Primitiv rekursive Funktionen

Falsche Variante der Subtraktion $\quad \mathrm{sub} = \mathbf{pr}\big[0^{(1)}, \mathrm{nf}\langle \pi_1^{(3)} \rangle\big]$

wesentliche Rekursion $(a+1) - b = (a-b) + 1$

$$\begin{aligned} \mathrm{sub}(0, b) &= 0 \\ \mathrm{sub}(a+1, b) &= \mathrm{nf}(\mathrm{sub}(a, b)) \end{aligned}$$

Warum nicht gewünschte Funktion?

Denn $\mathrm{sub}(a, b) = a$ für alle $a, b \in \mathbb{N}$ (trivialer Induktionsbeweis)

Primitiv rekursive Funktionen

Berechnung für $\mathrm{sub}(2, 1)$

$$\begin{aligned} \mathrm{sub}(2,1) &= \mathbf{pr}\big[0^{(1)}, \mathrm{nf}\langle \pi_1^{(3)} \rangle\big](2,1) \\ &= \mathrm{nf}\langle \pi_1^{(3)} \rangle\Big(\mathbf{pr}\big[0^{(1)}, \mathrm{nf}\langle \pi_1^{(3)} \rangle\big](1,1), 1, 1\Big) \\ &= \mathrm{nf}\bigg(\pi_1^{(3)}\Big(\mathbf{pr}\big[0^{(1)}, \mathrm{nf}\langle \pi_1^{(3)} \rangle\big](1,1), 1, 1\Big)\bigg) \\ &= \mathbf{pr}\big[0^{(1)}, \mathrm{nf}\langle \pi_1^{(3)} \rangle\big](1,1) + 1 \\ &= \mathrm{nf}\langle \pi_1^{(3)} \rangle\Big(\mathbf{pr}\big[0^{(1)}, \mathrm{nf}\langle \pi_1^{(3)} \rangle\big](0,1), 0, 1\Big) + 1 \\ &= \mathrm{nf}\bigg(\pi_1^{(3)}\Big(\mathbf{pr}\big[0^{(1)}, \mathrm{nf}\langle \pi_1^{(3)} \rangle\big](0,1), 0, 1\Big)\bigg) + 1 \\ &= \mathbf{pr}\big[0^{(1)}, \mathrm{nf}\langle \pi_1^{(3)} \rangle\big](0,1) + 2 \\ &= 0^{(1)}(1) + 2 = 0 + 2 = 2 \end{aligned}$$

Loop-Berechenbarkeit vs. primitive Rekursion

Ansatz Loop-berechenbar impliziert primitiv rekursiv

- Semantik $\|P\|_n \colon \mathbb{N}^n \to \mathbb{N}^n$ Loop-Programm P
- Primitiv rekursive Funktion $f \colon \mathbb{N}^n \to \mathbb{N}$
- Primitiv rekursive Variante $\|P\|_n$ benötigt Kodierung von $\mathbb{N}^n$ in $\mathbb{N}$ (z.B. Kellerspeicher)

Loop-Berechenbarkeit vs. primitive Rekursion

Binomialkoeffizient $\quad \mathrm{bk2} = \mathbf{pr}\big[0^{(0)}, \mathrm{add}\langle \pi_1^{(2)}, \pi_2^{(2)} \rangle\big]$

wesentliche Rekursion $\binom{a+1}{2} = \binom{a}{1} + \binom{a}{2} = a + \binom{a}{2}$

$$\begin{aligned} \mathrm{bk2}(0) &= 0 \\ \mathrm{bk2}(a+1) &= a + \mathrm{bk2}(a) \end{aligned}$$

Paarung $\quad c = \mathrm{add}\big\langle \pi_1^{(2)}, \mathrm{bk2}\big\langle \mathrm{nf}\langle \mathrm{add}\langle \pi_1^{(2)}, \pi_2^{(2)} \rangle\rangle\big\rangle\big\rangle$

$$c(a, b) = a + \mathrm{bk2}(a + b + 1) = a + \binom{a+b+1}{2}$$

Loop-Berechenbarkeit vs. primitive Rekursion

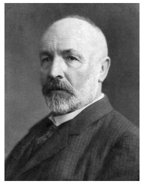
§7.1 Theorem (Cantorsche Paarungsfunktion)

