6 Berechenbarkeit

Programmierbare/Berechenbare Funktionen

- Imperative Programmiersprache: while-Programme
- Funktionale Programmiersprache: μ-rekursive Ausdrücke
- Logische Programmiersprachen: Prolog ...
  Deklarativ vs. Prozedural
- Abstrakte Maschinen Modelle
  Turing-Maschine, Register-Maschine;
- Techniken: Simulation von Berechnungen, Übersetzung, Interpretation
- Universelle Modelle: Compiler

Aufbau Kapitel 6:

- Primitiv rekursive Funktionen
- μ-rekursive Funktionen (partiell rekursive Funktionen)
- Universalität
- Rekursionstheorie
- Churchsche These
- Wortfunktionen

6.1 Primitiv rekursive Funktionen $\mathcal{P}(\mathbb{N})$

Funktionen: $f : \mathbb{N}^n \to \mathbb{N} \quad n \geq 1$. **Arithmetische Funktionen.**

Verwende „effektive"- Operatoren auf Funktionen, um aus Ausgangsfunktionen + Operatoren neue Funktionen zu definieren.

Erinnerung: Gleichheit von Funktionen $f : A \to B$, $g : A \to B$

$f \sqsubseteq g$ gdw $dom(f) \subseteq dom(g) \wedge f(x) = g(x)$
für $x \in dom(f)$

$f = g$ gdw $dom(f) = dom(g) \wedge f(x) = g(x)$
für $x \in dom(f)$ gdw $(f \sqsubseteq g \wedge g \sqsubseteq f)$.

6.1 Definition Komposition - Primitive Rekursion

a) Seien $g : \mathbb{N}^n \to \mathbb{N}$, $h_1, \ldots, h_n : \mathbb{N}^m \to \mathbb{N}$ Funktionen $n, m \geq 1$
$f : \mathbb{N}^m \to \mathbb{N}$ entsteht aus g und $h_1, \ldots, h_n$ durch **Komposition**, falls gilt
$f(\vec{x}) \downarrow$ gdw $h_1(\vec{x}) \downarrow, \ldots, h_n(\vec{x}) \downarrow$ und $g(h_1(\vec{x}), \ldots, h_n(\vec{x})) \downarrow$ und in diesem Fall ist
$f(\vec{x}) = g(h_1(\vec{x}), \ldots, h_n(\vec{x}))$
Schreibe dafür $f = g \circ (h_1, \ldots, h_n)$
Beachte Stelligkeiten der Funktionen.
Sind $g, h_1, \ldots, h_n$ total, so auch f.
Gilt die Umkehrung?

Operatoren auf Funktionen

b) Seien $g : \mathbb{N}^{n+1} \to \mathbb{N}$, $h : \mathbb{N}^{n+2} \to \mathbb{N}$ Funktionen.
$f : \mathbb{N}^{n+1} \to \mathbb{N}$ entsteht aus g und h durch **primitive Rekursion**, falls gilt:
$f(\vec{x}, 0) \downarrow$ gdw $g(\vec{x}, 0) \downarrow$ $(\vec{x} \in \mathbb{N}^n)$ und in diesem Fall ist
$f(\vec{x}, 0) = g(\vec{x}, 0)$ und
$f(\vec{x}, y+1) \downarrow$ gdw $f(\vec{x}, y) \downarrow$ und $h(\vec{x}, f(\vec{x}, y), y) \downarrow$ $(\vec{x} \in \mathbb{N}^n)$ und in diesem Fall ist
$f(\vec{x}, y+1) = h(\vec{x}, f(\vec{x}, y), y)$
Schreibe dafür $f = R(g, h)$.
Beachte Stelligkeiten der Funktionen.

6.2 Bemerkung - Beispiele: Betrachtet man die Gleichung

$$F(\vec{x}, z) = \begin{cases} g(\vec{x}, 0) & z = 0 \\ h(\vec{x}, F(\vec{x}, y), y) & z = y + 1 \end{cases}$$

so ist $f = R(g, h)$ die kleinste (bzgl. $\sqsubseteq$) Funktion, die diese Gleichung erfüllt.

Sind $g(\cdot, 0) : \mathbb{N}^n \to \mathbb{N}$ und $h : \mathbb{N}^{n+2} \to \mathbb{N}$ total, so ist auch f total.

