6.2 μ-Rekursive Funktionen $\mathcal{R}_p(\mathbb{N})$ (partiell rekursive Funktionen)

- Primitiv rekursive Funktionen $\mathcal{P}(\mathbb{N})$
  Die bisher betrachteten Operationen auf Funktionen bilden totale Funktionen wieder in totalen Funktionen ab.
  z. B. Komposition, primitive Rekursion, Fallunterscheidung, beschränkte Minimierung, Wertverlaufsrekursion.
- Diagonalisierung liefert für jede effektiv aufzählbare Menge von totalen Funktionen eine Diagonalfunktion d, die total, effektiv und nicht in der Menge liegt.
  Will man alle effektiv berechenbaren Funktionen charakterisieren, so benötigt man partielle Funktionen.

Welche Operationen führen zu partiellen Funktionen?

Vergleichbar dazu: While Konstrukt der Programmiersprache.

Idee: Unbeschränkte Minimierung: Suchen nach „erster Nullstelle“.

Hat eine Funktion keine Nullstelle, so ist eine solche Suche nicht erfolgreich sein und führt somit zur Partialität.

Minimierung

6.37 Definition Minimierungsoperator

Sei $f : \mathbb{N}^{n+1} \to \mathbb{N}$ eine Funktion $n \geq 1$.

$g : \mathbb{N}^n \to \mathbb{N}$ entsteht aus f durch **Minimierung**, falls gilt

$$g(\vec{x}) \downarrow \text{ gdw } \exists y \ (f(\vec{x}, y) \downarrow \wedge \ f(\vec{x}, y) = 0 \wedge \\ \forall z \ (z < y \to (f(\vec{x}, z) \downarrow \wedge \ f(\vec{x}, z) > 0)))$$

In diesem Fall ist $g(\vec{x})$ als das eindeutig bestimmte y definiert.

Schreibe: $g(\vec{x}) = \mu y.f(\vec{x}, y) = 0$,
(kleinste y, so dass $f(\vec{x}, y)$ null ist)

6.38 Beispiel

- $+ : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$
  $g(x) = \mu y.(x + y = 0) = \begin{cases} 0 & x = 0 \\ \uparrow & \text{sonst} \end{cases}$
- $* : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$
  $g(x) = \mu y.(x * y = 0) = 0$

Beachte:

Die Minimierung wird auf Funktionen mit Stelligkeit ≥ 2 angewendet.

Offenbar ist $\mathcal{P}(\mathbb{N})$ **nicht** abgeschlossen gegen Minimierung
(siehe $\mu y.x + y = 0$).

μ-rekursive Ausdrücke und Funktionen: Die Klasse $\mathcal{R}_p(\mathbb{N})$

6.39 Definition Erweiterung der Ausdrücke

- **Syntax: μ-rekursive Ausdrücke** entstehen durch Hinzunahme der Regel

$$\frac{G}{MIN(G)}$$

zum Kalkül der primitiv rekursiven Ausdrücke.

- **Semantik:** Jeder μ-rekursive Ausdruck π repräsentiert für beliebige Stelligkeit $n \geq 1$ eine Funktion $f_\pi^{(n)} : \mathbb{N}^n \longrightarrow \mathbb{N}$ durch:
$f_{MIN(G)}^{(n)}(x_1, \ldots, x_n) = \mu y. f_G^{(n+1)}(x_1, \ldots, x_n, y) = 0$, falls $\pi = MIN(G)$.
Sonst bleibt $f_\pi^{(n)}$ wie im primitiv rekursiven Fall unter Beobachtung, dass nun partielle Funktionen vorkommen dürfen, d. h. ist beim rekursiven Auswerten eine Teilfunktion an der betrachteten Stelle nicht definiert, so ist auch die gesamte Funktion an der betrachteten Stelle nicht definiert.
- Eine Funktion $f : \mathbb{N}^n \longrightarrow \mathbb{N}$ heißt **μ-rekursiv** (oder **partiell rekursiv**), falls $f = f_\pi^{(n)}$ für einen μ-rekursiven Ausdruck π gilt.

Sei $\mathcal{R}_p(\mathbb{N})$ die Menge der partiell rekursiven Funktionen.

Beispiel

6.40 Beispiel

Sei $\pi = REK(SUCC, MIN(PROJ(1)))$.