$c\colon \mathbb{N}^2 \to \mathbb{N}$ bijektiv

$a \setminus b$	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

Georg Cantor (* 1845; † 1918)

- Dtsch. Mathematiker
- Begründer moderner Mengenlehre
- Kardinal- & Ordinalzahlen



Loop-Berechenbarkeit vs. primitive Rekursion

Notizen

- Offenbar $a \leq c(a, b)$ und $b \leq c(a, b)$ für alle $a, b \in \mathbb{N}$
- Kodierung Paare natürlicher Zahlen möglich

$$\begin{pmatrix} a \\ b \end{pmatrix} = c(a, b)$$

- Erweiterbar auf beliebige n-Tupel

$$\begin{pmatrix} a_1 \\ \vdots \\ a_n \end{pmatrix} = c\left(a_1, \begin{pmatrix} a_2 \\ \vdots \\ a_n \end{pmatrix}\right) = c\Big(a_1, c\big(a_2, \cdots, c(a_{n-1}, a_n) \cdots\big)\Big)$$

- Primitiv rekursiv für alle $n \in \mathbb{N}$

Loop-Berechenbarkeit vs. primitive Rekursion

Dekodierung

- Funktionen $\pi_1, \pi_2 \colon \mathbb{N} \to \mathbb{N}$

$$\pi_1\big(c(a, b)\big) = \pi_1 \begin{pmatrix} a \\ b \end{pmatrix} = a$$

$$\pi_2\big(c(a, b)\big) = \pi_2 \begin{pmatrix} a \\ b \end{pmatrix} = b$$

- Längere Tupel $\begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix} = \begin{pmatrix} a_1 \\ \begin{pmatrix} a_2 \\ a_3 \end{pmatrix} \end{pmatrix}$ ebenso dekodierbar

$$a_1 = \pi_1 \begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix} \quad a_2 = \pi_1\left(\pi_2 \begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix}\right) \quad a_3 = \pi_2\left(\pi_2 \begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix}\right)$$

Loop-Berechenbarkeit vs. primitive Rekursion

§7.2 Definition (beschränkter max-Operator; *bounded maximum*)

Sei $P\colon \mathbb{N}^{n+1} \to \{0, 1\}$ Prädikat und $\max \emptyset = 0$.

$$\max_P(a, a_1, \ldots, a_n) = \max\,\{b \leq a \mid P(b, a_1, \ldots, a_n) = 1\}$$

Notizen

- $\max_P(a, a_1, \ldots, a_n)$ maximaler Wert $b \leq a$ mit $P(b, a_1, \ldots, a_n) = 1$
- Liefert 0 falls kein Wert $b \leq a$ Prädikat $P(b, a_1, \ldots, a_n)$ erfüllt

Loop-Berechenbarkeit vs. primitive Rekursion

§7.3 Theorem

$\max_P\colon \mathbb{N}^{n+1} \to \mathbb{N}$ primitiv rek. falls $P\colon \mathbb{N}^{n+1} \to \{0,1\}$ primitiv rek.

Beweis

Offenbar $\max_P(0, a_1, \dots, a_n) = 0$ und

$$\max_P(a+1, a_1, \dots, a_n) = \begin{cases} a+1 & \text{falls } P(a+1, a_1, \dots, a_n) = 1 \\ \max_P(a, a_1, \dots, a_n) & \text{sonst} \end{cases}$$

Fallunterscheidung äquivalent zu

$$\max_P(a, a_1, \dots, a_n) + P(a+1, a_1, \dots, a_n) \cdot (a + 1 - \max_P(a, a_1, \dots, a_n)) \qquad \square$$

$$\max_P = \mathbf{pr}\big[0^{(n)}, \mathrm{add}\langle \pi_1^{(n+2)}, \mathrm{mult}\langle P\langle \mathrm{nf}\langle \pi_2^{(n+2)}\rangle, \pi_3^{(n+2)}, \dots, \pi_{n+2}^{(n+2)}\rangle, \mathrm{sub}\langle \mathrm{nf}\langle \pi_2^{(n+2)}\rangle, \pi_1^{(n+2)}\rangle\rangle\rangle\big]$$

