Berechenbarkeit und Komplexität
2. Vorlesung
Primitiv-rekursive Funktionen und Hilberts Vermutung

TECHNISCHE UNIVERSITÄT ILMENAU

Prof. Dr. Dietrich Kuske

welt offenes Thüringen

FG Automaten und Logik, TU Ilmenau

Sommersemester 2026

(A^+) Viele Abschlußeigenschaften (vgl. Folie 1.24 f.)

Lemma

Sind $f\colon \mathbb{N}^j \to \mathbb{N}$ und $g_i\colon \mathbb{N}^k \to \mathbb{N}$ für alle $1 \leq i \leq j$ (mit $j, k \geq 0$) loop-berechenbar, so auch die Funktion

$$\mathrm{subst}(f; g_1, \ldots, g_j)\colon \mathbb{N}^k \to \mathbb{N}\colon \overline{n} = (n_1, \ldots, n_k) \mapsto f\big(g_1(\overline{n}), g_2(\overline{n}), \ldots, g_j(\overline{n})\big).$$

Die Abbildung $(f, g_1, g_2, \ldots, g_j) \mapsto \mathrm{subst}(f; g_1, \ldots, g_j)$ wird als Substitution bezeichnet. Sind f und g_i intuitiv berechenbar, so sicher auch $\mathrm{subst}(f; g_1, \ldots, g_j)$. Das Lemma sagt, daß die Klasse der loop-berechenbaren Funktionen unter der Substitution abgeschlossen ist.

Beweis Seien F und G_i Loop-Programme, die die Funktionen f bzw. g_i berechnen (für alle $1 \leq i \leq j$) und dabei höchstens die Variablen $x_1, \ldots, x_\ell$ verwenden. Dann berechnet das Loop-Programm auf der folgenden Folie die Funktion $\mathrm{subst}(f; g_1, \ldots, g_j)$.

$$
\begin{array}{lr}
y_1 := x_1; \ldots; y_k := x_k; & \text{(sichern der Argumente)} \\
G_1; & \\
z_1 := x_1; & \text{(jetzt gilt } z_1 = g_1(x_1, \ldots, x_k)\text{)} \\
x_1 := y_1; \ldots; x_k := y_k; & \text{(rekonstruieren der Argumente)} \\
x_{k+1} := 0; \ldots; x_\ell := 0; & \text{(re-initialisieren der restlichen Variablen)} \\
G_2; & \\
z_2 := x_1; & \text{(jetzt gilt } z_2 = g_2(x_1, \ldots, x_k)\text{)} \\
\vdots & \\
x_1 := y_1; \ldots; x_k := y_k; & \text{(rekonstruieren der Argumente)} \\
x_{k+1} := 0; \ldots; x_\ell := 0; & \text{(re-initialisieren der restlichen Variablen)} \\
G_j; & \\
z_j := x_1; & \text{(jetzt gilt } z_j = g_j(x_1, \ldots, x_k)\text{)} \\
x_1 := z_1; \ldots; x_j := z_j; & \text{(jetzt gilt } x_i = g_i(x_1, \ldots, x_k) \ldots\text{)} \\
x_{j+1} := 0; \ldots; x_\ell := 0; & \text{(\ldots und } x_i = 0 \text{ für die restlichen Variablen)} \\
F & \square
\end{array}
$$

Rekursion

Beispiele

1. Die einzige Funktion $f_1 \colon \mathbb{N} \to \mathbb{N}$ mit $f_1(0) = 1$ und $f_1(m+1) = 2 \cdot f_1(m)$ für alle $m \geq 0$ ist $f_1(x) = 2^x$.
2. Die einzige Funktion $f_2 \colon \mathbb{N} \to \mathbb{N}$ mit $f_2(0) = 1$ und $f_2(m+1) = (m+1) \cdot f_2(m)$ für alle $m \geq 0$ ist $f_2(x) = x!$.
3. Die einzige Funktion $f_3 \colon \mathbb{N}^2 \to \mathbb{N}$ mit $f_3(n,0) = 1$ und $f_3(n, m+1) = n \cdot f_3(n,m)$ für alle $m \geq 0$ (für alle $n \in \mathbb{N}$) ist $f_3(y,x) = y^x$.

Definition

Seien $k \geq 1$, $g\colon \mathbb{N}^{k-1} \to \mathbb{N}$, $h\colon \mathbb{N}^{k+1} \to \mathbb{N}$ und $f\colon \mathbb{N}^k \to \mathbb{N}$. Gelten

$$\begin{aligned} f(n_1, \ldots, n_{k-1}, 0) &= g(n_1, \ldots, n_{k-1}) \text{ und} \\ f(n_1, \ldots, n_{k-1}, m+1) &= h\big(n_1, \ldots, n_{k-1}, m, f(n_1, n_2, \ldots, n_{k-1}, m)\big) \end{aligned}$$

für alle $n_1, \ldots, n_{k-1}, m \in \mathbb{N}$, so entsteht f aus g und h mittels Rekursion. Hierfür schreiben wir $f = \mathrm{rec}(g, h)$.

Anschaulich: bei der Rekursion wird die Definition von $f(\ldots, m+1)$ zurückgeführt auf $f(\ldots, m)$ (und die von $f(\ldots, 0)$ auf $g(\ldots)$). Das bedeutet, daß die Rekursion immer terminiert, d.h. für alle $(k{-}1)$- bzw. $(k{+}1)$-stelligen Funktionen g und h existiert genau eine k-stellige Funktion f mit $f = \mathrm{rec}(g, h)$. Sind g und h intuitiv berechenbar, so sicher auch f. Also sollte auch die Klasse der loop-berechenbaren Funktionen unter Rekursion abgeschlossen sein.

Beispiele (vgl. Folie 2.4)

1. f_1 ist definiert mittels
   - $k = 1$,
   - $g() = 1$ und
   - $h(x_1, x_2) = 2 \cdot x_2$.
2. f_2 ist definiert mittels
   - $k = 1$,
   - $g() = 1$ und
   - $h(x_1, x_2) = (x_1 + 1) \cdot x_2$.
3. f_3 ist definiert mittels
   - $k = 2$,
   - $g(n) = 1$ und
   - $h(x_1, x_2, x_3) = x_1 \cdot x_3$.

Lemma

Sind $k \geq 1$, $g\colon \mathbb{N}^{k-1} \to \mathbb{N}$ und $h\colon \mathbb{N}^{k+1} \to \mathbb{N}$ loop-berechenbar, so auch die Funktion $\mathrm{rec}(g, h)\colon \mathbb{N}^k \to \mathbb{N}$.