Gilt die Umkehrung?

Die primitive Rekursion folgt einem sehr strengen Schema und entspricht der Berechnung des Funktionswerts $f(\vec{x}, n+1)$ aus dem Funktionswert $f(\vec{x}, n)$, wobei die Verankerung bei $f(\vec{x}, 0)$ erfolgt.

Beispiele

- $+$ Addition auf $\mathbb{N}^2$: Mit $g(u, v) = u$ und $h(u, v, w) = v + 1$ gilt $x + y = R(g, h)$

$$x + y = \begin{cases} x & \text{falls } y = 0 \\ (x + (y - 1)) + 1 & \text{falls } y \neq 0 \end{cases}$$

oder

$$x + y = \begin{cases} x & \text{falls } y = 0 \\ (x + z) + 1 & \text{falls } y = z + 1 \end{cases}$$

- $\cdot$ Multiplikation auf $\mathbb{N}^2$: Mit $g(u, v) = 0$ und $h(u, v, w) = v + u$ gilt $x \cdot y = R(g, h)$

$$x \cdot y = \begin{cases} 0 & \text{falls } y = 0 \\ (x \cdot z) + x & \text{falls } y = z + 1 \end{cases}$$

- Fakultät fac auf $\mathbb{N}$: Mit $g(u) = 1$ und $h(u, v) = u \cdot (v + 1)$ gilt $fac(y) = R(g, h)$

$$fac(y) = \begin{cases} 1 & \text{falls } y = 0 \\ fac(z) \cdot (z + 1) & \text{falls } y = z + 1 \end{cases}$$

- Welche Funktion f wird durch folgende Festlegung definiert. Sei

$$h(u, v) = \begin{cases} u & u \text{ gerade} \\ \uparrow & \text{sonst} \end{cases}$$

$$f(y) = \begin{cases} 0 & \text{falls } y = 0 \\ h(f(z), z) & \text{falls } y = z + 1 \end{cases}$$

Primitiv rekursive Ausdrücke

6.3 Definition

Syntax: Die Menge der primitiv rekursiven Ausdrücke sind die Zeichenreihen, die durch den folgenden Kalkül erzeugt werden:

$$\overline{NULL} \qquad \overline{SUCC} \qquad \overline{PROJ(i)} \text{ für } i \geq 1$$

$$\frac{G, H_1, \ldots, H_m}{KOMP(G, H_1, \ldots, H_m)} \text{ für } m \geq 1 \qquad \frac{G, H}{REK(G, H)}$$

Semantik: Jeder primitiv rekursive Ausdruck π repräsentiert für beliebige Stelligkeit $n \geq 1$ eine Funktion $f_\pi^{(n)} : \mathbb{N}^n \to \mathbb{N}$, die induktiv über den Aufbau von π wie folgt definiert ist:

$$\left.\begin{array}{l} f_{NULL}^{(n)}(x_1, \ldots, x_n) = 0 \\ f_{SUCC}^{(n)}(x_1, \ldots, x_n) = x_1 + 1 \\ f_{PROJ(i)}^{(n)}(x_1, \ldots, x_n) = \begin{cases} x_i & \text{falls } 1 \leq i \leq n \\ 0 & \text{sonst} \end{cases} \end{array}\right\} \text{Grundfunktionen}$$

$$f_{KOMP(G,H_1,\ldots,H_m)}^{(n)}(x_1, \ldots, x_n) = f_G^{(m)} \circ (f_{H_1}^{(n)}, \ldots, f_{H_m}^{(n)})(x_1, \ldots, x_n)$$

$f_{REK(G,H)}^{(n)} = R(f_G^{(n)}, f_H^{(n+1)})$, d. h.

$f_{REK(G,H)}^{(n)}(x_1, \ldots, x_{n-1}, 0) = f_G^{(n)}(x_1, \ldots, x_{n-1}, 0)$ und

$$f_{REK(G,H)}^{(n)}(x_1, \ldots, x_{n-1}, y+1) = f_H^{(n+1)}(x_1, \ldots, x_{n-1}, f_{REK(G,H)}^{(n)}(x_1, \ldots, x_{n-1}, y), y)$$

Primitiv rekursive Funktionen

Die Menge aller Funktionen $f : \mathbb{N}^n \to \mathbb{N}$ $(n > 0)$, für die $f = f_\pi^{(n)}$ mit einem primitiv rekursiven Ausdruck π gilt, heißt die Menge der **primitiv rekursiven Funktionen**.