Loop-Berechenbarkeit vs. primitive Rekursion

§7.4 Definition (beschränkter ∃-Quantor; *bounded ∃-quantifier*)

Sei $P\colon \mathbb{N}^{n+1} \to \{0,1\}$ Prädikat

$$\exists_P(a, a_1, \dots, a_n) = \begin{cases} 1 & \text{falls } \exists b \leq a\colon P(b, a_1, \dots, a_n) = 1 \\ 0 & \text{sonst} \end{cases}$$

Notizen

- $\exists_P(a, a_1, \dots, a_n) = 1$, falls $b \leq a$ mit $P(b, a_1, \dots, a_n) = 1$ existiert
- Liefert 0 falls kein Wert $b \leq a$ Prädikat $P(b, a_1, \dots, a_n)$ erfüllt

Loop-Berechenbarkeit vs. primitive Rekursion

Maximum $\qquad \max = \mathrm{add}\langle \mathrm{sub}\langle \pi_1^{(2)}, \pi_2^{(2)}\rangle, \pi_2^{(2)}\rangle$

$$\max(a, b) = (a - b) + b$$

2 Fälle: Falls $a \geq b$, dann ist $(a - b) + b = a$ und damit $\max(a, b) = a$. Sonst ist $a - b = 0$ und damit $\max(a, b) = b$.

Loop-Berechenbarkeit vs. primitive Rekursion

§7.5 Theorem

$\exists_P\colon \mathbb{N}^{n+1} \to \mathbb{N}$ primitiv rekursiv falls $P\colon \mathbb{N}^{n+1} \to \{0,1\}$ primitiv rekursiv

Beweis

Offenbar $\exists_P(0, a_1, \dots, a_n) = P(0, a_1, \dots, a_n)$ und

$$\exists_P(a+1, a_1, \dots, a_n) = \begin{cases} 1 & \text{falls } P(a+1, a_1, \dots, a_n) = 1 \\ \exists_P(a, a_1, \dots, a_n) & \text{sonst} \end{cases}$$

Fallunterscheidung äquivalent zu

$$\max(P(a+1, a_1, \dots, a_n), \exists_P(a, a_1, \dots, a_n)) \qquad \square$$

$$\exists_P = \mathbf{pr}\big[P\langle 0^{(n)}, \pi_1^{(n)}, \dots, \pi_n^{(n)}\rangle, \max\langle P\langle \mathrm{nf}\langle \pi_2^{(n+2)}\rangle, \pi_3^{(n+2)}, \dots, \pi_{n+2}^{(n+2)}\rangle, \pi_1^{(n+2)}\rangle\big]$$

Loop-Berechenbarkeit vs. primitive Rekursion

Dekodierung Paare $\binom{a}{b}$

- Definiere Prädikat $C\colon \mathbb{N}^3 \to \{0,1\}$ mit Hilfe von c

$$\boxed{C(a,b,d) = \Big(1 - \big(c(a,b) - d\big)\Big) \cdot \Big(1 - \big(d - c(a,b)\big)\Big)}$$

- C primitiv rekursiv
- $C(a,b,d) = 0$ falls $c(a,b) > d$
- $C(a,b,d) = 0$ falls $c(a,b) < d$
- $C(a,b,d) = 1$ falls $c(a,b) = d$
- Also $C(a,b,d) = 1$ gdw. $c(a,b) = d$