Beweis: Die Funktion $\mathrm{rec}(g, h)$ läßt sich durch das folgende (Pseudocode-) Loop-Programm berechnen (wobei wir die Rekursion als Iteration implementieren):

$y := g(x_1, \ldots, x_{k-1});\;\; m := 0;$

`loop` x_k `do`

$\quad y := h(x_1, x_2, \ldots, x_{k-1}, m, y); m := m + 1$

`end`

$x_1 := y$

Unter Verwendung von Loop-Programmen für g und h sowie Zwischenspeicherung und Re-initialisierung ähnlich zum Beweis des Lemmas auf Folie 2.2 kann dieser Pseudocode in ein Loop-Programm für f umgesetzt werden. $\square$

Zwischenbilanz Loop-Programme

Damit haben wir Argumente der Form (K^+) und (A^+) für die Loop-Vermutung vorgebracht. Wir haben also gute Gründe, von ihrer Gültigkeit überzeugt zu sein.

Ein bißchen Geschichte (vgl. Folie 1.22)

Hilbert, der an der Frage interessiert war, was ein „Verfahren“ sei, kannte keine Loop-Programme, sondern formulierte die folgende Vermutung:

Hilberts Vermutung (1926)

Eine Funktion $\mathbb{N}^k \to \mathbb{N}$ mit $k \geq 0$ ist genau dann intuitiv berechenbar, wenn sie primitiv-rekursiv ist.

Wir werden zeigen, daß Hilberts Vermutung äquivalent zur Loop-Vermutung (Folie 1.23) ist (für deren Gültigkeit wir ja gute Argumente haben). Da Hilbert unabhängig von uns auf seine Vermutung kam, wird dies ein Argument der Form (U^+) für die Loop-Vermutung darstellen.

Primitiv-rekursive Funktionen

Loop-Programme sind vereinfachte imperative Programme und stehen für imperative Programmiersprachen, bei denen Programme als Folgen von Befehlen aufgefaßt werden.

Die als nächstes eingeführten rek-Programme sind vereinfachte funktionale Programme und stehen für funktionale Programmiersprachen, bei denen Programme als sich gegenseitig aufrufende Funktionen aufgefaßt werden.

Definition

- $\mathtt{CONST}_0$ ist ein 0-stelliges rek-Programm.
- S ist ein 1-stelliges rek-Programm.
- $\mathtt{PROJ}_i^k$ ist ein k-stelliges rek-Programm (für alle $1 \leq i \leq k$).
- Sind F ein j- und $G_1, \ldots, G_j$ k-stellige rek-Programme ($j, k \geq 0$), so ist $\mathtt{SUBST}(F; G_1, \ldots, G_j)$ ein k-stelliges rek-Programm.
- Sind G ein $(k-1)$- und H ein $(k+1)$-stelliges rek-Programm ($k \geq 1$), so ist $\mathtt{REC}(G, H)$ ein k-stelliges rek-Programm.
- Nichts ist rek-Programm, was sich nicht mittels obiger Regeln erzeugen läßt.

Wir werden jetzt jedem k-stelligen rek-Programm F eine Funktion $[\![F]\!] : \mathbb{N}^k \to \mathbb{N}$ zuordnen:

- Es gelte $[\![\mathtt{CONST}_0]\!]() = 0$. Also ist

  $$[\![\mathtt{CONST}_0]\!] = \mathrm{const}_0 : \mathbb{N}^0 = \{()\} \to \mathbb{N}$$

  die konstante 0-Funktion.
- Für $n \in \mathbb{N}$ sei $[\![S]\!](n) = n + 1$. Also ist $[\![S]\!] : \mathbb{N} \to \mathbb{N}$ die Nachfolgerfunktion.
- Für $\overline{n} = (n_1, \ldots, n_k) \in \mathbb{N}^k$ sei $[\![\mathtt{PROJ}_i^k]\!](\overline{n}) = n_i$. Also ist (vgl. Folie 1.21)

  $$[\![\mathtt{PROJ}_i^k]\!] := \pi_i^k : \mathbb{N}^k \to \mathbb{N}$$

  die Projektionsfunktion der k-Tupel auf den i-ten Eintrag.

- Für $\overline{n} \in \mathbb{N}^k$ sei
  $[\![\mathtt{SUBST}(F; G_1, \ldots, G_j)]\!](\overline{n}) = [\![F]\!]\big([\![G_1]\!](\overline{n}), \ldots, [\![G_j]\!](\overline{n})\big)$.
  Also ist (vgl. Folie 2.2)
  $$[\![\mathtt{SUBST}\big(F; G_1, \ldots, G_j\big)]\!] = \mathrm{subst}\big([\![F]\!]; [\![G_1]\!], \ldots, [\![G_j]\!]\big)$$
  die Substitution der Funktionen $[\![G_1]\!], \ldots, [\![G_j]\!]$ in die Funktion $[\![F]\!]$.
- Für $\overline{n} \in \mathbb{N}^{k-1}$ und $m \in \mathbb{N}$ seien
  $$\begin{aligned} [\![\mathtt{REC}(G, H)]\!](\overline{n}, 0) &= [\![G]\!](\overline{n}) \\ \text{bzw. } [\![\mathtt{REC}(G, H)]\!](\overline{n}, m+1) &= [\![H]\!]\big(\overline{n}, m, [\![\mathtt{REC}(G, H)]\!](\overline{n}, m)\big). \end{aligned}$$
  Also ist (vgl. Folie 2.5)
  $$[\![\mathtt{REC}(G, H)]\!] := \mathrm{rec}\big([\![G]\!], [\![H]\!]\big)$$
  aus den Funktionen $[\![G]\!]$ und $[\![H]\!]$ durch Rekursion entstanden.

Definition

Eine Funktion $f : \mathbb{N}^k \to \mathbb{N}$ ist rek-berechenbar oder primitiv-rekursiv, wenn es ein k-stelliges rek-Programm F gibt mit $[\![F]\!] = f$.