Bestimme $f_\pi^{(2)} : \mathbb{N}^2 \to \mathbb{N}$.

$$
\begin{aligned}
f_\pi^{(2)}(x,0) &= f_{SUCC}^{(2)}(x,0) = x+1 \\
f_\pi^{(2)}(x,y+1) &= f_{MIN(PROJ(1))}^{(3)}(x, f_\pi^{(2)}(x,y), y) \\
&= \mu z . f_{PROJ(1)}^{(4)}(x, f_\pi^{(2)}(x,y), y, z) = 0 \\
&= \begin{cases} 0 & x = 0 \\ \uparrow & \text{sonst} \end{cases} \\
f_\pi^{(2)}(x,y) &= \begin{cases} x+1 & \text{falls } y = 0 \\ 0 & \text{falls } x = 0 \wedge y > 0 \\ \uparrow & \text{sonst} \end{cases}
\end{aligned}
$$

6.41 Satz

Die Klasse der μ-rekursiven Funktionen enthält die Grundfunktionen und ist abgeschlossen gegenüber Komposition, primitiver Rekursion und Minimierung. Sie ist die kleinste Klasse von Funktionen mit dieser Eigenschaft.

Es gilt $\mathcal{P}(\mathbb{N}) \subsetneq \mathcal{R}_p(\mathbb{N})$.

Beweisprinzip für Eigenschaften μ-rekursiver Funktionen:

- Eigenschaft E gilt für die Grundfunktionen.
- E bleibt erhalten bei Komposition, primitiver Rekursion, Minimierung.

Entscheidbare Mengen
Abschlusseigenschaften

6.42 Definition

Eine Relation $R \subset \mathbb{N}^n$ heißt **(rekursiv-) entscheidbar,** falls ihre charakteristische Funktion $\chi_R : \mathbb{N}^n \longrightarrow \mathbb{N}$ μ-rekursiv ist, d. h.

$$\chi_R \in \mathcal{R}_p(\mathbb{N})$$

Beachte: Jede primitiv rekursive Relation ist entscheidbar.

Es gelten für die μ-rekursiven Funktionen und entscheidbaren Relationen zum primitiv rekursiven Fall analoge Abschlusseigenschaften.

Insbesondere:

6.43 Lemma Reduzierbarkeit

Ist $S \subseteq \mathbb{N}^n$ entscheidbar und sind $f_1, \ldots, f_n : \mathbb{N}^m \longrightarrow \mathbb{N}$ **totale** μ-rekursive Funktionen (z.B. wenn die f_i primitiv rekursiv sind).

Dann ist auch $R \subseteq \mathbb{N}^m$ mit
$R\vec{x} \;\text{ gdw }\; Sf_1(\vec{x}) \ldots f_n(\vec{x})$ entscheidbar.

6.44 Beispiel

Sei f eine totale μ-rekursive Funktion, dann ist ihr Graph entscheidbar.

Beweis:

$$\begin{aligned} R(\vec{x}) &\text{ gdw } (\vec{x}, y) \in graph(f) \\ &\text{ gdw } f(\vec{x}) = y \end{aligned}$$

Lemma mit $f_1(\vec{x}, y) = f(\vec{x}), f_2(\vec{x}, y) = y, S \equiv \;\; =$.

Fallunterscheidung mit entscheidbaren Relationen

6.45 Lemma Fallunterscheidung

Seien $R_1, \ldots, R_m$ paarweise disjunkte entscheidbare Relationen und $h_1, \ldots, h_{m+1}$ μ-rekursive Funktionen auf $\mathbb{N}^n$. Dann ist die Funktion $f : \mathbb{N}^n \longrightarrow \mathbb{N}$ mit

$$f(\vec{x}) = \begin{cases} h_1(\vec{x}) & \text{falls } R_1\vec{x} \\ \vdots & \\ h_m(\vec{x}) & \text{falls } R_m\vec{x} \\ h_{m+1}(\vec{x}) & \text{sonst} \end{cases}$$

μ-rekursiv.

Sind die Funktionen h_i auf R_i definiert und ist h_{m+1} auf $\mathbb{N}^n \setminus \bigcup_{1}^{m} R_i$ definiert, so ist f total.

Beweis:

Der Beweis geht nicht wie im primitiv rekursiven Fall!

