SS 2005

Einführung in die Informatik IV

Ernst W. Mayr

Fakultät für Informatik
TU München

http://www14.in.tum.de/lehre/2005SS/info4/index.html.de

10. Juni 2005

1.4 Primitiv rekursive Funktionen

Betrachte die kleinste Klasse der Funktion $\mathbb{N}_0^k \to \mathbb{N}_0$, für die gilt:

1. Sie enthält die konstanten Funktionen.
2. Sie enthält die Nachfolgerfunktionen: $n \mapsto n+1$.
3. Sie enthält die Projektionsfunktionen:

$$\text{proj}_{k,i} : \mathbb{N}_0^k \ni (x_1, \ldots, x_k) \mapsto x_i \in \mathbb{N}_0$$

4. Sie ist abgeschlossen unter Komposition.
5. Sie ist abgeschlossen unter primitiver Rekursion, d.h. mit

$$g : \mathbb{N}_0^n \to \mathbb{N}_0$$
$$h : \mathbb{N}_0^{n+2} \to \mathbb{N}_0$$

ist auch

$$f(0, y_1, \ldots, y_n) := g(y_1, \ldots, y_n)$$
$$f(m+1, y_1, \ldots, y_n) := h(f(m, y_1, \ldots, y_n), m, y_1, \ldots, y_n)$$

(primitive Rekursion) in der Klasse (und sonst nichts).

1.4 Primitiv rekursive Funktionen

Betrachte die kleinste Klasse der Funktion $\mathbb{N}_0^k \to \mathbb{N}_0$, für die gilt:

1. Sie enthält die konstanten Funktionen.
2. Sie enthält die Nachfolgerfunktionen: $n \mapsto n+1$.
3. Sie enthält die Projektionsfunktionen:

$$\text{proj}_{k,i} : \mathbb{N}_0^k \ni (x_1, \ldots, x_k) \mapsto x_i \in \mathbb{N}_0$$

4. Sie ist abgeschlossen unter Komposition.
5. Sie ist abgeschlossen unter primitiver Rekursion, d.h. mit

$$g : \mathbb{N}_0^n \to \mathbb{N}_0$$
$$h : \mathbb{N}_0^{n+2} \to \mathbb{N}_0$$

ist auch

$$f(0, y_1, \ldots, y_n) := g(y_1, \ldots, y_n)$$
$$f(m+1, y_1, \ldots, y_n) := h(f(m, y_1, \ldots, y_n), m, y_1, \ldots, y_n)$$

(primitive Rekursion) in der Klasse (und sonst nichts).

1.4 Primitiv rekursive Funktionen

Betrachte die kleinste Klasse der Funktion $\mathbb{N}_0^k \to \mathbb{N}_0$, für die gilt:

1. Sie enthält die konstanten Funktionen.
2. Sie enthält die Nachfolgerfunktionen: $n \mapsto n+1$.
3. Sie enthält die Projektionsfunktionen:

$$\text{proj}_{k,i} : \mathbb{N}_0^k \ni (x_1, \ldots, x_k) \mapsto x_i \in \mathbb{N}_0$$

4. Sie ist abgeschlossen unter Komposition.
5. Sie ist abgeschlossen unter primitiver Rekursion, d.h. mit

$$g : \mathbb{N}_0^n \to \mathbb{N}_0$$
$$h : \mathbb{N}_0^{n+2} \to \mathbb{N}_0$$

ist auch

$$f(0, y_1, \ldots, y_n) := g(y_1, \ldots, y_n)$$
$$f(m+1, y_1, \ldots, y_n) := h(f(m, y_1, \ldots, y_n), m, y_1, \ldots, y_n)$$

(primitive Rekursion) in der Klasse (und sonst nichts).

1.4 Primitiv rekursive Funktionen

Betrachte die kleinste Klasse der Funktion $\mathbb{N}_0^k \to \mathbb{N}_0$, für die gilt:

1. Sie enthält die konstanten Funktionen.
2. Sie enthält die Nachfolgerfunktionen: $n \mapsto n+1$.
3. Sie enthält die Projektionsfunktionen:

$$\text{proj}_{k,i} : \mathbb{N}_0^k \ni (x_1, \ldots, x_k) \mapsto x_i \in \mathbb{N}_0$$

4. Sie ist abgeschlossen unter Komposition.
5. Sie ist abgeschlossen unter primitiver Rekursion, d.h. mit

$$g : \mathbb{N}_0^n \to \mathbb{N}_0$$
$$h : \mathbb{N}_0^{n+2} \to \mathbb{N}_0$$

ist auch

$$f(0, y_1, \ldots, y_n) := g(y_1, \ldots, y_n)$$
$$f(m+1, y_1, \ldots, y_n) := h(f(m, y_1, \ldots, y_n), m, y_1, \ldots, y_n)$$

(primitive Rekursion) in der Klasse (und sonst nichts).