Beispiel

- $\mathtt{CONST}_0$ ist 0- und S ist 1-stelliges rek-Programm
  $\Longrightarrow$ $\mathtt{CONST}_1 := \mathtt{SUBST}(S; \mathtt{CONST}_0)$ ist 0-stelliges rek-Programm mit $[\![\mathtt{CONST}_1]\!] : \mathbb{N}^0 \to \mathbb{N} : () \mapsto 1$

  $\Longrightarrow$ $\mathtt{CONST}_2 := \mathtt{SUBST}(S; \mathtt{CONST}_1)$ ist 0-stelliges rek-Programm mit $[\![\mathtt{CONST}_2]\!] : \mathbb{N}^0 \to \mathbb{N} : () \mapsto 2$

  Iterativ können wir für alle $a \in \mathbb{N}$ also 0-stellige rek-Programme $\mathtt{CONST}_a$ konstruieren mit $[\![\mathtt{CONST}_a]\!]() = a$, d.h. die konstanten 0-stelligen Funktionen const_a sind primitiv-rekursiv.
- Sei $a \in \mathbb{N}$. Dann ist $\mathtt{CONST}_a$ ein 0-stelliges rek-Programm
  $\Longrightarrow$ für alle $k \geq 0$ ist $\mathtt{CONST}_a^k = \mathtt{SUBST}(\mathtt{CONST}_a;)$ ein k-stelliges rek-Programm mit $[\![\mathtt{CONST}_a^k]\!] : \mathbb{N}^k \to \mathbb{N} : \overline{n} \mapsto a$

  D.h. auch die konstanten k-stelligen Funktionen $\mathrm{const}_a^k : \mathbb{N}^k \to \mathbb{N} : \overline{n} \mapsto a$ sind primitiv-rekursiv.

Beispiel

Die Funktion *add*: $\mathbb{N}^2 \to \mathbb{N}: (n_1, n_2) \mapsto n_1 + n_2$ ist primitiv-rekursiv.

Beweis: *add*: $\mathbb{N}^2 \to \mathbb{N}$ ist die einzige Funktion mit

$$add(n_1, 0) = n_1 \text{ und } add(n_1, m+1) = add(n_1, m) + 1\,.$$

Das 2-stellige rek-Programm

$$F = \mathtt{REC}\big(\mathtt{PROJ}^1_1, \mathtt{SUBST}(S; \mathtt{PROJ}^3_3)\big)$$

erfüllt

$$\begin{aligned}
\llbracket F \rrbracket\,(n_1, 0) &= \left\llbracket \mathtt{PROJ}^1_1 \right\rrbracket (n_1) = \pi^1_1(n_1) = n_1 \\
\llbracket F \rrbracket\,(n_1, m+1) &= \left\llbracket \mathtt{SUBST}(S; \mathtt{PROJ}^3_3) \right\rrbracket \big(n_1, m, \llbracket F \rrbracket\,(n_1, m)\big) \\
&= \left\llbracket \mathtt{PROJ}^3_3 \right\rrbracket \big(n_1, m, \llbracket F \rrbracket\,(n_1, m)\big) + 1 \\
&= \llbracket F \rrbracket\,(n_1, m) + 1
\end{aligned}$$

und daher $\llbracket F \rrbracket = add$, d.h. *add* ist primitiv-rekursiv. $\square$

Beispiel

Die Funktion *mult*: $\mathbb{N}^2 \to \mathbb{N}: (n_1, n_2) \mapsto n_1 \cdot n_2$ ist primitiv-rekursiv.

Beweis: *mult*: $\mathbb{N}^2 \to \mathbb{N}$ ist die einzige Funktion mit

$$mult(n_1, 0) = 0 \text{ und } mult(n_1, m+1) = mult(n_1, m) + n_1 .$$

Sei *ADD* ein rek-Programm, das die Funktion *add* berechnet. Sei

$$F = \mathtt{REC}\big(\mathtt{CONST}^1_0, \mathtt{SUBST}(ADD; \mathtt{PROJ}^3_3, \mathtt{PROJ}^3_1)\big) .$$

Es gelten

$$\begin{aligned}
[\![F]\!]\,(n_1, 0) &= [\![\mathtt{CONST}^1_0]\!]\,(n_1) = 0 \\
[\![F]\!]\,(n_1, m+1) &= [\![\mathtt{SUBST}(ADD; \mathtt{PROJ}^3_3, \mathtt{PROJ}^3_1)]\!]\,\big(n_1, m, [\![F]\!]\,(n_1, m)\big) \\
&= [\![\mathtt{PROJ}^3_3]\!]\,\big(n_1, m, [\![F]\!]\,(n_1, m)\big) + \\
&\qquad [\![\mathtt{PROJ}^3_1]\!]\,\big(n_1, m, [\![F]\!]\,(n_1, m)\big) \\
&= [\![F]\!]\,(n_1, m) + n_1
\end{aligned}$$

und daher $[\![F]\!] = mult$, d.h. auch *mult* ist primitiv-rekursiv. □

Beispiel

Die Funktion *dec*: $\mathbb{N} \to \mathbb{N}: n \mapsto n \dot{-} 1$ ist primitiv-rekursiv.

Beweis: $dec\colon \mathbb{N} \to \mathbb{N}$ ist die einzige Funktion mit

$$dec(0) = 0 \text{ und } dec(m+1) = m\,.$$

Das 1-stellige rek-Programm

$$F = \mathtt{REC}(\mathtt{CONST}_0, \mathtt{PROJ}^2_1)$$

erfüllt

$$\begin{aligned} [\![F]\!]\,(0) &= [\![\mathtt{CONST}_0]\!]\,() = 0 \\ [\![F]\!]\,(m+1) &= \left[\!\!\left[\mathtt{PROJ}^2_1\right]\!\!\right]\left(m, [\![F]\!]\,(m)\right) = m \end{aligned}$$

und daher $[\![F]\!] = dec$, d.h. auch *dec* ist primitiv-rekursiv. $\square$

Beispiel

Die Funktion *sub*: $\mathbb{N}^2 \to \mathbb{N} \colon (n_1, n_2) \mapsto n_1 \dot{-} n_2$ ist primitiv-rekursiv.