Damals:

$$f(\vec{x}) = \chi_{R_1}(\vec{x}) \cdot h_1(\vec{x}) + \cdots + \chi_{R_m}(x) \cdot h_m(\vec{x}) + (1 - (\chi_{R_1 \vee \cdots \vee R_m}(\vec{x})) \cdot h_{m+1}(\vec{x})$$

Diese Gleichung gilt mit partiellen Funktionen h_i nicht!

Fallunterscheidung (Fort.)- Weitere Abschlusseigenschaften

Definiere μ-rekursive Hilfsfunktionen H_i für $1 \leq i \leq m+1$ auf $\mathbb{N}^{n+1}$ durch

$$H_i(\vec{x}, 0) = 0 \qquad H_i(\vec{x}, y+1) = h_i(\vec{x})$$

Dann gilt $H_i \in \mathcal{R}_p(\mathbb{N})$ und

$$f(\vec{x}) = H_1(\vec{x}, \chi_{R_1}(\vec{x})) + \cdots + H_m(\vec{x}, \chi_{R_m}(\vec{x})) + H_{m+1}(\vec{x}, 1 - (\chi_{R_1} + \cdots + \chi_{R_m})(\vec{x}))$$

(Beachte $H_i(\vec{x}, y)$ ist für $y > 0$ definiert gdw $h_i(\vec{x})$ definiert ist).

6.46 Lemma

- Die entscheidbaren Relationen sind abgeschlossen gegen $\neg$, $\wedge$, $\vee$ und beschränkte Quantifizierung.
- Die Klasse $\mathcal{R}_p(\mathbb{N})$ ist abgeschlossen gegen beschränkte und unbeschränkte Minimierung mit Relationen. Ist $R \subseteq \mathbb{N}$ entscheidbar, so ist die Funktion

  $$f(\vec{x}, b) = \mu y \leq b.R\vec{x}y$$

  und

  $$g(\vec{x}) = \mu y.R\vec{x}y$$

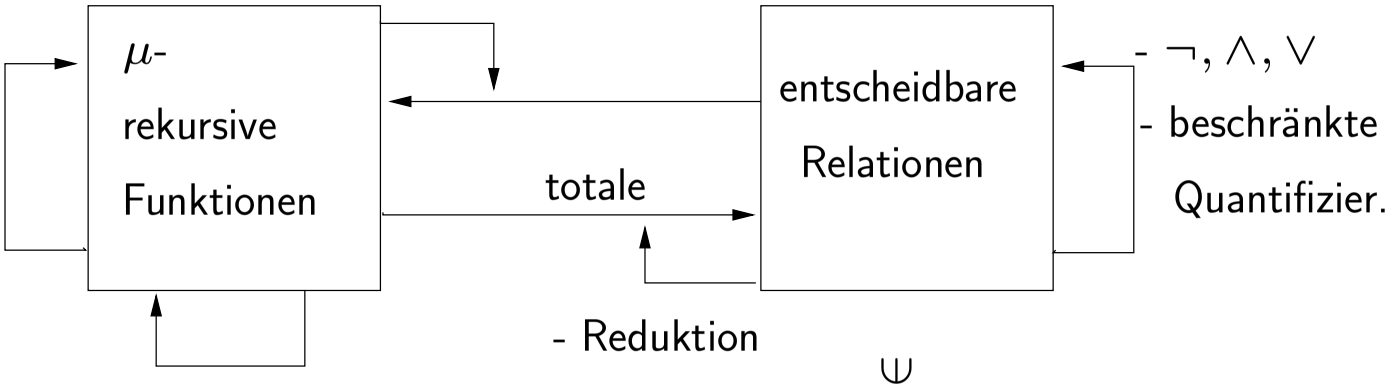
  μ-rekursiv.
  (Das kleinste y mit $R\vec{x}y$, falls es ein solches gibt, sonst $\uparrow$).
  Beachte dabei f ist stets total.