Loop-Berechenbarkeit vs. primitive Rekursion

Dekodierfunktionen

$$C'(b,a,d) = C(a,b,d) \qquad C' = C\langle \pi_2^{(3)}, \pi_1^{(3)}, \pi_3^{(3)} \rangle$$

$$E'(a,b,d) = \exists_{C'}(b,a,d) = \begin{cases} 1 & \text{falls } \exists y \leq b\colon C(a,y,d) = 1 \\ 0 & \text{sonst} \end{cases}$$

$$\begin{aligned} \Pi_1'(a,b,d) &= \max_{E'}(a,b,d) \\ &= \max\{x \leq a \mid E'(x,b,d) = 1\} \\ &= \max\{x \leq a \mid \exists y \leq b\colon C(x,y,d) = 1\} \\ &= \max\{x \leq a \mid \exists y \leq b\colon c(x,y) = d\} \end{aligned}$$

Loop-Berechenbarkeit vs. primitive Rekursion

Dekodierfunktionen

$$E(b,a,d) = \exists_C(a,b,d) = \begin{cases} 1 & \text{falls } \exists x \leq a\colon C(x,b,d) = 1 \\ 0 & \text{sonst} \end{cases}$$

$$\begin{aligned} \Pi_2'(a,b,d) &= \max_E(b,a,d) \\ &= \max\{y \leq b \mid E(y,a,d) = 1\} \\ &= \max\{y \leq b \mid \exists x \leq a\colon C(x,y,d) = 1\} \\ &= \max\{y \leq b \mid \exists x \leq a\colon c(x,y) = d\} \end{aligned}$$

Loop-Berechenbarkeit vs. primitive Rekursion

§7.6 Theorem (Dekodierung)

$\Pi_1\colon \mathbb{N} \to \mathbb{N}$ und $\Pi_2\colon \mathbb{N} \to \mathbb{N}$ primitiv rekursiv

Beweis

Beide Funktionen sind primitiv rekursiv da

$$\Pi_1(d) = \Pi_1'(d,d,d) \qquad \text{und} \qquad \Pi_2(d) = \Pi_2'(d,d,d) \quad \text{für alle } d \in \mathbb{N}$$

Da $a \leq c(a,b)$ und $b \leq c(a,b)$ wird d geeignet dekodiert. □

Notiz

- Verwenden Vektornotation und greifen direkt auf Komponenten zu

Loop-Berechenbarkeit vs. primitive Rekursion

§7.7 Theorem

Jede Loop-berechenbare Funktion ist primitiv rekursiv

Beweis (1/4)

Zeigen $\mathrm{sem}_P \colon \mathbb{N}^n \to \mathbb{N}$ primitiv rekursiv für jedes Loop-Programm P mit $\max \mathrm{var}(P) = n$ per Induktion über P. Für alle $a_1, \dots, a_n \in \mathbb{N}$

$$\begin{aligned} & \mathrm{sem}_P(a_1, \dots, a_n) = c(b_1, \dots, b_n) \\ \iff & \|P\|_n(a_1, \dots, a_n) = (b_1, \dots, b_n) \end{aligned}$$

- Sei P Zuweisung $x_i = x_\ell + z$. Dann

$$\mathrm{sem}_P(a_1, \dots, a_n) = c(a_1, \dots, a_{i-1}, a_\ell + z, a_{i+1}, \dots, a_n)$$

$\mathrm{sem}_P = c\langle \mathrm{nf}\langle \pi_2^{(2)} \rangle, \pi_2^{(2)} \rangle$ für $n = 2$ und $x_1 = x_2 + 1$

Loop-Berechenbarkeit vs. primitive Rekursion

Beweis (2/4)

- Sei $P = P_1 \,;\, P_2$. Dann sem_{P_1} und sem_{P_2} primitiv rekursiv gemäß IH

$$\begin{aligned} \mathrm{sem}_P(a_1, \dots, a_n) = \mathrm{sem}_{P_2}\Big(& \Pi_1(\mathrm{sem}_{P_1}(a_1, \dots, a_n)), \\ & \dots \\ & \Pi_2(\cdots \Pi_2(\mathrm{sem}_{P_1}(a_1, \dots, a_n)) \cdots)\Big) \\ = \mathrm{sem}_{P_2}&(b_1, \dots, b_n) \end{aligned}$$

mit $\mathrm{sem}_{P_1}(a_1, \dots, a_n) = c(b_1, \dots, b_n)$

$\mathrm{sem}_P = \big(\mathrm{sem}_{P_2}\langle \Pi_1, \Pi_2 \rangle\big)\langle \mathrm{sem}_{P_1} \rangle$ für $n = 2$