Beweis: *sub*: $\mathbb{N}^2 \to \mathbb{N}$ ist die einzige Funktion mit

$$sub(n_1, 0) = n_1 \text{ und } sub(n_1, m+1) = sub(n_1, m) \dot{-} 1\,.$$

Sei *DEC* ein rek-Programm, das die Funktion *dec* berechnet. Wir betrachten das 2-stellige rek-Programm

$$F = \mathtt{REC}\Big(\mathtt{PROJ}_1^1, \mathtt{SUBST}(DEC; \mathtt{PROJ}_3^3)\Big)\,.$$

Es gelten

$$\begin{aligned}
[\![F]\!]\,(n_1, 0) &= \left[\!\left[\mathtt{PROJ}_1^1\right]\!\right](n_1) = n_1 \\
[\![F]\!]\,(n_1, m+1) &= \left[\!\left[\mathtt{SUBST}(DEC; \mathtt{PROJ}_3^3)\right]\!\right]\Big(n_1, m, [\![F]\!]\,(n_1, m)\Big) \\
&= \left[\!\left[\mathtt{PROJ}_3^3\right]\!\right]\Big(n_1, m, [\![F]\!]\,(n_1, m)\Big) \dot{-} 1 \\
&= [\![F]\!]\,(n_1, m) \dot{-} 1
\end{aligned}$$

und daher $[\![F]\!] = sub$, d.h. auch *sub* ist primitiv-rekursiv. □

Hilberts Vermutung $\Rightarrow$ Loop-Vermutung

Lemma

Jede primitiv-rekursive Funktion $f\colon \mathbb{N}^k \to \mathbb{N}$ ist loop-berechenbar.

Beweis: Wir zeigen: Für jedes k-stellige rek-Programm F ist $[\![F]\!]\colon \mathbb{N}^k \to \mathbb{N}$ loop-berechenbar.
Dieser Beweis erfolgt durch Induktion über den Aufbau von F.

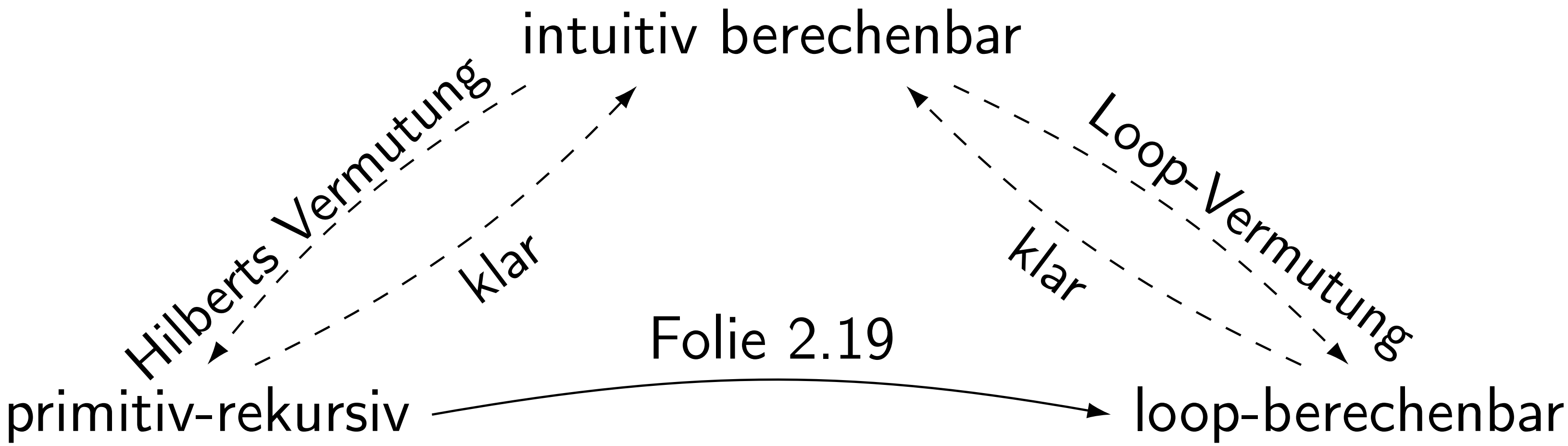
Fall 1: $F \in \{\mathtt{CONST}_0, S, \mathtt{PROJ}_i^k \mid 1 \le i \le k\}$: klar

Fall 2: $F = \mathtt{SUBST}(G; H_1, \dots, H_j)$.
Nach Induktionsvoraussetzung sind $[\![G]\!]$ und $[\![H_i]\!]$ loop-berechenbar.

Nach dem Lemma auf Folie 2.2 ist also $[\![F]\!] = \mathrm{subst}\big([\![G]\!]; [\![H_1]\!], \dots, [\![H_j]\!]\big)$ loop-berechenbar.

Fall 3: $F = \mathtt{REC}(G, H)$.
Nach Induktionsvoraussetzung sind $[\![G]\!]$ und $[\![H]\!]$ loop-berechenbar.

Nach dem Lemma auf Folie 2.7 ist also $[\![F]\!] = \mathrm{rec}\big([\![G]\!], [\![H]\!]\big)$ loop-berechenbar. $\square$



Hilberts Vermutung $\Leftarrow$ Loop-Vermutung

Grundidee und -problem

Übersetze ein Loop-Programm P per Induktion über dessen Aufbau in ein rek-Programm P' mit $[\![P]\!]_k = [\![P']\!]$.

Das ist für kein $k > 1$ möglich, da $[\![P]\!]_k : \mathbb{N}^k \to \mathbb{N}^k$ und $[\![P']\!] : \mathbb{N}^k \to \mathbb{N} \neq \mathbb{N}^k$.

Wunsch/Hoffnung/Traum

„rek“-Programme (analog zu rek-Programmen), die mehrere Ausgaben produzieren können, also z.B. Funktionen $\mathbb{N}^k \to \mathbb{N}^k$ berechnen.

Übersetzung Loop-Programme ⇒ „rek"-Programme

Sei $\overline{n} = (n_1, \ldots, n_k) \in \mathbb{N}^k$. Dann gilt

$$\begin{aligned}
\llbracket x_i := c \rrbracket_k (\overline{n}) &= \big(n_1, \ldots, n_{i-1}, c, n_{i+1}, \ldots, n_k\big) \\
&= \big(\pi^k_1(\overline{n}), \ldots, \pi^k_{i-1}(\overline{n}), \llbracket \mathtt{CONST}^k_c \rrbracket (\overline{n}), \pi^k_{i+1}(\overline{n}), \ldots, \pi^k_k(\overline{n})\big)
\end{aligned}$$

Ist $k = 1$, so ist $\llbracket x_i := c \rrbracket_k : \mathbb{N}^k \to \mathbb{N}^k$ also eine primitiv-rekursive Funktion.