$\mathcal{R}_p(\mathbb{N})$

$\mathcal{P}(\mathbb{N}) \subsetneq \mathcal{R}(\mathbb{N}) \subsetneq \mathcal{R}_p(\mathbb{N})$

- Fallunterscheidung
- Beschränkte Minimierung

- Komposition
- primitiv rekursiv
- Iteration
- verschiedene Rekursionen (z. B. Wertverlauf)



Ψ

$R \subset \mathbb{N}^n$

gdw $\chi_R \in \mathcal{R}_p(\mathbb{N})$

- $f \in \mathcal{R}(\mathbb{N})$ (d. h. total), so ist $graph(f)$ entscheidbar

6.3 Universalität der μ-rekursiven Funktionen

Ziel:

Äquivalenz der μ-rekursiven und der durch while-Programme berechenbaren Funktionen.

6.47 Satz

Jede μ-rekursive Funktion ist durch ein while-Programm über N programmierbar.

Beweis: **Simulationstechnik** Durch Induktion über Aufbau der μ-rekursiven Ausdrücken zeige, dass jede partiell rekursive Funktion durch ein while-Programm berechenbar ist.

Somit sind die Grundfunktionen programmierbar. Komposition, primitive Rekursion, Minimierung von programmierbaren Funktionen liefern programmierbare Funktionen.

Voraussetzung: Werden mehrere Programme benötigt, so Verwendung unterschiedlicher Variablen (ggf. Umbenennung von Variablen).

π μ-rekursiver Ausdruck, $n \geq 1$.
Induktive Konstruktion eines While-Programms $\alpha_\pi^{(n)}$, das die Funktion $f_\pi^{(n)}$ mit Eingabevariable $\vec{X}_\pi = (X_\pi^{(1)}, \ldots, X_\pi^{(n)})$ und der Ausgabevariable Y_π berechnet.

Verwende dabei als Abkürzung
$\vec{X} := \vec{t};$ für $X^1 := t^1; \ \ldots, X^n := t^n;$

Programmierbarkeit der μ-rekursiven Funktionen

- **Grundfunktionen**:

  $\alpha^{(n)}_{NULL}$ ist das Programm $Y_{NULL} := 0;$

  $\alpha^{(n)}_{SUCC}$ ist das Programm $Y_{SUCC} := succ(X^1_{SUCC});$

  $\alpha^{(n)}_{PROJ(i)}$ ist das Programm $Y_{PROJ(i)} := X^i_{PROJ(i)}$ $i \leq n$
  $Y_{PROJ(i)} := 0$ $i > n$

- Sei $F = KOMP(G, H_1, \ldots, H_m)$.

  Dann ist $\alpha^{(n)}_F$ das Programm:

  $\{\text{true}\}$ $\vec{X}_{H_1} := \vec{X}_F;\ \alpha^{(n)}_{H_1}$ $\{Y_{H_1} = f^{(n)}_{H_1}(\vec{X}_F)\}$
  $\ldots$
  $\ldots$
  $\vec{X}_{H_m} := \vec{X}_F;\ \alpha^{(n)}_{H_m}$ $\{Y_{H_m} = f^{(n)}_{H_m}(\vec{X}_F)\}$

  $\vec{X}_G := (Y_{H_1}, \ldots, Y_{H_m});\ \alpha^{(m)}_G$
  $\{Y_G = f^{(m)}_G(Y_{H_1}, \ldots, Y_{H_m})\}$

  $Y_F := Y_G;$
  $\{Y_F = f^{(m)}_G(f^{(n)}_{H_1}(\vec{X}_F), \ldots, f^{(n)}_{H_m}(\vec{X}_F))\}$

Programmierbarkeit der μ-rekursiven Funktionen

- Sei $F = REK(G, H)$. Dann ist $\alpha_F^{(n)}$ das Programm:

$I := 0;$
$\vec{X}_G := (X_F^1, \ldots, X_F^n, I);\ \alpha_G^{(n)} \qquad \{Y_G = g(\vec{X}, 0)\}$
$Y_H := Y_G; \qquad \{Y_H = g(\vec{X}, 0)\}$
while $\neg I = X_F^{n+1}$ **do**
$\quad \vec{X}_H := (X_F^1, \ldots, X_F^n, Y_H, I);\ \alpha_H^{(n+2)}$
$\qquad \{Y_H = h(\ldots)\}$
$\quad I := succ(I);$
end;
$Y_F := Y_H;$

