{k+1} \rightarrow \mathbb{N}$  primitiv rekursiv sind. Induktionshypothese liefert LOOP-Programme, die  $g, h$  berechnen. Konstruiere LOOP-Programm für  $f$  nach dem **Schema**

```
y := 0;
x_0 := g(x_2, ..., x_k);
LOOP x_1 DO y := y + 1; x_0 := h(x_0, y, x_2, ..., x_k) END
```

## LOOP-berechenbar = primitiv rekursiv

### Theorem 11.1.8

Die primitiv rekursiven Funktionen sind genau die LOOP-berechenbaren Funktionen.

## $\mu$ -Operator

### Definition $\mu$ -Operator

Sei  $h : \mathbb{N}^{k+1} \rightarrow \mathbb{N}$  eine (partielle oder totale) Funktion.

Dann ist  $(\mu h) : \mathbb{N}^k \rightarrow \mathbb{N}$  definiert als

$$(\mu h)(x_1, \dots, x_k) = \begin{cases} n & \text{falls } h(n, x_1, \dots, x_k) = 0 \text{ und f\u00fcr} \\ & \text{alle } m < n: h(m, x_1, \dots, x_k) \text{ ist} \\ & \text{definiert und } h(m, x_1, \dots, x_k) > 0. \\ \text{undefiniert,} & \text{sonst} \end{cases}$$

- $\mu$ -Operator „sucht“ nach einer kleinsten Nullstelle von  $h$ .
- Wenn diese nicht existiert (entweder da  $h$  keine Nullstelle hat, oder da  $h$  undefiniert ist f\u00fcr Werte, die kleiner als die Nullstelle sind), dann ist auch der  $\mu$ -Operator angewendet auf  $h$  undefiniert.

## $\mu$ -rekursive Funktionen

### Definition

Die Menge aller  $\mu$ -rekursiven Funktionen sei die kleinste Menge, so dass gilt:

- Jede **konstante Funktion**  $f(x_1, \dots, x_k) = c \in \mathbb{N}$  ist  $\mu$ -rekursiv.
- Die **Projektionsfunktionen**  $\pi_i^k(x_1, \dots, x_k) = x_i$  sind  $\mu$ -rekursiv.
- Die **Nachfolgerfunktion**  $\text{succ}(x) = x + 1$  ist  $\mu$ -rekursiv.
- **Komposition / Einsetzung:** Wenn  $g : \mathbb{N}^m \rightarrow \mathbb{N}$  und f\u00fcr  $i = 1, \dots, m: h_i : \mathbb{N}^k \rightarrow \mathbb{N}$   $\mu$ -rekursiv sind, dann ist auch  $f$  mit  $f(x_1, \dots, x_k) = g(h_1(x_1, \dots, x_k), \dots, h_m(x_1, \dots, x_k))$   $\mu$ -rekursiv.
- **Rekursion:** Wenn  $g : \mathbb{N}^{k-1} \rightarrow \mathbb{N}$  und  $h : \mathbb{N}^{k+1} \rightarrow \mathbb{N}$   $\mu$ -rekursiv, dann ist

$$f(x_1, \dots, x_k) = \begin{cases} g(x_2, \dots, x_k), & \text{wenn } x_1 = 0 \\ h(f(x_1 - 1, x_2, \dots, x_k), x_1 - 1, x_2, \dots, x_k), & \text{sonst} \end{cases}$$

auch  $\mu$ -rekursiv.

- **$\mu$ -Operator:** Wenn  $h : \mathbb{N}^{k+1} \rightarrow \mathbb{N}$   $\mu$ -rekursiv, dann auch  $f = \mu h$   $\mu$ -rekursiv.

## WHILE-berechenbare Funktionen sind $\mu$ -rekursiv

### Satz 11.2.3

Jede WHILE-berechenbare Funktion ist  $\mu$ -rekursiv.

Beweis: Analog zu Satz 11.1.6 \u00fcber Struktur des WHILE-Programms  $P$

Neuer Fall:  $P$  ist **WHILE**  $x_i \neq 0$  **DO**  $Q$  **END**

Induktionshypothese liefert  $\mu$ -rekursive Funktion  $g_Q$  f\u00fcr  $Q$ .

Konstruiere:

$$\begin{aligned} g_P(x) &= \text{run}(\mu(\text{run}_i)(x), x) \\ \text{run}_i(n, x) &= d_i(\text{run}(n, x)) \\ \text{run}(n, x) &= \begin{cases} x, & \text{falls } n = 0 \\ g_Q(\text{run}(n - 1, x)) & \text{sonst} \end{cases} \end{aligned}$$

- $\text{run}(n, x)$  f\u00fchrt  $n$ -mal  $g_Q$  aus.
- $\mu(\text{run}_i)(x)$  berechnet, wie oft die Schleife minimal durchlaufen werden muss, bis  $x_i$  den Wert 0 hat.

## $\mu$ -rekursive Funktionen sind WHILE-berechenbar

### Satz 11.2.4

Jede  $\mu$ -rekursive Funktion ist WHILE-berechenbar.

Beweis: Analog zu Satz 11.1.7 \u00fcber die Struktur der Funktion.

Neuer Fall:  $f = \mu h$  f\u00fcr eine  $\mu$ -rek. Funktion  $h$ : Die Induktionshypothese liefert WHILE-Programm  $P_h$ , das  $h$  berechnet. Konstruiere  $P_f$  nach dem **Schema**:

```
 $x_0 := 0;$   
 $y := h(0, x_1, \dots, x_n);$   
WHILE  $y \neq 0$  DO  
     $x_0 := x_0 + 1; y := h(x_0, \dots, x_n)$   
END
```

- **WHILE**-Schleife berechnet minimalen Wert f\u00fcr  $h(x_0, \dots, x_n) = 0$
- Wenn dieser nicht existiert, terminiert die Schleife nicht.
- Entspricht der Berechnung von  $\mu h$ .

### Theorem 11.2.5

Die  $\mu$ -rekursiven Funktionen entsprechen genau den WHILE-berechenbaren (und damit auch den GOTO- und Turingberechenbaren) Funktionen.