Dies gilt analog für die Funktionen $\llbracket x_i := x_j \pm c \rrbracket_k$ wegen

$$\begin{aligned}
\llbracket x_i := x_j + c \rrbracket_k (\overline{n}) &= \Big(\ldots, \mathit{add}\big(\pi^k_j(\overline{n}), \mathrm{const}^k_c(\overline{n})\big), \ldots\Big) \\
&= \Big(\ldots, \mathrm{subst}\big(\mathit{add}; \pi^k_j, \mathrm{const}^k_c\big)(\overline{n}), \ldots\Big) \\
\text{und } \llbracket x_i := x_j \dot{-} c \rrbracket_k (\overline{n}) &= \Big(\ldots, \mathrm{subst}\big(\mathit{sub}; \pi^k_j, \mathrm{const}^k_c\big)(\overline{n}), \ldots\Big)
\end{aligned}$$

Sei wieder $\overline{n} = (n_1, \ldots, n_k) \in \mathbb{N}^k$. Dann gilt

$$\begin{array}{rcl} [\![P;Q]\!]_k\,(\overline{n}) & = & [\![Q]\!]_k\left([\![P]\!]_k\,(\overline{n})\right) \\ \text{und damit } [\![P;Q]\!]_k & = & \text{‘subst‘}\left([\![Q]\!]_k\,;[\![P]\!]_k\right) \end{array}$$

Gilt $k = 1$ und sind die Funktionen $[\![P]\!]_k$ und $[\![Q]\!]_k$ primitiv-rekursiv, so also auch die Funktion $[\![P;Q]\!]_k$.

Sei jetzt P das Programm `loop` x_i `do` Q `end` und $k' = k + 1$.
Wir wollen die Funktion

$$f\colon \mathbb{N}^{k'} \to \mathbb{N}^k\colon (\overline{n}, m) \mapsto \left(\llbracket Q \rrbracket_k\right)^m (\overline{n})$$

mittels Rekursion beschreiben, denn mit f kann $\llbracket P \rrbracket_k$ ausgedrückt werden:

$$\llbracket P \rrbracket_k (\overline{n}) = f\left(\overline{n}, \pi_i^k(\overline{n})\right), \text{ also } \llbracket P \rrbracket_k = \text{`subst`}(f; \pi_1^k, \ldots, \pi_k^k, \pi_i^k).$$

Dazu definieren wir Funktionen $g\colon \mathbb{N}^{k'-1} \to \mathbb{N}^k\colon \overline{n} \mapsto \overline{n}$ und
$h\colon \mathbb{N}^{k'+k} \to \mathbb{N}^k\colon (m_1, \ldots, m_{k'+k}) \mapsto \llbracket Q \rrbracket_k (m_{k'+1}, \ldots, m_{k'+k})$.
Dann gelten

$$\begin{aligned}
f(\overline{n}, 0) &= \overline{n} = g(\overline{n}) \\
f(\overline{n}, m+1) &= \left(\llbracket Q \rrbracket_k\right)^{m+1}(\overline{n}) = \llbracket Q \rrbracket_k \left(\left(\llbracket Q \rrbracket_k\right)^m (n)\right) \\
&= \llbracket Q \rrbracket_k \left(f(\overline{n}, m)\right) = h\left(\overline{n}, m, f(\overline{n}, m)\right), \\
\text{und damit } f &= \text{`rec`}(g, h).
\end{aligned}$$

Ist $k = 1$ und $\llbracket Q \rrbracket_k$ primitiv-rekursiv, so also auch $\llbracket P \rrbracket_k$.

- Für Loop-Programme, die nur die Variable x_1 verwenden, haben wir also die Übersetzung in rek-Programme ☺

  ... aber dies ist viel zu speziell ☹
- Wenn man nur $\overline{n}$ als eine natürliche Zahl auffassen könnte ...

Satz

Sei $k \geq 1$.

- Es gibt eine primitiv-rekursive Bijektion $\langle \ldots \rangle \colon \mathbb{N}^k \to \mathbb{N}$.
- Es gibt primitiv-rekursive Funktionen $d_1, \ldots, d_k \colon \mathbb{N} \to \mathbb{N}$, so daß für alle $1 \leq i \leq k$ und alle $n_1, \ldots, n_k \in \mathbb{N}$

$$d_i\big(\langle n_1, \ldots, n_k \rangle\big) = n_i \text{ gilt}.$$

d.h. $\overline{n}$ kann tatsächlich als die natürliche Zahl $n = \langle \overline{n} \rangle$ aufgefaßt werden, und die Einträge von $\overline{n}$ können aus der „Kodierung“ n berechnet werden.

Beweis siehe Zusatzmaterial Folie 2.35 ff. □

Übersetzung Loop-Programme $\Rightarrow$ rek-Programme

Die Strategie

Sei $f\colon \mathbb{N}^j \to \mathbb{N}$ loop-berechenbar. Dann existieren $k \geq j$ und ein Loop-Programm P, das höchstens die Variablen $x_1, \ldots, x_k$ verwendet, mit $f(\overline{m}) = \pi_1^k\big(\llbracket P \rrbracket_k(\overline{m}, 0, \ldots, 0)\big)$ für alle $\overline{m} \in \mathbb{N}^j$.

Zunächst definieren wir eine abgewandelte Semantik von P:

$$\langle\!\langle P \rangle\!\rangle_k\colon \mathbb{N} \to \mathbb{N}\colon n \mapsto \Big\langle \llbracket P \rrbracket_k\big(d_1(n), \ldots, d_k(n)\big) \Big\rangle$$

D.h. die Funktion $\langle\!\langle P \rangle\!\rangle_k$ behandelt eine Zahl n wie folgt:

1. Wandle n in das Tupel $\overline{n} \in \mathbb{N}^k$ mit $\langle \overline{n} \rangle = n$ um!
2. Wende das Programm P auf $\overline{n}$ an!
3. Wandle Tupel der erhaltenen Variablenwerten mittels $\langle \ldots \rangle$ in natürliche Zahl um!

Mit anderen Worten: Die Funktion $\langle\!\langle P \rangle\!\rangle_k$ simuliert die Funktion $[\![P]\!]_k$, wobei nicht mit dem Tupel der Variablenwerte $(n_1, \dots, n_k)$, sondern mit dessen Kodierung $\langle n_1, \dots, n_k \rangle$ gerechnet wird.