- Sei $F = MIN(G)$. Dann ist $\alpha_F^{(n)}$ das Programm

$I := 0;$
$\vec{X}_G := (\vec{X}_F, I);\ \alpha_G^{(n+1)} \qquad \{Y_G = g(\vec{X}, 0)\}$
while $\neg Y_G = 0$ **do**
$\quad I := succ(I);$
$\quad \vec{X}_G := (\vec{X}_F, I);\ \alpha_G^{(n+1)} \qquad \{Y_G = g(\vec{X}, I)\}$
end;
$Y_F := I;$

Universalität der μ-rekursiven Funktionen

D. h. alle partiell-rekursiven Funktionen sind while-berechenbar.

Dies gilt für jede praktisch anwendbare Programmiersprache, die $0, succ$ enthält und Zuweisung, Fallanweisung und Whileanweisungen zulässt. Insbesondere gilt die Behauptung auch für $N = (\mathbb{N}, 0, succ)$.

Wie zeigt man die Umkehrung: Simulation der Berechnung eines while-Programms durch eine μ-rekursive Funktion.

$I_A(\alpha, z)$: Interpreterfunktion, als primitiv-rekursive Funktion
Iteration
Minimierung.

da Semantik $z[[\alpha]]_A z' \ \text{gdw} \ \exists n \in \mathbb{N}\ I_A^n(\alpha, z) = (\varepsilon, z')$

Problem: Die μ-rekursiven Funktionen sind arithmetische Funktionen, also muss man eine Arithmetisierung aller Konstrukte die bei den While-Programmen vorkommen durchführen.

Codierung syntaktischer Objekte, die in der Definition der Interpreterfunktion vorkommen

$code : syn\_obj \longrightarrow \mathbb{N}$

Vereinbarungen:

- Variablen sind aus Menge $Var = \{V_0, V_1, \dots\}$ zu wählen
- Terme enthalten nur $Var, 0, succ$ (d.h. Programme über N).
- Boolesche Formeln nur mit $\wedge, \neg$

Codierungsfunktion $code$

6.48 Definition Codierungsfunktion $code : syn\_obj \to \mathbb{N}$

Induktiv über Aufbau der syn_obj, definiert durch:

- $code(\varepsilon) = 0$
- $code(0) = [1]$
- $code(V_i) = [2, i]$
- $code(succ(t)) = [3, code(t)]$
- $code(s = t) = [4, code(s), code(t)]$
- $code(\neg B) = [5, code(B)]$
- $code(B \wedge C) = [6, code(B), code(C)]$
- $code(V_i := t;\ ) = [7, i, code(t)]$
- $code(\underline{\textbf{if}}\ B\ \underline{\textbf{then}}\ \beta\ \underline{\textbf{else}}\ \gamma\ \underline{\textbf{end}};\ ) = [8, code(B), code(\beta), code(\gamma)]$
- $code(\underline{\textbf{while}}\ B\ \underline{\textbf{do}}\ \beta\ \underline{\textbf{end}};\ ) = [9, code(B), code(\beta)]$
- $code(A_1 \ldots A_n) = [code(A_1), \ldots, code(A_n)]$
  (Anweisungen $A_i, n \geq 2$)
- $z : V \to \mathbb{N}$ Zustand mit $V \subseteq \{V_0, \ldots, V_m\}$ so
  $code(z) = [x_0, \ldots, x_m]$
  mit $x_i = z(V_i)$, falls $V_i \in V$, sonst $x_i = 0$.

Codierungsfunktion $code$ (Forts.)

Beachte:

- $[\cdot]$ ist die Codierung von endlichen Zahlenfolgen mit Folgenzugriffsfunktion
  $$get(z,i) = \begin{cases} x_i & \text{für } [x_0,\ldots,x_n] = z \ \ (i \leq n) \\ 0 & \text{sonst} \end{cases}$$
  get ist primitiv rekursiv.

  Abkürzung: $z[i]$ für $get(z,i)$.

- $code(z)$ ist wohldefiniert, wegen der Invarianz der Folgencodierung in Bezug auf Nullen am rechten Folgenende.
- Codierung ist nicht injektiv:
  Termcodes, Formelcodes und Programmcodes können gleich sein.