Bezeichnung $\mathcal{P}(\mathbb{N})$

6.4 Beispiel Folgende Funktionen sind primitiv rekursiv.

- Konstante Funktionen beliebiger Stelligkeit:
  $a \in \mathbb{N} \qquad c_a^{(n)} : \mathbb{N}^n \to \mathbb{N} \qquad c_a^{(n)}(x_1, \ldots, x_n) = a$ (total)

Zeige $c_a^{(n)} = f_\pi^{(n)}$ für geeignetes π:

$a = 0$ so klar, wähle $\pi = NULL$

$a = 1 \quad f_{KOMP(SUCC,NULL)}^{(n)}(\vec{x}) = f_{SUCC}^{(1)}(f_{NULL}^{(n)}(\vec{x})) = 1$

Ind. Schritt:

$a = m \qquad$ sei $f_{\pi_a}^{(n)}(\vec{x}) = a$

$a = m+1 \; f_{KOMP(SUCC,\pi_a)}^{(n)}(\vec{x}) = f_{SUCC}^{(1)}(f_{\pi_a}^{(n)}(\vec{x})) = m + 1$

d. h. $\pi_a =$ KOMP(SUCC,KOMP(SUCC,...KOMP(SUCC,NULL)...))
a-mal $KOMP(SUCC, \ldots)$

Beispiel (Forts.)

- Die Vorgänger Funktion auf $\mathbb{N}$ $pred : \mathbb{N} \to \mathbb{N}$ ist primitiv rekursiv:
  $pred(0) = 0$
  $pred(y+1) = y$ (total)
  $pred(0) = f_{NULL}^{(1)}(0)$
  $pred(y+1) = f_{PROJ(2)}^{(2)}(pred(y), y) = y$,
  d. h. mit $PRED = REK(NULL, PROJ(2))$ gilt
  $pred = f_{PRED}^{(1)} = f_{REK(NULL,PROJ(2))}^{(1)}$.

6.5 Lemma

Die Menge der primitiv rekursiven Funktionen ist abgeschlossen gegenüber Komposition und primitiver Rekursion.

Sind $g : \mathbb{N}^m \to \mathbb{N}, h_1, \ldots, h_m : \mathbb{N}^n \to \mathbb{N} \in \mathcal{P}(\mathbb{N})$, so auch $g \circ (h_1, \ldots, h_m) : \mathbb{N}^n \to \mathbb{N} \in \mathcal{P}(\mathbb{N})$.

Sind $g : \mathbb{N}^{n+1} \to \mathbb{N}, h : \mathbb{N}^{n+2} \to \mathbb{N} \in \mathcal{P}(\mathbb{N})$, so auch $R(g, h)$.

Beweis: Seien G und $H_1, \ldots, H_m$ primitiv rekursive Ausdrücke für g und $h_1, \ldots, h_m$. Dann ist $KOMP(G, H_1, \ldots, H_m)$ ein primitiv rekursiver Ausdruck für $g \circ (h_1, \ldots, h_m)$.

Analog ist $REK(G, H)$ primitiv rekursiver Ausdruck für $R(g, h)$, falls G, H primitiv rekursive Ausdrucke für g bzw. h sind.

Primitiv rekursive Funktionen (Fort.)

Die Menge der primitiv rekursiven Funktionen $\mathcal{P}(\mathbb{N})$ ist also charakterisiert als die kleinste Menge von Funktionen $f : \mathbb{N}^n \to \mathbb{N}$ $(n > 0)$ die die Grundfunktionen enthält und abgeschlossen ist gegenüber Komposition und primitiver Rekursion.

In der Literatur findet man oft die Betrachtung von Funktionen $f : \mathbb{N}^n \to \mathbb{N}^m$ $(n, m > 0)$. Diese lassen sich über die **Parallelausführung** $f := \langle g, h \rangle$ von $g : \mathbb{N}^n \to \mathbb{N}^s, h : \mathbb{N}^n \to \mathbb{N}^t$ die erklärt ist durch

$$f : \mathbb{N}^n \to \mathbb{N}^{s+t} \qquad \langle g, h \rangle(x) = (g(x), h(x))$$

aus den obigen Funktionen gewinnen.