Im Ergebnis gilt für alle $\overline{m} \in \mathbb{N}^j$ insbesondere

$$f(\overline{m}) = \pi_1^k\big([\![P]\!]_k (\overline{m}, 0, \dots, 0)\big) = d_1\Big(\langle\!\langle P \rangle\!\rangle_k\big(\langle \overline{m}, 0, \dots, 0 \rangle\big)\Big).$$

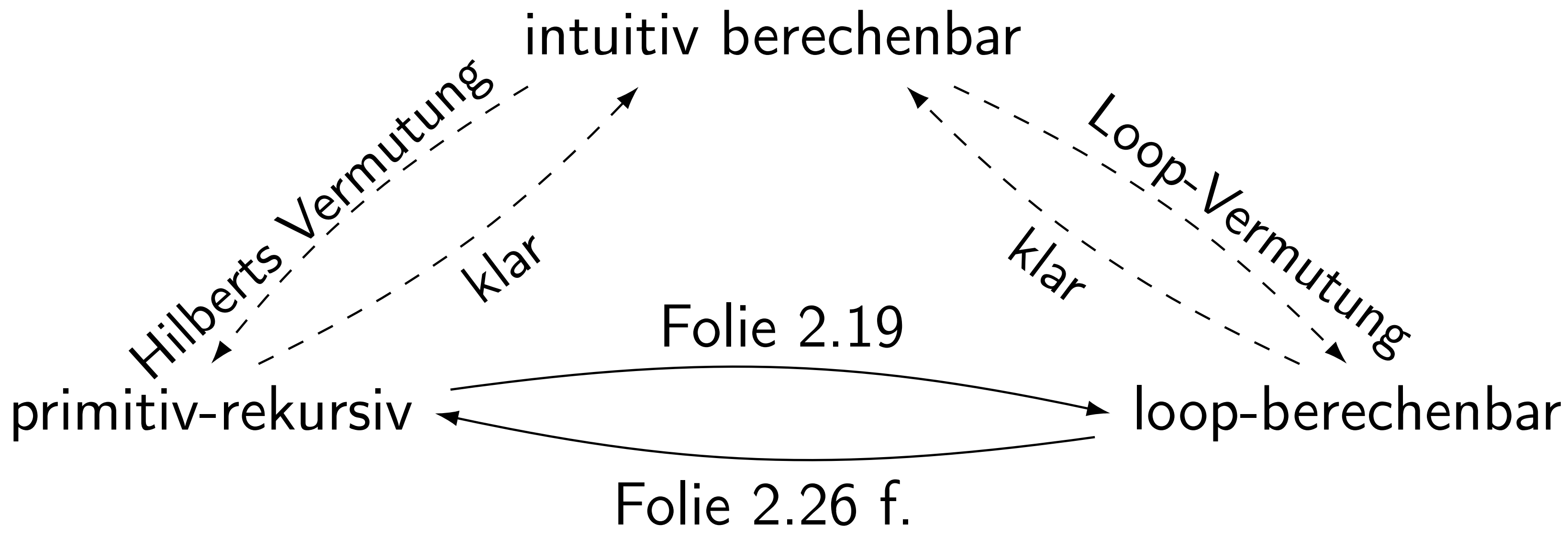
Ist $\langle\!\langle P \rangle\!\rangle_k$ primitiv-rekursiv, so also auch f. Und dies ist tatsächlich der Fall:

Satz

Sei P Loop-Programm, das höchstens die Variablen $x_1, \dots, x_k$ verwendet. Dann ist $\langle\!\langle P \rangle\!\rangle_k \colon \mathbb{N} \to \mathbb{N}$ primitiv-rekursiv.

Beweis ersetze auf den Folie 2.22 ff. $\overline{n}$ durch n und $[\![\;]\!]_k$ durch $\langle\!\langle \; \rangle\!\rangle_k$, siehe Zusatzmaterial Folie 2.32 ff. $\square$

Zwischenbilanz



(U$^+$) zwei äquivalente Vorschläge, den Begriff „intuitiv berechenbar“ zu formalisieren: loop-berechenbar und primitiv-rekursiv

(K$^+$) viele Funktionen sind loop-berechenbar bzw. primitiv-rekursiv

(A$^+$) die beiden Klassen erfüllen viele Abschlußeigenschaften

Damit haben wir gute Gründe, der Loop-Vermutung (Folie 1.23) bzw. der Hilbertschen Vermutung (Folie 2.9) zuzustimmen.

Beispielhafte Testfragen aus dieser Vorlesung (andere sind möglich):

- Wann ist eine Funktion rek-berechenbar?
- Was besagt Hilberts Vermutung?

Zusammenfassung 2. Vorlesung

in dieser Vorlesung neu

- viele Argumente der Form (A^+) für die Loop-Vermutung
- Hilberts Vermutung: intuitiv berechenbar $=$ primitiv-rekursiv
- Argument der Form (U^+) für die Loop-Vermutung: loop-berechenbar $=$ primitiv-rekursiv

kommende Vorlesung

- zwei große Enttäuschungen als Motivation für weitere Überlegungen:
- wir müssen auch partielle Funktionen betrachten

Zusatzmaterial

Beweis des Satzes auf Folie 2.27

Der Beweis erfolgt induktiv über die Konstruktion des Loop-Programms P.

Sei $n \in \mathbb{N}$. Dann gilt

$$\langle\!\langle x_i := c \rangle\!\rangle_k(n) \quad = \quad \Big\langle d_1(n), \ldots, d_{i-1}(n), \mathrm{const}_c^1(n), d_{i+1}(n), \ldots, d_k(n) \Big\rangle$$

und damit

$$\langle\!\langle x_i := c \rangle\!\rangle_k \quad = \quad \mathrm{subst}\Big(\langle \ldots \rangle; d_1, \ldots, d_{i-1}, \mathrm{const}_c^1, d_{i+1}, \ldots, d_k\Big).$$

Die Funktion $\langle\!\langle x_i := c \rangle\!\rangle_k$ erhält man also durch Substitution der primitiv-rekursiven Funktionen d_j und const_c^1 in die primitiv-rekursive Funktion $\langle \ldots \rangle$, sie ist also primitiv-rekursiv.