1.4 Primitiv rekursive Funktionen

Betrachte die kleinste Klasse der Funktion $\mathbb{N}_0^k \to \mathbb{N}_0$, für die gilt:

1. Sie enthält die konstanten Funktionen.
2. Sie enthält die Nachfolgerfunktionen: $n \mapsto n+1$.
3. Sie enthält die Projektionsfunktionen:

$$\text{proj}_{k,i} : \mathbb{N}_0^k \ni (x_1, \ldots, x_k) \mapsto x_i \in \mathbb{N}_0$$

4. Sie ist abgeschlossen unter Komposition.
5. Sie ist abgeschlossen unter primitiver Rekursion, d.h. mit

$$g : \mathbb{N}_0^n \to \mathbb{N}_0$$
$$h : \mathbb{N}_0^{n+2} \to \mathbb{N}_0$$

ist auch

$$f(0, y_1, \ldots, y_n) := g(y_1, \ldots, y_n)$$
$$f(m+1, y_1, \ldots, y_n) := h(f(m, y_1, \ldots, y_n), m, y_1, \ldots, y_n)$$

(primitive Rekursion) in der Klasse (und sonst nichts).

1.4 Primitiv rekursive Funktionen

Betrachte die kleinste Klasse der Funktion $\mathbb{N}_0^k \to \mathbb{N}_0$, für die gilt:

1. Sie enthält die konstanten Funktionen.
2. Sie enthält die Nachfolgerfunktionen: $n \mapsto n+1$.
3. Sie enthält die Projektionsfunktionen:

$$\text{proj}_{k,i} : \mathbb{N}_0^k \ni (x_1, \ldots, x_k) \mapsto x_i \in \mathbb{N}_0$$

4. Sie ist abgeschlossen unter Komposition.
5. Sie ist abgeschlossen unter primitiver Rekursion, d.h. mit

$$g : \mathbb{N}_0^n \to \mathbb{N}_0$$
$$h : \mathbb{N}_0^{n+2} \to \mathbb{N}_0$$

ist auch

$$f(0, y_1, \ldots, y_n) := g(y_1, \ldots, y_n)$$
$$f(m+1, y_1, \ldots, y_n) := h(f(m, y_1, \ldots, y_n), m, y_1, \ldots, y_n)$$

(primitive Rekursion) in der Klasse (und sonst nichts).

Die soeben definierte Funktionenklasse sind die primitiv rekursiven Funktionen.

Beispiel 123

Die folgenden Funktionen sind primitiv rekursiv:

1. $(x, y) \mapsto x + y$;
2. $(x, y) \mapsto x * y$;
3. $(x, y) \mapsto \max\{x - y, 0\}$;
4. $x \mapsto 2^x$;
5. $x \mapsto 2^{2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}}$ (Turm der Höhe x).

Die soeben definierte Funktionenklasse sind die primitiv rekursiven Funktionen.

Beispiel 123

Die folgenden Funktionen sind primitiv rekursiv:

1. $(x, y) \mapsto x + y$;
2. $(x, y) \mapsto x * y$;
3. $(x, y) \mapsto \max\{x - y, 0\}$;
4. $x \mapsto 2^x$;
5. $x \mapsto 2^{2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}}$ (Turm der Höhe x).

Die soeben definierte Funktionenklasse sind die primitiv rekursiven Funktionen.

Beispiel 123

Die folgenden Funktionen sind primitiv rekursiv:

1. $(x, y) \mapsto x + y$;
2. $(x, y) \mapsto x * y$;
3. $(x, y) \mapsto \max\{x - y, 0\}$;
4. $x \mapsto 2^x$;
5. $x \mapsto 2^{2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}}$ (Turm der Höhe x).

Die soeben definierte Funktionenklasse sind die primitiv rekursiven Funktionen.