Offenbar sind die primitiv rekursiven Ausdrücke sehr einfache Programme. Sie sind aufgebaut aus $NULL, SUCC, PROJ(i)$ und den variadischen Operator $KOMP$ und den binären Operator REK. Die Interpretation der atomaren Ausdrücke durch die Grundfunktionen und der Operatoren durch die offensichtlich „effektiven“ Operationen Komposition und primitive Rekursion machen deutlich, dass diese Programme effektive Berechnungen darstellen. Jedes Programm erlaubt es für jedes $n > 0$ eine Funktion der Stelligkeit n zu berechnen. D. h. ein Programm berechnet unendlich viele Funktionen.

Beachte: Ein primitiv rekursiver Ausdruck stellt stets für jedes n eine Funktion dar. Diese können recht unterschiedlich sein. Siehe z. B. $f_{PROJ(i)}^{(n)}$. Welche Funktion ist $f_{REK(NULL,PROJ(2))}^{(2)}$?

Nachweis von Eigenschaften primitiv rekursiver Ausdrücke oder primitiv rekursiver Funktionen

Erneut: Induktion über Aufbau der primitiv rekursiven Ausdrücke (strukturelle Induktion) bzw. für die Menge $\mathcal{P}(\mathbb{N})$ die sogenannte **Induktion über den Aufbau**: Zeige die Eigenschaft gilt für die Grundfunktionen und die Eigenschaft bleibt erhalten bei Komposition und primitiver Rekursion.

6.6 Lemma

Jede primitiv rekursive Funktion ist total.

6.7 Beispiel Weitere primitiv rekursive Funktionen

$add : \mathbb{N}^2 \to \mathbb{N}$ $\quad add(x, y) = x + y$ primitiv rekursiv

$add(x, 0) = f^{(2)}_{PROJ(1)}(x, 0)$

$add(x, y + 1) = f^{(1)}_{SUCC}(f^{(3)}_{PROJ(2)}(x, add(x, y), y))$

$ADD :: REK(PROJ(1), KOMP(SUCC, PROJ(2))$

ist ein primitiv rekursiver Ausdruck für add, d. h. $add = f^{(2)}_{ADD}$.

Die Multiplikation $mult : \mathbb{N}^2 \to \mathbb{N}$ mit $mult(x, y) = x \cdot y$ ist primitiv rekursiv:

$mult(x, 0) = f^{(2)}_{NULL}(x, 0)$

$mult(x, y + 1) = add(f^{(3)}_{PROJ(1)}(x, mult(x, y), y),$
$\qquad f^{(3)}_{PROJ(2)}(x, mult(x, y), y))$

d. h. $REK(NULL, KOMP(ADD, PROJ(1), PROJ(2)))$ repräsentiert folglich $mult$.

Beispiele und Vereinfachungen

Die Funktion $sgn : \mathbb{N} \to \mathbb{N}$ mit

$$sgn(x) = \begin{cases} 0 & x = 0 \\ 1 & \text{sonst} \end{cases}$$

$sgn(0) = f^{(1)}_{NULL}(0)$

$sgn(y + 1) = f^{(2)}_{KOMP(SUCC,NULL)}(sgn(y), y)$

d. h. $REK(NULL, KOMP(SUCC, NULL))$ repräsentiert sgn.

Analog die Funktion $\overline{sgn} : \mathbb{N} \to \mathbb{N}$ mit

$$\overline{sgn}(x) = \begin{cases} 1 & x = 0 \\ 0 & \text{sonst} \end{cases}$$

Vereinfachungen: Auflockerung des strengen Schemas der primitiven Rekursion.

- Variablen permutieren, mehrfache Verwendung, oder Nicht-Verwendung von Variablen.
- Weitere Abschlusseigenschaften.
- Verwendung bereits als primitiv rekursiv nachgewiesener Funktionen.

Vereinfachungen

6.8 Lemma

Sei $g : \mathbb{N}^m \to \mathbb{N}$ primitiv rekursiv, $m \geq n$ und seien $1 \leq i_1 \leq n, \ldots, 1 \leq i_m \leq n$ Indizes. Dann ist auch die Funktion $h : \mathbb{N}^n \to \mathbb{N}$ mit

$$h(x_1, \ldots, x_n) = g(x_{i_1}, \ldots, x_{i_m})$$

primitiv rekursiv.