Loop-Berechenbarkeit vs. primitive Rekursion

Beweis (3/4)

- Sei $P = \mathbf{LOOP}(x_i)\,\{P'\}$. Dann $\mathrm{sem}_{P'}$ primitiv rekursiv gemäß IH. Definiere Funktion

$$\begin{aligned} f(0, a_1, \dots, a_n) &= c(a_1, \dots, a_n) \\ f(a + 1, a_1, \dots, a_n) &= \mathrm{sem}_{P'}(b_1, \dots, b_n) \end{aligned}$$

wobei $f(a, a_1, \dots, a_n) = c(b_1, \dots, b_n)$. Dann ist

$$\mathrm{sem}_P(a_1, \dots, a_n) = f(a_i, a_1, \dots, a_n)$$

$\mathrm{sem}_P = \left(\mathrm{pr}\left[c, \big(\mathrm{sem}_{P'}\langle \Pi_1, \Pi_2 \rangle\big)\langle \pi_1^{(4)} \rangle\right]\right)\langle \pi_2^{(2)}, \pi_1^{(2)}, \pi_2^{(2)} \rangle$ für $n = i = 2$

Loop-Berechenbarkeit vs. primitive Rekursion

Beweis (4/4)

Sei $f \colon \mathbb{N}^k \to \mathbb{N}$ Loop-berechenbar via P $\quad (k \le n)$

Für alle $a_1, \dots, a_k \in \mathbb{N}$

$$\begin{aligned} f(a_1, \dots, a_k) &= |P|_k(a_1, \dots, a_k) \\ &= \pi_1^{(n)}(\|P\|(a_1, \dots, a_k, 0, \dots, 0)) \\ &= \Pi_1(\mathrm{sem}_P(a_1, \dots, a_k, 0, \dots, 0)) \end{aligned}$$

Damit f primitiv rekursiv $\square$

Loop-Berechenbarkeit vs. primitive Rekursion

§7.8 Lemma

Für alle $k, k' \in \mathbb{N}$ und Loop-Programme P existiert Loop-Programm P' mit $\max \operatorname{var}(P') = n$ und

- $|P'|_k = |P|_k$ (gleiche berechnete Funktion)
- $\pi_i^{(n)}(\|P'\|_n(a_1, \dots, a_n)) = a_i$ (überschreibt $x_2, \dots, x_{k'}$ nicht) für alle $2 \leq i \leq k'$ und $a_1, \dots, a_n \in \mathbb{N}$

Loop-Berechenbarkeit vs. primitive Rekursion

§7.9 Theorem

Jede primitiv rekursive Funktion ist Loop-berechenbar

Beweis (1/3)

Induktion über Struktur primitiv rekursiver Funktionen

- Basisfunktionen: Trivial Loop-berechenbar (Konstanten, Projektionen, Nachfolger)

Loop-Berechenbarkeit vs. primitive Rekursion

Beweis (2/3)

- Sei $f' = f\langle g_1, \dots, g_m \rangle$ Komposition primitiv rekursiver Funktionen $f, g_1, \dots, g_m$

$$f'(a_1, \dots, a_n) = f\big(g_1(a_1, \dots, a_n), \dots, g_m(a_1, \dots, a_n)\big)$$

IH und §7.8 liefern äquivalente Loop-Programme $P, P_1, \dots, P_m$ die Variablen $x_2, \dots, x_{n+m+1}$ nicht überschreiben.
Folgendes Programm berechnet f'

$x_{n+m+1} = x_1$ (1. Eingabe sichern)
$P_m \,;\, x_{n+m} = x_1 \,;\, x_1 = x_{n+m+1}$ (Ergebnis sichern; 1. Eingabe setzen)
$\dots$
$P_1 \,;\, x_2 = x_{n+2} \,; \cdots ;\, x_m = x_{n+m}$ (Eingaben auf Ergebnisse setzen)
P