Beispiel 123

Die folgenden Funktionen sind primitiv rekursiv:

1. $(x, y) \mapsto x + y$;
2. $(x, y) \mapsto x * y$;
3. $(x, y) \mapsto \max\{x - y, 0\}$;
4. $x \mapsto 2^x$;
5. $x \mapsto 2^{2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}}$ (Turm der Höhe x).

Die soeben definierte Funktionenklasse sind die primitiv rekursiven Funktionen.

Beispiel 123

Die folgenden Funktionen sind primitiv rekursiv:

1. $(x, y) \mapsto x + y$;
2. $(x, y) \mapsto x * y$;
3. $(x, y) \mapsto \max\{x - y, 0\}$;
4. $x \mapsto 2^x$;
5. $x \mapsto 2^{2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}}$ (Turm der Höhe x).

Die soeben definierte Funktionenklasse sind die primitiv rekursiven Funktionen.

Beispiel 123

Die folgenden Funktionen sind primitiv rekursiv:

1. $(x, y) \mapsto x + y$;
2. $(x, y) \mapsto x * y$;
3. $(x, y) \mapsto \max\{x - y, 0\}$;
4. $x \mapsto 2^x$;
5. $x \mapsto 2^{2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}}$ (Turm der Höhe x).

Satz 124

Jede primitiv-rekursive Funktion ist total.

Beweis:

Induktion über den Aufbau einer primitiv-rekursiven Funktion. □

Satz 125

Jede primitiv-rekursive Funktion ist berechenbar.

Beweis:

Induktion über den Aufbau einer primitiv-rekursiven Funktion. □

Korollar 126

Die primitiv-rekursiven Funktionen sind eine echte Teilklasse der berechenbaren Funktionen.

Es gibt nicht-totale berechenbare Funktionen.

Satz 124

Jede primitiv-rekursive Funktion ist total.

Beweis:

Induktion über den Aufbau einer primitiv-rekursiven Funktion. □

Satz 125

Jede primitiv-rekursive Funktion ist berechenbar.

Beweis:

Induktion über den Aufbau einer primitiv-rekursiven Funktion. □

Korollar 126

Die primitiv-rekursiven Funktionen sind eine echte Teilklasse der berechenbaren Funktionen.

Es gibt nicht-totale berechenbare Funktionen.

Satz 124

Jede primitiv-rekursive Funktion ist total.

Beweis:

Induktion über den Aufbau einer primitiv-rekursiven Funktion. □

Satz 125

Jede primitiv-rekursive Funktion ist berechenbar.

Beweis:

Induktion über den Aufbau einer primitiv-rekursiven Funktion. □

Korollar 126

Die primitiv-rekursiven Funktionen sind eine echte Teilklasse der berechenbaren Funktionen.

Es gibt nicht-totale berechenbare Funktionen.

Satz 124

Jede primitiv-rekursive Funktion ist total.

Beweis:

Induktion über den Aufbau einer primitiv-rekursiven Funktion. □

Satz 125

Jede primitiv-rekursive Funktion ist berechenbar.

Beweis:

Induktion über den Aufbau einer primitiv-rekursiven Funktion. □

Korollar 126

Die primitiv-rekursiven Funktionen sind eine echte Teilklasse der berechenbaren Funktionen.

Es gibt nicht-totale berechenbare Funktionen.

Satz 124

Jede primitiv-rekursive Funktion ist total.

Beweis:

Induktion über den Aufbau einer primitiv-rekursiven Funktion. □

Satz 125

Jede primitiv-rekursive Funktion ist berechenbar.

Beweis:

Induktion über den Aufbau einer primitiv-rekursiven Funktion. □

Korollar 126

Die primitiv-rekursiven Funktionen sind eine echte Teilklasse der berechenbaren Funktionen.

Es gibt nicht-totale berechenbare Funktionen.

Satz 124

Jede primitiv-rekursive Funktion ist total.

Beweis:

Induktion über den Aufbau einer primitiv-rekursiven Funktion. □

Satz 125

Jede primitiv-rekursive Funktion ist berechenbar.

Beweis:

Induktion über den Aufbau einer primitiv-rekursiven Funktion. □

Korollar 126

Die primitiv-rekursiven Funktionen sind eine echte Teilklasse der berechenbaren Funktionen.

Es gibt nicht-totale berechenbare Funktionen.