Beweis: Es gilt

$h(x_1, \ldots, x_n) =$
$g(f^{(n)}_{PROJ(i_1)}(x_1, \ldots, x_n), \ldots, f^{(n)}_{PROJ(i_m)}(x_1, \ldots, x_n))$

6.9 Beispiel Es genügt in Zukunft, den Nachweis der primitiven Rekursion einer Funktion auf der Basis einer Rekursionsgleichung wie bei den folgenden Funktionen zu führen.

- Nicht negative Differenz: $- : \mathbb{N}^2 \to \mathbb{N}$
  $x - 0 \quad = x$
  $x - (y + 1) = pred(x - y)$
- Fakultät $fac : \mathbb{N} \to \mathbb{N}$
  $fac(0) \quad = 1 \; (= f^{(1)}_{KOMP(SUCC,NULL)}(0))$
  $fac(y + 1) = fac(y) \cdot (y + 1)$
  $(= f^{(2)}_{KOMP(MULT(PROJ(1),KOMP(SUCC,PROJ(2)))}(fac(y), y))$

Weitere Abschlusseigenschaften

- Insbesondere ist die Funktion $|x - y| : \mathbb{N}^2 \to \mathbb{N}$, mit

  $$|x - y| \; = \; (x - y) + (y - x)$$

  primitiv rekursiv.
- Einfache Fallunterscheidung. Oft werden Funktionen nur in bestimmten Bereichen benötigt. Sei etwa $h : \mathbb{N}^2 \to \mathbb{N}$ primitiv rekursiv, dann ist auch die Funktion

  $$F(x, y) = \begin{cases} h(x, y) & \text{falls } x > y \\ 0 & \text{sonst} \end{cases}$$ primitiv rekursiv.

  Es ist $F(x, y) = sgn(x - y) \cdot h(x, y)$
  Später werden wir allgemeinere Formen der Fallunterscheidung kennenlernen.

6.10 Lemma Abschluss von $\mathcal{P}(\mathbb{N})$ gegenüber Iteration.

Sei $f : \mathbb{N} \to \mathbb{N}$ primitiv rekursiv, dann ist auch $g : \mathbb{N}^2 \to \mathbb{N}$ mit $g(x, t) = f^t(x)$ primitiv rekursiv.

Beweis: Es ist

$g(x, 0) \quad = x$
$g(x, t + 1) = f(g(x, t))$

Frage: Lässt sich jede „Rekursionsgleichung“ durch primitive Rekursion simulieren?

Andere Rekursionsformate

- Das folgende Format ist erlaubt:
  $f(0,y) = g(y)$
  $f(x+1,y) = h(x,f(x,y),y)$
  Mit primitiv rekursiven Funktionen g, h. Dann ist f auch primitiv rekursiv (Argumente vertauscht).
- Hingegen ist die alternative Definition der **Iteration**:
  $g(x,0) = x$
  $g(x,t+1) = g(f(x),t)$
  nicht vom Format einer primitiven Rekursion, da der Parameter $f(x)$ statt x in der Rekursion verwandt wird.
  Wir werden gleich zeigen, dass auch dieses Format erlaubt ist.
- Allerdings ist die **Ackermannfunktion** $A : \mathbb{N}^2 \to \mathbb{N}$, die durch folgende Rekursionsgleichung definiert wird
  $A(0,y) = y+1$
  $A(x+1,0) = A(x,1)$
  $A(x+1,y+1) = A(x,A(x+1,y))$
  keine primitiv rekursive Funktion (Beweis später).

A ist eine totale Funktion, die sehr schnell wächst. Überzeugen Sie sich!

Andere Rekursionsformate (Fort.)

6.11 Lemma

Seien $g : \mathbb{N} \to \mathbb{N}$, $h : \mathbb{N}^3 \to \mathbb{N}$, $w : \mathbb{N} \to \mathbb{N}$ primitiv rekursiv und $f : \mathbb{N}^2 \to \mathbb{N}$ durch die Gleichungen

$f(x,0) = g(x)$
$f(x,y+1) = h(x,f(w(x),y),y)$ definiert.

Dann ist auch f primitiv rekursiv.