Loop-Berechenbarkeit vs. primitive Rekursion

Beweis (3/3)

- Sei $f' = \mathbf{pr}[f, g]$ primitive Rekursion mit primitiv rekursiven Funktionen f und g

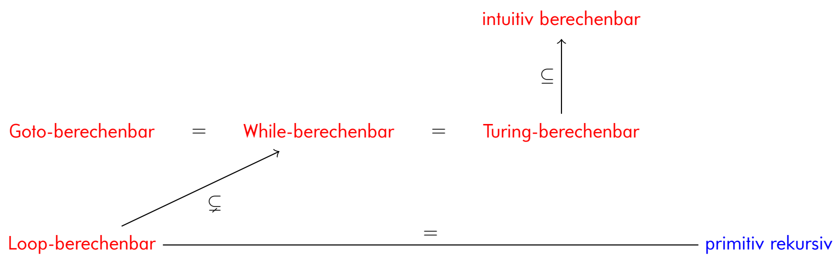
$$f'(0, a_1, \dots, a_n) = f(a_1, \dots, a_n)$$
$$f'(a+1, a_1, \dots, a_n) = g\big(f'(a, a_1, \dots, a_n), a, a_1, \dots, a_n\big)$$

IH und Lemma §7.8 liefern äquivalente Loop-Programme P_f und P_g die Variablen $x_2, \dots, x_{n+4}$ nicht überschreiben.
Folgendes Programm berechnet f'

$x_{n+3} = x_1 \,;\, x_{n+4} = x_2$ (1. & 2. Eingabe sichern)
$x_1 = x_2 \,;\, x_2 = x_3 \,; \cdots ;\, x_n = x_{n+1} \,;\, P_f$ (Eingaben für f)
$x_{n+2} = x_n \,; \cdots ;\, x_4 = x_2 \,;\, x_3 = x_{n+3} \,;\, x_2 = 0$ (Eingaben für g)
$\mathbf{LOOP}(x_{n+3})\,\{P_g \,;\, x_2 = x_2 + 1\}$ (Iterationen zählen)

$\square$

Loop-Berechenbarkeit vs. primitive Rekursion



Notizen

- Primitiv rekursive Funktionen total
- Nicht jede While-berechenbaren Funktion primitiv rekursiv (z.B. Ackermann-Funktion nicht primitiv rekursiv)
- Allgemeine Rekursion noch nicht erfasst

Rekursive partielle Funktionen

§7.10 Definition (μ-rekursiv; μ-*recursive*)

Genau folgende partielle Funktionen sind **μ-rekursiv**

- **rekursive Basisfunktionen**
- **Komposition** und **primitive Rekursion** μ-rekursiver Funktionen
- **Minimierung** $\mu f \colon \mathbb{N}^n \dashrightarrow \mathbb{N}$ (μ-Operator) μ-rekursiver Funktion $f \colon \mathbb{N}^{n+1} \dashrightarrow \mathbb{N}$ gegeben für alle $a_1, \dots, a_n \in \mathbb{N}$ durch

$$\mu f(a_1, \dots, a_n) = \min \big\{ a \in \mathbb{N} \mid f(a, a_1, \dots, a_n) = 0 \text{ und } \forall b < a \colon f(b, a_1, \dots, a_n) \text{ definiert} \big\}$$

mit $\min \emptyset = \text{undef}$

Keine weiteren partiellen Funktionen μ-rekursiv

Rekursive partielle Funktionen

Intuition

Berechnung von $\mu f(a_1, \dots, a_n)$

1. Setze $a \leftarrow 0$
2. Berechne $f(a, a_1, \dots, a_n)$
3. Liefere **undefiniert** falls $f(a, a_1, \dots, a_n)$ undefiniert (Endlosschleife)
4. Erhöhe a und zurück zu 2. falls $f(a, a_1, \dots, a_n) \neq 0$
5. Liefere a (falls $f(a, a_1, \dots, a_n) = 0$)