Definition 127

Sei $P(x)$ ein Prädikat, d.h. ein logischer Ausdruck, der in Abhängigkeit von $x \in \mathbb{N}_0$ den Wert **true** oder **false** liefert. Dann können wir diesem Prädikat in natürlicher Weise eine 0-1 Funktion

$$\hat{P} : \mathbb{N}_0 \to \{0, 1\}$$

zuordnen, indem wir definieren, dass $\hat{P}(x) = 1$ genau dann, wenn $P(x) =$ **true** ist.

Wir nennen $P(x)$ primitiv rekursiv genau dann, wenn $\hat{P}(x)$ primitiv rekursiv ist.

Definition 127

Sei $P(x)$ ein Prädikat, d.h. ein logischer Ausdruck, der in Abhängigkeit von $x \in \mathbb{N}_0$ den Wert **true** oder **false** liefert. Dann können wir diesem Prädikat in natürlicher Weise eine 0-1 Funktion

$$\hat{P} : \mathbb{N}_0 \to \{0, 1\}$$

zuordnen, indem wir definieren, dass $\hat{P}(x) = 1$ genau dann, wenn $P(x) =$ **true** ist.

Wir nennen $P(x)$ primitiv rekursiv genau dann, wenn $\hat{P}(x)$ primitiv rekursiv ist.

Definition 128

Beschränkter max-Operator: Zu einem Prädikat $P(x)$ definieren wir

$$q : \mathbb{N}_0 \to \mathbb{N}_0$$

$$n \mapsto \begin{cases} 0 & \text{falls } \neg P(x) \text{ für alle } x < n \\ \max\{x < n;\ P(x)\} & \text{sonst} \end{cases}$$

Dann gilt: Ist P primitiv rekursiv, so auch q, denn:

$$q(0) = 0$$

$$q(n+1) = \begin{cases} n & \text{falls } P(n) \\ q(n) & \text{sonst} \end{cases}$$

$$= q(n) + \hat{P}(n) * (n - q(n))$$

Definition 128

Beschränkter max-Operator: Zu einem Prädikat $P(x)$ definieren wir

$$q : \mathbb{N}_0 \to \mathbb{N}_0$$

$$n \mapsto \begin{cases} 0 & \text{falls } \neg P(x) \text{ für alle } x < n \\ \max\{x < n;\ P(x)\} & \text{sonst} \end{cases}$$

Dann gilt: Ist P primitiv rekursiv, so auch q, denn:

$$q(0) = 0$$

$$q(n+1) = \begin{cases} n & \text{falls } P(n) \\ q(n) & \text{sonst} \end{cases}$$

$$= q(n) + \hat{P}(n) * (n - q(n))$$

Definition 129

Beschränkter Existenzquantor: Zu einem Prädikat $P(x)$ definieren wir ein neues Prädikat $Q(x)$ mittels:
$Q(n)$ ist genau dann **true**, wenn ein $x < n$ existiert, so dass $P(x) =$ **true**.

Dann gilt: Ist P primitiv rekursiv, so auch Q, denn:

$$\hat{Q}(0) = 0$$
$$\hat{Q}(n+1) = \hat{P}(n) + \hat{Q}(n) - \hat{P}(n) * \hat{Q}(n)$$

Definition 129

Beschränkter Existenzquantor: Zu einem Prädikat $P(x)$ definieren wir ein neues Prädikat $Q(x)$ mittels:
$Q(n)$ ist genau dann **true**, wenn ein $x < n$ existiert, so dass $P(x) =$ **true**.

Dann gilt: Ist P primitiv rekursiv, so auch Q, denn:

$$\hat{Q}(0) = 0$$
$$\hat{Q}(n+1) = \hat{P}(n) + \hat{Q}(n) - \hat{P}(n) * \hat{Q}(n)$$

Beispiel 130

Zur bijektiven Abbildung von 2-Tupeln (bzw. n-Tupeln bei iterierter Anwendung der Paarbildung) natürlicher Zahlen in die Menge der natürlichen Zahlen verwendet man eine Paarfunktion, z.B.:

	0	1	2	3	4	$\cdots$	n_2
0	0	2	5	9	14		
1	1	4	8	13			
2	3	7	12				
3	6	11					
$\vdots$							
n_1							

Beispiel 130

Zur bijektiven Abbildung von 2-Tupeln (bzw. n-Tupeln bei iterierter Anwendung der Paarbildung) natürlicher Zahlen in die Menge der natürlichen Zahlen verwendet man eine Paarfunktion, z.B.:

Betrachte: $p : \mathbb{N}_0^2 \to \mathbb{N}_0$ mit

$$p(n_1, n_2) := \frac{(n_1+n_2)(n_1+n_2+1)}{2} + n_2$$

$$c_1 : \mathbb{N}_0 \to \mathbb{N}_0$$

$$c_1(n) := s - (n - \frac{s(s+1)}{2}); \quad \text{wobei}$$

$$s := \max\{i;\ \frac{i(i+1)}{2} \leq n\}$$

$$c_2 : \mathbb{N}_0 \to \mathbb{N}_0$$

$$c_2(n) := n - \frac{s(s+1)}{2}, \quad s \text{ wie oben}$$

Satz 131

1. *p stellt eine primitiv rekursive, bijektive Paarfunktion von $\mathbb{N}_0^2$ nach $\mathbb{N}_0$ mit den Umkehrfunktionen c_1 und c_2 dar.*
2. *Die Umkehrfunktionen c_1, c_2 sind ebenfalls primitiv rekursiv.*

Beweis:
Übungsaufgabe. □

Satz 131

1. p *stellt eine primitiv rekursive, bijektive Paarfunktion von* $\mathbb{N}_0^2$ *nach* $\mathbb{N}_0$ *mit den Umkehrfunktionen* c_1 *und* c_2 *dar.*
2. *Die Umkehrfunktionen* c_1, c_2 *sind ebenfalls primitiv rekursiv.*

Beweis:
Übungsaufgabe. □

Satz 131

1. *p stellt eine primitiv rekursive, bijektive Paarfunktion von $\mathbb{N}_0^2$ nach $\mathbb{N}_0$ mit den Umkehrfunktionen c_1 und c_2 dar.*
2. *Die Umkehrfunktionen c_1, c_2 sind ebenfalls primitiv rekursiv.*

Beweis:
Übungsaufgabe. □

Satz 131

1. *p stellt eine primitiv rekursive, bijektive Paarfunktion von $\mathbb{N}_0^2$ nach $\mathbb{N}_0$ mit den Umkehrfunktionen c_1 und c_2 dar.*
2. *Die Umkehrfunktionen c_1, c_2 sind ebenfalls primitiv rekursiv.*

Beweis:

Übungsaufgabe. □

1.5 LOOP-Berechenbarkeit

LOOP-Programme sind wie folgt definiert:

Variablen: $x_1, x_2, x_3, \ldots$

Konstanten: 0, 1, 2, ...

Trennsymbole: ; :=

Operatoren: + -

Schlüsselwörter: LOOP DO END

Der Aufbau von LOOP-Programmen:

- $x_i := c$, $x_i := x_j + c$, $x_i := x_j - c$ sind LOOP-Programme.
  Die Interpretation dieser Ausdrücke erfolgt, wie üblich, mit der Einschränkung, dass $x_j - c$ als 0 gewertet wird, falls $c > x_j$.
- Sind P_1 und P_2 LOOP-Programme, so ist auch

  $$P_1; P_2$$

  ein LOOP-Programm.
  Interpretation: Führe zuerst P_1 und dann P_2 aus.
- Ist P ein LOOP-Programm, so ist auch

  $$\text{LOOP } x_i \text{ DO } P \text{ END}$$

  ein LOOP-Programm.
  Interpretation: Führe P genau x_i-mal aus.
  Achtung: Zuweisungen an x_i im Innern von P haben keinen Einfluss auf die Anzahl der Schleifendurchläufe!

Der Aufbau von LOOP-Programmen:

- $x_i := c$, $x_i := x_j + c$, $x_i := x_j - c$ sind LOOP-Programme.
  Die Interpretation dieser Ausdrücke erfolgt, wie üblich, mit der Einschränkung, dass $x_j - c$ als 0 gewertet wird, falls $c > x_j$.
- Sind P_1 und P_2 LOOP-Programme, so ist auch

  $$P_1; P_2$$

  ein LOOP-Programm.
  Interpretation: Führe zuerst P_1 und dann P_2 aus.
- Ist P ein LOOP-Programm, so ist auch

  $$\text{LOOP } x_i \text{ DO } P \text{ END}$$

  ein LOOP-Programm.
  Interpretation: Führe P genau x_i-mal aus.
  Achtung: Zuweisungen an x_i im Innern von P haben keinen Einfluss auf die Anzahl der Schleifendurchläufe!