Beweis:

Sei $F(t,x,y) = \begin{cases} f(w^{t-y}(x),y) & \text{falls } t \geq y \\ 0 & \text{sonst} \end{cases}$

Es gilt $F(t,x,0) = f(w^t(x),0) = g(w^t(x))$ und
für $t \geq y+1$

$$\begin{aligned} F(t,x,y+1) &= f(w^{t-y-1}(x), y+1) \\ &= h(w^{t-y-1}(x), f(w(w^{t-y-1}(x)),y),y) \\ &= h(w^{t-y-1}(x), f(w^{t-y}(x),y),y) \\ &= h(w^{t-y-1}(x), F(t,x,y),y) \end{aligned}$$

Für $t < y+1$ gilt
$F(t,x,y+1) = 0$

F ist also primitiv rekursiv.
Wegen $f(x,y) = F(y,x,y)$ ist auch f primitiv rekursiv.

Fallunterscheidung

6.12 Lemma Abschluss gegenüber Fallunterscheidung

Seien $g_i : \mathbb{N}^n \to \mathbb{N}$, $h_i : \mathbb{N}^n \to \mathbb{N}$ mit h_i totale Funktionen für $i = 1,\ldots,r$, so dass es zu jedem $\vec{x} \in \mathbb{N}^n$ es genau ein i gibt mit $h_i(\vec{x}) = 0$. Die Fallunterscheidung mit den Funktionen g_i, h_i ist die Funktion

$f = FU(g_i, h_i \;\; i = 1,\ldots,k)$, mit

$$f(\vec{x}) = \begin{cases} g_1(\vec{x}) & \text{falls } h_1(\vec{x}) = 0 \\ \vdots & \\ g_k(\vec{x}) & \text{falls } h_k(\vec{x}) = 0 \end{cases}$$

Offenbar gilt:
$f(\vec{x}) = \overline{sgn}(h_1(\vec{x})) \cdot g_1(\vec{x}) + \cdots + \overline{sgn}(h_k(\vec{x})) \cdot g_k(\vec{x})$,
d. h. sind $g_i, h_i \in \mathcal{P}(\mathbb{N})$, so ist auch $f \in \mathcal{P}(\mathbb{N})$.

Die Fallunterscheidung wird meistens mit $k = 2$ angewendet. Sei h gegeben und $h_1(x) = h(x), h_2(x) = \overline{sgn}(h(x))$. Dann gilt

$$f(x) = \begin{cases} g_1(x) & h_1(x) = 0 \\ g_2(x) & h_2(x) = 0 \end{cases} = \begin{cases} g_1(x) & h(x) = 0 \\ g_2(x) & \text{sonst} \end{cases}$$

Primitiv rekursive Relationen

6.13 Definition

Eine **Relation** $R \subseteq \mathbb{N}^n$ **heißt primitiv rekursiv**, falls ihre charakteristische Funktion $\chi_R \in \mathcal{P}(\mathbb{N})$.

Erinnerung für $\vec{x} \in \mathbb{N}^n$ gilt:

$$\chi_R(\vec{x}) = \begin{cases} 1 & \text{falls } \vec{x} \in R \\ 0 & \text{falls } \vec{x} \notin R \end{cases}$$

6.14 Beispiel

Primitiv rekursive Relationen sind:

- Die Gleichheitsrelation: $= \;\subseteq\; \mathbb{N} \times \mathbb{N}$
  $\chi_=(x,y) = 1 - |x-y|$
- Kleinerrelation: $< \;\subseteq\; \mathbb{N} \times \mathbb{N}$
  $\chi_<(x,y) = sgn(y-x)$
- Kleinergleichrelation: $\leq \;\subseteq\; \mathbb{N} \times \mathbb{N}$
  $\chi_\leq(x,y) = \chi_=(x,y) + \chi_<(x,y)$
- Analog Größerrelation und Größergleichrelation.

Abschlusseigenschaften primitiv rekursiver Relationen

Erinnerung:

Seien $R, S \subseteq \mathbb{N}^n$ Relationen.

Dann

- $\neg R$ **Komplement** von R $\quad \mathbb{N}^n - R$
- $R \wedge S$ **Durchschnitt** von R und S $\quad R \cap S$
- $R \vee S$ **Vereinigung** von R und S $\quad R \cup S$

6.15 Lemma

Sind $R, S \subseteq \mathbb{N}^n$ primitiv rekursiv, so auch $\neg R$, $R \wedge S$, $R \vee S$.