Dies gilt analog für die Funktionen $\langle\!\langle x_i := x_j \pm c \rangle\!\rangle_k$ wegen

$$
\begin{array}{rcl}
\langle\!\langle x_i := x_j + c \rangle\!\rangle_k(n) & = & \Big\langle \ldots, \mathrm{subst}\big(add; d_j, \mathrm{const}_c^1\big)(n), \ldots \Big\rangle \\
\text{und } \langle\!\langle x_i := x_j \dot{-} c \rangle\!\rangle_k(n) & = & \Big\langle \ldots, \mathrm{subst}\big(sub; d_j, \mathrm{const}_c^1\big)(n), \ldots \Big\rangle
\end{array}
$$

Sei wieder $n \in \mathbb{N}$. Dann gilt

$$
\begin{array}{rcl}
\langle\!\langle P; Q \rangle\!\rangle_k(n) & = & \langle\!\langle Q \rangle\!\rangle_k\big(\langle\!\langle P \rangle\!\rangle_k(n)\big) \\
\text{und damit } \langle\!\langle P; Q \rangle\!\rangle_k & = & \mathrm{subst}\big(\langle\!\langle Q \rangle\!\rangle_k; \langle\!\langle P \rangle\!\rangle_k\big)
\end{array}
$$

Sind $\langle\!\langle P \rangle\!\rangle_k$ und $\langle\!\langle Q \rangle\!\rangle_k$ primitiv-rekursiv, so also auch $\langle\!\langle P; Q \rangle\!\rangle_k$.

Sei jetzt P das Programm `loop` x_i `do` Q `end`.
Wir wollen die Funktion

$$f\colon \mathbb{N}^2 \to \mathbb{N}\colon (n,m) \mapsto \big(\langle\!\langle Q \rangle\!\rangle_k\big)^m(n)$$

mittels Rekursion beschreiben, denn mit f kann $\langle\!\langle P \rangle\!\rangle_k$ ausgedrückt werden:

$$\langle\!\langle P \rangle\!\rangle_k(n) = \big(\langle\!\langle Q \rangle\!\rangle_k\big)^{d_i(n)}(n) = f\big(n, d_i(n)\big)\,,\ \text{also } \langle\!\langle P \rangle\!\rangle_k = \mathrm{subst}(f; \pi^1_1, d_i)\,.$$

Dazu definieren wir Funktionen $g\colon \mathbb{N} \to \mathbb{N}\colon n \mapsto n$ und
$h\colon \mathbb{N}^3 \to \mathbb{N}\colon (m_1, m_2, m_3) \mapsto \langle\!\langle Q \rangle\!\rangle_k(m_3)$. Dann gelten

$$\begin{aligned}
f(n,0) &= n = g(n)\\
f(n,m+1) &= \big(\langle\!\langle Q \rangle\!\rangle_k\big)^{m+1}(n) = \langle\!\langle Q \rangle\!\rangle_k\Big(\big(\langle\!\langle Q \rangle\!\rangle_k\big)^m(n)\Big)\\
&= \langle\!\langle Q \rangle\!\rangle_k\big(f(n,m)\big) = h\big(n,m,f(n,m)\big)\,,\\
\text{und damit } f &= \mathrm{rec}(g,h)\,.
\end{aligned}$$

Ist $\langle\!\langle Q \rangle\!\rangle_k$ primitiv-rekursiv, so also auch $\langle\!\langle P \rangle\!\rangle_k$. □

Beweis des Satzes auf Folie 2.25 - die umfangreiche Vorbereitung

Beispiel

Für $m, n \in \mathbb{N}$ sei $\mathrm{kg}(m, n) = \begin{cases} 1 & \text{falls } m \leq n \\ 0 & \text{falls } m > n\,. \end{cases}$

Dann gilt

$$\mathrm{kg}(m, n) = 1 \dot{-} (m \dot{-} n)$$

und damit

$$\mathrm{kg} = [\![\mathtt{SUBST}(SUB; \mathtt{CONST}_1^2, SUB)]\!]$$

Die Funktion $\mathrm{kg}\colon \mathbb{N}^2 \to \{0, 1\}$ ist also primitiv-rekursiv.

Definition

Seien $k \geq 0$ und $f, g \colon \mathbb{N}^{k+1} \to \mathbb{N}$ mit

$$g(x_1, \ldots, x_{k+1}) = \begin{cases} \min\left\{x \leq x_{k+1} \mid f(x_1, \ldots, x_k, x) = 0\right\} & \\ & \text{falls diese Menge nicht leer ist} \\ 0 & \text{sonst} \end{cases}$$

Dann sagen wir „g geht durch den beschränkten min-Operator aus f hervor“.

Die Funktion g bestimmt die kleinste Nullstelle von f. I.a. muß f aber keine Nullstelle haben, die „Suche“ nach der kleinsten Nullstelle würde also nicht terminieren. Um dieses Problem zu umgehen, bestimmt g

- die kleinste Nullstelle von f, falls diese existiert und höchstens dem letzten Argument von g ist,
- bzw. liefert im anderen Fall 0.

Lemma

Seien $f, g\colon \mathbb{N}^{k+1} \to \mathbb{N}$ Funktionen, so daß g durch den beschränkten min-Operator aus f hervorgeht. Ist f primitiv-rekursiv, so auch g.

Beweis: In diesem Beweis sei $\overline{n} = (n_1, \ldots, n_k) \in \mathbb{N}^k$ ein beliebiges k-Tupel natürlicher Zahlen. Wir zeigen zunächst, daß die folgende Hilfsfunktion $g'\colon \mathbb{N}^{k+1} \to \mathbb{N}$ primitiv-rekursiv ist:

$$\begin{aligned} g'(\overline{n}, m) &= \min\Big(\big\{i \le m \,\big|\, f(\overline{n}, i) = 0\big\} \cup \{m+1\}\Big) \\ &= \begin{cases} \min\big\{i \le m \,\big|\, f(\overline{n}, i) = 0\big\} & \text{falls diese Menge } \neq \emptyset \\ m+1 & \text{sonst} \end{cases} \end{aligned}$$

Es gelten

$$
\begin{aligned}
g'(\overline{n},0) &= \begin{cases} 1 & \text{falls } 1 \le f(\overline{n},0) \\ 0 & \text{falls } 1 > f(\overline{n},0) \end{cases} \\
&= \mathrm{kg}\big(1, f(\overline{n},0)\big) \\
g'(\overline{n},m+1) &= \begin{cases} g'(\overline{n},m) & \text{falls } g'(\overline{n},m) \le m \\ m+1 & \text{falls } g'(\overline{n},m) \ge m+1 \text{ und } f(\overline{n},m+1) \le 0 \\ m+2 & \text{falls } g'(\overline{n},m) \ge m+1 \text{ und } f(\overline{n},m+1) \ge 1 \end{cases} \\
&= g'(\overline{n},m) \cdot \mathrm{kg}\big(g'(\overline{n},m), m\big) \\
&\quad + (m+1) \cdot \mathrm{kg}\big(m+1, g'(\overline{n},m)\big) \cdot \mathrm{kg}\big(f(\overline{n},m+1), 0\big) \\
&\quad + (m+2) \cdot \mathrm{kg}\big(m+1, g'(\overline{n},m)\big) \cdot \mathrm{kg}\big(1, f(\overline{n},m+1)\big)
\end{aligned}
$$

Also geht g' mittels Rekursion aus primitiv-rekursiven Funktionen hervor, ist also selbst primitiv-rekursiv.