Codierungsfunktion $code$ Beispiel

6.49 Beispiel

Sei α das Programm:

$V_0 := V_1;$
$V_3 := 0;$
while $\neg V_2 = V_3$ **do**
$\quad V_0 := succ(V_0);$
$\quad V_3 := succ(V_3);$
end;

$$code(\alpha) = [code(A_1), code(A_2), code(A_3)]$$
$$= \langle code(A_1), \langle code(A_2), \langle code(A_3), 0\rangle\rangle\rangle$$
$$code(A_1) = [7, 0, [2, 1]] = [7, 0, 7] = 97895$$
$$code(A_2) = [7, 3, 1] = 182$$
$$code(A_3) = [9, \underbrace{code(\neg V_2 = V_3)}, code(V_0 := succ(V_0); V_3 := succV_3))]$$
$$=$$
$$[5, code(V_2 = V_3)]$$
$$=$$
$$[5, [4, [2, 2], [2, 3]]]$$
$$\vdots$$

Primitiv rekursive Simulation der Termauswertung

- Definiere **Termauswertungsfunktion** $\tau : \mathbb{N}^2 \longrightarrow \mathbb{N}$, so dass für alle Terme t und Zustände z gilt

$$(*) \qquad \tau(code(t), code(z)) = val_{N,z}(t)$$

τ ist somit auf den Codes von Termen und Zuständen eindeutig definiert. (Dies genügt!)
Definiere τ durch Fallunterscheidung wie folgt

$$\tau(x,y) = \begin{cases} 0 & \text{falls } x[0] = 1 \ \ (\text{konst } 0) \\ y[x[1]] & \text{falls } x[0] = 2 \ \ (\text{var}) \\ \tau(x[1], y) + 1 & \text{falls } x[0] = 3 \ \ (succ) \\ 2001 & \text{sonst} \end{cases}$$

Offenbar ist $\tau \in \mathcal{P}(\mathbb{N})$ (Fallunterscheidung primitiv rekursiver Relation) und τ erfüllt $(*)$.

Primitiv rekursive Simulation der Auswertung B-Ausdrücke

- Definiere **Auswertungsfunktion Boolescher Formeln** $\beta : \mathbb{N}^2 \longrightarrow \mathbb{N}$, so dass für alle B-Formeln B und Zustände z gilt

$$(**) \qquad \beta(code(B), code(z)) = \begin{cases} 1 & \text{falls } \mathbb{N} \models_z B \\ 0 & \text{sonst} \end{cases}$$

  Definiere β durch Fallunterscheidung + Wertverlaufsrekursion.

$$\beta(x,y) = \begin{cases} \chi_=(\tau(x[1],y), \tau(x[2],y)) & \text{falls } x[0] = 4 \\ 1 - \beta(x[1],y) & \text{falls } x[0] = 5 \\ \beta(x[1],y) \cdot \beta(x[2],y) & \text{falls } x[0] = 6 \\ 2001 & \text{sonst} \end{cases}$$

  Offenbar ist $\beta \in \mathcal{P}(\mathbb{N})$ und β erfüllt $(**)$.

Primitiv rekursive Simulation der Speicheränderungen + Interpreterfunktion

- Definiere **Funktion für Speicheränderungen** (bei Zuweisungen): $\sigma : \mathbb{N}^3 \longrightarrow \mathbb{N}$, so dass für alle Zustände z, Variablen V_i und Werte $a \in \mathbb{N}$ gilt:

$$(***) \qquad \sigma(code(z), i, a) = code(z(V_i/a))$$

  Definiere σ rekursiv über i durch

$$\sigma(x, 0, a) = \langle a, rest(x)\rangle$$
$$\sigma(x, i+1, a) = \langle \text{first}(x), \sigma(\text{rest}(x), i, a)\rangle$$

  Offenbar ist $\sigma \in \mathcal{P}(\mathbb{N})$ und erfüllt $(***)$.

- Definiere **Funktion zur Simulation der Interpreterfunktion** I_N, $i : \mathbb{N} \longrightarrow \mathbb{N}$, so dass für alle Programme α und α' und Zustände z und z' gilt:

$$(****) \qquad I_N(\alpha, z) = (\alpha', z') \qquad \text{gdw}$$
$$i(\langle code(\alpha), code(z)\rangle) = \langle code(\alpha'), code(z')\rangle$$

  i muss für Codierungen der Form $\langle p, y\rangle$ richtig arbeiten.