While-Schleife erkennbar

Rekursive partielle Funktionen

Überall undefinierte Funktion $\mu 1^{(2)} \colon \mathbb{N} \to \mathbb{N}$

- $1^{(2)}(a, b) = 1$ für alle $a, b \in \mathbb{N}$
- Also $\mu 1^{(2)}(b) = \text{undef}$ für alle $b \in \mathbb{N}$

Logarithmus $\qquad \text{ld}(13) = \lceil 3,7 \rceil = 4$

$\text{ld} \colon \mathbb{N} \to \mathbb{N}$ μ-rekursiv mit $\text{ld}(0) = 0$ und $\text{ld}(a) = \lceil \log_2(a) \rceil$ für alle $a \in \mathbb{N}_+$

- $f(a, b) = b - 2^a$ primitiv rekursiv (Loop-berechenbar)
- $\text{ld}(b) = \mu f(b)$ (kleinstes a mit $2^a \geq b$)

ld primitiv rekursiv

While-Berechenbarkeit vs. Rekursion

§7.11 Theorem

Jede While-berechenbare partielle Funktion ist μ-rekursiv

Beweis

Analog zu Loop-Programm mit neuem dritten Fall

- Sei $P = \textbf{WHILE}(x_i \neq 0)\,\{P'\}$. Dann existiert μ-rekursive Funktion $\text{sem}_{P'}$ gemäß IH. Sei

$$\begin{aligned} f(0, a_1, \dots, a_n) &= c(a_1, \dots, a_n) \\ f(a+1, a_1, \dots, a_n) &= \text{sem}_{P'}(b_1, \dots, b_n) \\ g(a, a_1, \dots, a_n) &= b_i \end{aligned}$$

wobei $f(a, a_1, \dots, a_n) = c(b_1, \dots, b_n)$

$$\text{sem}_P(a_1, \dots, a_n) = f((\mu g)(a_1, \dots, a_n), a_1, \dots, a_n) \qquad \square$$

While-Berechenbarkeit vs. Rekursion

§7.12 Theorem

Jede μ-rekursive partielle Funktion ist While-berechenbar

Beweis

Induktion über Struktur μ-rekursiver partieller Funktionen

- Sei $f' = \mu f$ für μ-rekursive Funktion f. Dann existiert äquivalentes While-Programm P ohne Überschreibung Variablen $x_2, \dots, x_{n+2}$. Programm für f'

$x_{n+2} = 0\,;\, x_{n+1} = x_n\,;\, \cdots\,;\, x_2 = x_1\,;\, x_1 = 0\,;\, P$ (Eingaben setzen)
$\textbf{WHILE}(x_1 \neq 0)\,\{x_{n+2} = x_{n+2} + 1\,;\, x_1 = x_{n+2}\,;\, P\}$
(Iteration erhöhen, Eingaben vorbereiten, weiterer Aufruf)
$x_1 = x_{n+2}$ (Ergebnis ist Anzahl Iterationen)

$\square$

While-Berechenbarkeit vs. Rekursion

$$\begin{tikzcd}
 & & & \text{intuitiv berechenbar} & & \\
\text{Goto-berechenbar} \arrow[r, phantom, "="] & \text{While-berechenbar} \arrow[r, phantom, "="] & & \text{Turing-berechenbar} \arrow[u, "\subseteq"] \arrow[r, phantom, "="] & \mu\text{-rekursiv} & \\
\text{Loop-berechenbar} \arrow[ur, "\subsetneq"] \arrow[rrrr, dash, "="] & & & & \text{primitiv rekursiv} &
\end{tikzcd}$$

Notizen

- Verschiedene Berechnungsmodelle & viele weitere existieren
- Alle höchstens Turing-Berechenbarkeit

These von Church

§7.13 Hypothese (These von Church; *Church's conjecture*)

Jede intuitiv berechenbare Funktion ist Turing-berechenbar

Alonzo Church (* 1903; † 1995)

- Amer. Mathematiker und Logiker
- Entwickelte λ-Kalkül (nicht vorgestellt)
- Doktorvater von Stephen Kleene & Alan Turing