Der Aufbau von LOOP-Programmen:

- $x_i := c$, $x_i := x_j + c$, $x_i := x_j - c$ sind LOOP-Programme.
  Die Interpretation dieser Ausdrücke erfolgt, wie üblich, mit der Einschränkung, dass $x_j - c$ als 0 gewertet wird, falls $c > x_j$.
- Sind P_1 und P_2 LOOP-Programme, so ist auch

  $$P_1; P_2$$

  ein LOOP-Programm.
  Interpretation: Führe zuerst P_1 und dann P_2 aus.
- Ist P ein LOOP-Programm, so ist auch

  LOOP x_i DO P END

  ein LOOP-Programm.
  Interpretation: Führe P genau x_i-mal aus.
  Achtung: Zuweisungen an x_i im Innern von P haben keinen Einfluss auf die Anzahl der Schleifendurchläufe!

Der Aufbau von LOOP-Programmen:

- $x_i := c$, $x_i := x_j + c$, $x_i := x_j - c$ sind LOOP-Programme.
  Die Interpretation dieser Ausdrücke erfolgt, wie üblich, mit der Einschränkung, dass $x_j - c$ als 0 gewertet wird, falls $c > x_j$.
- Sind P_1 und P_2 LOOP-Programme, so ist auch

  $$P_1; P_2$$

  ein LOOP-Programm.
  Interpretation: Führe zuerst P_1 und dann P_2 aus.
- Ist P ein LOOP-Programm, so ist auch

  $$\text{LOOP } x_i \text{ DO } P \text{ END}$$

  ein LOOP-Programm.
  Interpretation: Führe P genau x_i-mal aus.
  Achtung: Zuweisungen an x_i im Innern von P haben keinen Einfluss auf die Anzahl der Schleifendurchläufe!

Definition 132

Eine Funktion f heißt LOOP-berechenbar genau dann, wenn es ein LOOP-Programm gibt, das f berechnet.

LOOP-Programme können IF . . . THEN . . . ELSE . . . END Konstrukte simulieren. Der Ausdruck IF $x = 0$ THEN A END kann durch folgendes Programm nachgebildet werden:

$y := 1;$
LOOP x DO $y := 0$ END;
LOOP y DO A END;

LOOP-berechenbare Funktion sind immer total, denn:
LOOP-Programme stoppen immer. Damit stellt sich natürlich die Frage, ob alle totalen Funktionen LOOP-berechenbar sind.

Definition 132

Eine Funktion f heißt LOOP-berechenbar genau dann, wenn es ein LOOP-Programm gibt, das f berechnet.

LOOP-Programme können IF ... THEN ... ELSE ... END Konstrukte simulieren. Der Ausdruck IF $x = 0$ THEN A END kann durch folgendes Programm nachgebildet werden:

$y := 1;$
LOOP x DO $y := 0$ END;
LOOP y DO A END;

LOOP-berechenbare Funktion sind immer total, denn: LOOP-Programme stoppen immer. Damit stellt sich natürlich die Frage, ob alle totalen Funktionen LOOP-berechenbar sind.

Definition 132

Eine Funktion f heißt LOOP-berechenbar genau dann, wenn es ein LOOP-Programm gibt, das f berechnet.

LOOP-Programme können IF ... THEN ... ELSE ... END Konstrukte simulieren. Der Ausdruck IF $x = 0$ THEN A END kann durch folgendes Programm nachgebildet werden:

$y := 1;$
LOOP x DO $y := 0$ END;
LOOP y DO A END;

LOOP-berechenbare Funktion sind immer total, denn:
LOOP-Programme stoppen immer. Damit stellt sich natürlich die Frage, ob alle totalen Funktionen LOOP-berechenbar sind.

Satz 133

f ist primitiv rekursiv $\iff$ f ist LOOP-berechenbar.

Beweis:

Wir zeigen zunächst „$\Leftarrow$“: Sei also P ein LOOP-Programm, das $f : \mathbb{N}_0^n \to \mathbb{N}_0$ berechnet. P verwende die Variablen $x_1, \ldots, x_k$, $k \geq n$.

Zu zeigen: f ist primitiv rekursiv.

Der Beweis erfolgt durch strukturelle Induktion über den Aufbau von P.