  Beachte es gilt
  $p = \langle \text{first}(p), \text{rest}(p)\rangle$ (Paarungsfunktion) und für $p > 0$ die erste Anweisung des von p codierten Programms den Code first(p) und das Restprogramm den Code rest(p) haben.

Interpreterfunktion (Forts.)

$i(\langle p, y\rangle)$ wird durch Fallunterscheidung definiert:

$i(\langle p, y\rangle) =$

$$\begin{cases} \langle p, y\rangle & \text{falls } p = 0 \\ \langle rest(p), \sigma(y, \text{first}(p)[1], & \text{falls first}(p)[0] = 7 \\ \qquad \tau(\text{first}(p)[2], y))\rangle & \\ \langle \text{first}(p)[2] * \text{rest}(p), y\rangle & \text{falls first}(p)[0] = 8 \\ & \text{und } \beta(\text{first}(p)[1], y) = 1 \\ \langle \text{first}(p)[3] * \text{rest}(p), y\rangle & \text{falls first}(p)[0] = 8 \\ & \text{und } \beta(\text{first}(p)[1], y) = 0 \\ \langle \text{first}(p)[2] * p, y\rangle & \text{falls first}(p)[0] = 9 \\ & \text{und } \beta(\text{first}(p)[1], y) = 1 \\ \langle \text{rest}(p), y\rangle & \text{falls first}(p)[0] = 9 \\ & \text{und } \beta(\text{first}(p)[1], y) = 0 \\ 2001 & \text{sonst} \end{cases}$$

Offenbar ist $i \in \mathcal{P}(\mathbb{N})$ und erfüllt $(* * **)$.

Simulation der Speicherinitialisierung und Ausgabefunktion

- **Speicherinitialisierung** für Programm mit Code p,
  $\mathsf{inp}^{(n)} : \mathbb{N}^{n+1} \longrightarrow \mathbb{N}$.
  Initialisierung bei Eingabe von n-Zahlen $x_1, \ldots, x_n$ in den Variablen $V_1, \ldots, V_n$. V_0 dient als Ausgabevariable.

$$\mathsf{inp}^{(n)}(p, x_1, \ldots, x_n) = \langle p, [0, x_1, \ldots, x_n] \rangle$$

- **Ausgabe** des berechneten Wertes für Programm mit Code p,
  $\mathsf{out} : \mathbb{N} \longrightarrow \mathbb{N}$

$$\mathsf{out}(\langle p, x \rangle) = x[0]$$

- $\mathsf{inp}^{(n)}$ und out sind in $\mathcal{P}(\mathbb{N})$.

Wir haben nun alle Bestandteile zusammen um die Simulation der while-berechenbaren Funktion durch die partiell rekursiven Funktionen nachzuweisen. Die Interpreterfunktion ist so lange zu iterieren, bis als Restprogramm das leere Programm (mit Codezahl 0) entsteht. Diese Iterationszahl kann als Zeitkomplexitätsfunktion betrachtet werden. Sie misst nämlich die Anzahl der ausgeführten Anweisungen sowie die Anzahl der ausgewerteten B-Formeln.

Laufzeitfunktionen
Universelle Funktionen

6.50 Definition Zeitkomplexitätsfunktion

Sei p Code eines Programms mit Eingabevariablen $V_1, \ldots, V_n$. Die Laufzeit (Anzahl der Rechenschritte) bei Eingabe $x_1, \ldots, x_n$ sei definiert durch folgende μ-rekursive Funktion $\Phi^{(n)} : \mathbb{N}^{n+1} \longrightarrow \mathbb{N}$ mit

$$\Phi^{(n)}(p, x_1, \ldots, x_n) = \mu t.\text{first}(i^t(\text{inp}^{(n)}(p, x_1, \ldots, x_n)) = 0$$

Φ heißt **Zeitkomplexitätsfunktion.**

Simulation der Berechnung des Programms mit Codezahl p bei Eingabe $x_1, \ldots, x_n$ durch μ-rekursive Funktion $\varphi^{(n)} : \mathbb{N}^{n+1} \longrightarrow \mathbb{N}$.

$$\varphi^{(n)}(p, x_1, \ldots, x_n) = \text{out}(i^{\Phi^{(n)}(p,x_1,\ldots,x_n)}(\text{inp}^{(n)}(p, x_1, \ldots, x_n)))$$

$\varphi^{(n)}(p, x_1, \ldots, x_n) \downarrow$ $\;\text{gdw}$ Programm mit Code p terminiert bei Eingabe $x_1, \ldots, x_n$.