Wegen $g(\overline{n},m) = g'(\overline{n},m) \cdot \mathrm{kg}\big(g'(\overline{n},m), m\big)$ ist auch g primitiv-rekursiv. $\square$

Definition

Seien $f, h\colon \mathbb{N}^{k+1} \to \mathbb{N}$ Funktionen mit

$$h(\overline{n}, m) = \begin{cases} 1 & \text{falls } \exists i \leq m\colon f(\overline{n}, i) \geq 1 \\ 0 & \text{sonst} \end{cases}$$

für alle $\overline{n} \in \mathbb{N}^k$. Wir sagen, h geht durch den beschränkten Existenzquantor aus f hervor.

Lemma

Ist $f\colon \mathbb{N}^{k+1} \to \mathbb{N}$ eine primitiv-rekursive Funktion und geht h durch den beschränkten Existenzquantor aus f hervor, so ist auch h primitiv-rekursiv.

Beweis: Die Hilfsfunktion $f'\colon \mathbb{N}^{k+1} \to \mathbb{N}\colon (\overline{n}, m) \mapsto 1 \dot{-} f(m, \overline{n})$ ist primitiv-rekursiv. Gehe $g'\colon \mathbb{N}^{k+1} \to \mathbb{N}$ aus f' durch den beschränkten min-Operator hervor. Dann ist auch g' primitiv-rekursiv und es gilt

$$\begin{aligned} h(\overline{n}, m) = 1 \quad &\Longleftrightarrow \quad \exists i \leq m\colon f(\overline{n}, i) \geq 1 \\ &\Longleftrightarrow \quad \exists i \leq m\colon f'(\overline{n}, i) = 0 \\ &\Longleftrightarrow \quad f'\big(\overline{n}, g'(\overline{n}, m)\big) \leq 0 \\ &\Longleftrightarrow \quad \mathrm{kg}\Big(f'\big(\overline{n}, g'(\overline{n}, m)\big), 0\Big) = 1\,, \end{aligned}$$

d.h. $h(\overline{n}, m) = \mathrm{kg}\Big(f'\big(\overline{n}, g'(\overline{n}, m)\big), 0\Big)$. Also ist auch h primitiv-rekursiv. $\square$

Beispiel

Die Funktion $\binom{\cdot}{2}\colon \mathbb{N} \to \mathbb{N}\colon n \mapsto \binom{n}{2}$ ist primitiv-rekursiv.

Beweis: Es gelten

$$\begin{aligned} \binom{0}{2} &= 0 \\ \text{und } \binom{m+1}{2} &= \binom{m}{2} + m\,. \end{aligned}$$

Also geht $\binom{\cdot}{2}$ mittels Rekursion aus den Funktionen const_0^0 und Addition hervor. Da diese primitiv-rekursiv sind, ist auch $\binom{\cdot}{2}$ primitiv-rekursiv. □

Lemma

Die Funktion

$$c\colon \mathbb{N}^2 \to \mathbb{N}\colon (m,n) \mapsto m + \binom{m+n+1}{2}$$

ist eine primitiv-rekursive Bijektion.

Beweisidee: Die Funktion c entsteht durch Substitution der primitiv-rekursiven Funktionen Addition und $\binom{n}{2}$ und ist daher ebenfalls primitiv-rekursiv.

Betrachtet man die Wertetabelle von c, so ist einsichtig, daß sie bijektiv ist:

	$m = 0$	1	2	3	4
$n = 0$	0	2	5	9	14
1	1	4	8	13	19
2	3	7	12	18	25
3	6	11	17	24	32
4	10	16	23	31	40

$\square$

Lemma

Es seien $\ell \colon \mathbb{N} \to \mathbb{N}$ und $r \colon \mathbb{N} \to \mathbb{N}$ die durch

$$\forall m, n \in \mathbb{N} \colon \ell\big(c(m,n)\big) = m \text{ und } r\big(c(m,n)\big) = n$$

eindeutig gegebenen Funktionen. Dann sind ℓ und r primitiv-rekursiv.

Beweis: Betrachte die Funktion $C \colon \mathbb{N}^3 \to \{0,1\}$ mit $C(x,y,z) = 1$ gdw. $x = c(y,z)$, d.h. $C(x,y,z) = \mathrm{kg}\big(x, c(y,z)\big) \cdot \mathrm{kg}\big(c(y,z), x\big)$, die Funktion C ist also primitiv-rekursiv.
Deshalb sind auch die folgenden Funktionen ℓ' und r' primitiv-rekursiv (vgl. Folien 2.37 und 2.40):

$$\begin{aligned}
\ell'(n, m_1, m_2) &= \min\big\{y \,\big|\, y = m_2 + 1 \text{ oder } \exists z \le m_1 \colon C(n,y,z) = 1\big\} \\
r'(n, m_1, m_2) &= \min\big\{z \,\big|\, z = m_2 + 1 \text{ oder } \exists y \le m_1 \colon C(n,y,z) = 1\big\}
\end{aligned}$$

Schließlich gilt $\ell(n) = \ell'(n,n,n)$ und $r(n) = r'(n,n,n)$. $\square$

Beweis des Satzes auf Folie 2.25

Die Funktion c kann verwendet werden, um k-Tupel natürlicher Zahlen durch eine Zahl zu kodieren:

$$\langle n_1, n_2, \ldots, n_k \rangle = c(n_1, c(n_2, c(\ldots, c(n_k, 0) \ldots)))$$

Die Funktion $(n_1, n_2, \ldots, n_k) \mapsto \langle n_1, n_2, \ldots, n_k \rangle$ ist dann auch eine primitiv-rekursive Bijektion.

Mit den Funktionen ℓ und r lassen sich dann auch primitiv-rekursive Dekodierfunktionen für kodierte k-Tupel definieren, die $d_i\big(\langle n_1, n_2, \ldots, n_k \rangle\big) = n_i$ erfüllen:

$$\begin{aligned} d_1(n) &= \ell(n) \\ d_2(n) &= \ell\big(r(n)\big) \\ &\vdots \\ d_k(n) &= \ell\big(r^{k-1}(n)\big) \end{aligned}$$

$\square$