Beweis:

Induktionsanfang: $P : x_i := x_j \pm c$

Wir zeigen: Es gibt eine primitiv rekursive Funktion

$$\underbrace{g_P(\underbrace{\langle a_1, \ldots, a_k \rangle}_{\text{Belegung der Variablen beim Start von } P})}_{} = \underbrace{\langle b_1, \ldots, b_k \rangle}_{\text{Belegung der Variablen am Ende von } P}$$

Für $P : x_i := x_j \pm c$ erhält man:

$$g_P(\langle a_1, \ldots, a_k \rangle) = \langle a_1, \ldots, a_{i-1}, a_j \pm c, a_{i+1}, \ldots, a_k \rangle$$

Beweis:

Induktionsschritt: Hier unterscheiden wir 2 Fälle:

1. Sei $P : Q; R$. Dann ist $g_P(x) = g_R(g_Q(x))$.
2. Sei nun P : LOOP x_i DO Q END.
   Idee: Definiere Funktion $h(n, x)$, die die Belegung der Variablen berechnet, wenn man mit Belegung x startet und dann Q genau n mal ausführt. Dann ist:

$$h(0, x) = x$$
$$h(n+1, x) = g_Q(h(n, x))$$

   und damit $g_P(x) = h(d_i(x), x)$, wobei d_i die i-te Umkehrfunktion von $\langle x_1, \ldots, x_k \rangle$ ist, also $d_i(\langle x_1, \ldots, x_k \rangle) = x_i$.

Die Richtung „$\Rightarrow$“ wird über strukturelle Induktion über den Aufbau von f gezeigt (Übungsaufgabe). □

Beweis:

Induktionsschritt: Hier unterscheiden wir 2 Fälle:

1. Sei $P : Q; R$. Dann ist $g_P(x) = g_R(g_Q(x))$.
2. Sei nun P : LOOP x_i DO Q END.
   Idee: Definiere Funktion $h(n, x)$, die die Belegung der Variablen berechnet, wenn man mit Belegung x startet und dann Q genau n mal ausführt. Dann ist:

$$h(0, x) = x$$
$$h(n + 1, x) = g_Q(h(n, x))$$

   und damit $g_P(x) = h(d_i(x), x)$, wobei d_i die i-te Umkehrfunktion von $\langle x_1, \ldots, x_k \rangle$ ist, also $d_i(\langle x_1, \ldots, x_k \rangle) = x_i$.

Die Richtung „$\Rightarrow$“ wird über strukturelle Induktion über den Aufbau von f gezeigt (Übungsaufgabe). □

Beweis:

Induktionsschritt: Hier unterscheiden wir 2 Fälle:

1. Sei $P : Q; R$. Dann ist $g_P(x) = g_R(g_Q(x))$.
2. Sei nun P : LOOP x_i DO Q END.
   Idee: Definiere Funktion $h(n, x)$, die die Belegung der Variablen berechnet, wenn man mit Belegung x startet und dann Q genau n mal ausführt. Dann ist:

$$h(0, x) = x$$
$$h(n + 1, x) = g_Q(h(n, x))$$

   und damit $g_P(x) = h(d_i(x), x)$, wobei d_i die i-te Umkehrfunktion von $\langle x_1, \ldots, x_k \rangle$ ist, also $d_i(\langle x_1, \ldots, x_k \rangle) = x_i$.

Die Richtung „$\Rightarrow$“ wird über strukturelle Induktion über den Aufbau von f gezeigt (Übungsaufgabe). □

Beweis:

Induktionsschritt: Hier unterscheiden wir 2 Fälle:

1. Sei $P : Q; R$. Dann ist $g_P(x) = g_R(g_Q(x))$.
2. Sei nun P : LOOP x_i DO Q END.
   Idee: Definiere Funktion $h(n, x)$, die die Belegung der Variablen berechnet, wenn man mit Belegung x startet und dann Q genau n mal ausführt. Dann ist:

$$h(0, x) = x$$
$$h(n + 1, x) = g_Q(h(n, x))$$

   und damit $g_P(x) = h(d_i(x), x)$, wobei d_i die i-te Umkehrfunktion von $\langle x_1, \ldots, x_k \rangle$ ist, also $d_i(\langle x_1, \ldots, x_k \rangle) = x_i$.

Die Richtung „$\Rightarrow$“ wird über strukturelle Induktion über den Aufbau von f gezeigt (Übungsaufgabe). □