$\varphi^{(n)}(p, x_1, \ldots, x_n)$ ist dann die Ausgabe (in V_0) vom Programm mit Codezahl p bei Eingabe $x_1, \ldots, x_n$ in $V_1, \ldots, V_n$.

$\rightsquigarrow$ **Universeller Rechner für While-Programme**

Schreibweisen: $\varphi_p^{(n)}(x_1, \ldots, x_n)$ für $\varphi^{(n)}(p, x_1, \ldots, x_n)$

bzw. $\Phi_p^{(n)}(x_1, \ldots, x_n)$ für $\Phi^{(n)}(p, x_1, \ldots, x_n)$.

(n) weglassen, falls $n = 1$.

Offenbar gilt $\Phi^{(n)}, \varphi^{(n)} \in \mathcal{R}_p(\mathbb{N})$.

Äquivalenz while-berechenbaren und μ-rekursiven Funktionen

6.51 Satz

Ist $f : \mathbb{N}^n \to \mathbb{N}$ mit $n \geq 1$ durch ein while-Programm in N berechenbar, dann ist f μ-rekursiv.

Beweis: o.b.d.A. Eingabevariable $V_1, \ldots, V_n$, Ausgabe V_0.

Alle anderen benutzten Variablen mit 0 initialisiert (d. h. Codezahl für Zustand der $V_0, \ldots, V_n$ belegt ist auch Codezahl für Zustand der $V_0, \ldots, V_n, V_{n+1}, \ldots, V_m$ belegt, wobei $V_{n+1}, \ldots, V_m$ mit 0 belegt werden).

Sei α ein solches Programm, α berechne $f : \mathbb{N}^n \to \mathbb{N}$, dann gilt

$$f(x_1, \ldots, x_n) = \varphi^{(n)}(code(\alpha), x_1, \ldots, x_n)$$

Da $\varphi^{(n)} \in \mathcal{R}_p(\mathbb{N})$ gilt auch $f \in \mathcal{R}_p(\mathbb{N})$.

These: Die Klasse der berechenbaren Funktionen ist $\mathcal{R}_p(\mathbb{N})$.

6.52 Folgerung Jede Funktion $f \in \mathcal{R}_p$ lässt sich mittels maximal einmaliger Anwendung der Minimierung angewandt auf eine totale Funktion definieren.

Beweis: $f(x_1, \ldots, x_n) = \varphi^{(n)}(code(\alpha_f), x_1, \ldots, x_n)$, wobei f vom Programm α_f mit Eingabevariablen $V_1, \ldots, V_n$ und Ausgabe V_0 berechnet wird.

Beachte: Zu $f \in \mathcal{R}_p$, $f : \mathbb{N}^n \to \mathbb{N}$ gibt es ein $p \in \mathbb{N}$, so dass $f(x_1, \ldots, x_n) = \varphi_p^{(n)}(x_1, \ldots, x_n)$. p **ist Index für** f**.**

Universelle μ-rekursive Funktionen

Es gibt Funktionen $f : \mathbb{N}^2 \longrightarrow \mathbb{N}$, so dass es für jede μ-rekursive Funktion $g : \mathbb{N} \longrightarrow \mathbb{N}$ einen Index i gibt, mit $f(i, \_) = g$.

Beachte: Es gibt stets ∞-viele i für festes g. (wähle z. B. $f = \varphi^{(1)}$)

Das Ergebnis für While-Programme lässt sich leicht auf rekursive Programme übertragen.

$code(call \ldots)$
Für die erweiterte Interpreterfunktion I_Ω lässt sich wiederum eine primitiv rekursive Simulation i_Ω leicht angeben. Man erhält entsprechend:

$$\exists t \; I^t_\Omega(\alpha, z) = (\varepsilon, z') \Leftrightarrow \exists t' \; i^{t'}_\Omega(\langle code(\alpha), code(z)\rangle) = \langle 0, code(z')\rangle$$

Insbesondere gilt:

Rekursive While-Programme können nicht mehr berechnen als While-Programme.

Beide Klassen sind $\mathcal{R}_p(\mathbb{N})$.(Für die Algebra N)