Beweis: Es ist

$\chi_{\neg R} = 1 - \chi_R$, $\chi_{R \wedge S} = \chi_R \cdot \chi_S$ und $R \vee S = \neg(\neg R \wedge \neg S)$.

Frage: Gilt $\chi_{R \vee S} = \chi_R + \chi_S$?

Insbesondere sind $\neq, \leq, >, \geq$ primitiv rekursiv.

Reduzierbarkeit

6.16 Lemma

Seien $S \subseteq \mathbb{N}^n$, $h_i : \mathbb{N}^n \to \mathbb{N}$ $(1 \leq i \leq n)$ primitiv rekursiv. Dann ist auch die Relation $R \subseteq \mathbb{N}^m$ mit

$$R\vec{x} \quad \text{gdw} \quad Sh_1(\vec{x}) \cdots h_n(\vec{x})$$

primitiv rekursiv.

R ist auf S **primitiv rekursiv reduzierbar**, falls es primitiv rekursive Funktionen $h_i : \mathbb{N}^m \to \mathbb{N}$ $(1 \leq i \leq n)$ gibt mit obiger Eigenschaft.

Beweis:

$$\begin{aligned}\chi_R(\vec{x}) &= \chi_S(h_1(\vec{x}), \ldots, h_n(\vec{x})) \\ &= \chi_S \circ (h_1, \ldots, h_n)(\vec{x})\end{aligned}$$

6.17 Beispiel Sei $R \subseteq \mathbb{N}^2$ mit

Rxy gdw x ist ganzzahliger Anteil der Quadratwurzel von y.

Dann ist R primitiv rekursiv

$$Rxy \quad \text{gdw} \quad x \cdot x \leq y \wedge (x+1) \cdot (x+1) > y$$

Wende Lemma an mit

$$Suvw \text{ gdw } u \leq v \wedge w > v$$

$h_1(x, y) = x \cdot x, h_2(x, y) = y, h_3(x, y) = (x+1) \cdot (x+1)$

Fallunterscheidung mit primitiv rekursiven Relationen

6.18 Lemma

$R_i \subseteq \mathbb{N}^n$ $1 \leq i \leq m$ sind paarweise disjunkte primitiv rekursive Relationen und $h_1, \ldots, h_{m+1}$ n-stellige Funktionen mit $h_i \in \mathcal{P}(\mathbb{N})$ $i = 1, \ldots, m+1$.

Dann gilt für $f : \mathbb{N}^n \to \mathbb{N}$ mit

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

$f \in \mathcal{P}(\mathbb{N})$.

Beweis:

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

6.19 Beispiel Maximum und Minimum von (n)-zwei Zahlen

$$\max(x, y) = \begin{cases} x & \text{falls } x > y \\ y & \text{sonst} \end{cases}$$

Weitere Abschlusseigenschaften von $\mathcal{P}(\mathbb{N})$

6.20 Definition

1. Die **beschränkte Summation** und **beschränkte Multiplikation** mit der Funktion $g : \mathbb{N}^{n+1} \to \mathbb{N}$ sind erklärt durch

$$\begin{aligned}f, h : \mathbb{N}^{n+1} &\to \mathbb{N} \\ f(\vec{x}, y) &= \sum_{z \leq y} g(\vec{x}, z) \qquad h(\vec{x}, y) = \prod_{z \leq y} g(\vec{x}, z)\end{aligned}$$

2. Die **beschränkte Minimierung** mit der Funktion $g : \mathbb{N}^{n+1} \to \mathbb{N}$ ist erklärt durch die Funktion $f : \mathbb{N}^{n+1} \to \mathbb{N}$ mit

$$f(\vec{x}, y) = \begin{cases} u & u \leq y,\ g(\vec{x}, u) = 0 \text{ und } g(\vec{x}, z) > 0 \text{ für } z < u \\ 0 & g(\vec{x}, z) > 0 \text{ für alle } z \leq y \\ \uparrow & \text{es gibt } u \leq y \text{ mit } g(\vec{x}, u)\uparrow,\ g(\vec{x}, z) > 0 \text{ für } z < u \end{cases}$$

   Bezeichnung $f(\vec{x}, y) = \mu_{z \leq y}[g(\vec{x}, z) = 0]$

3. Die **beschränkte Minimierung** mit einer Relation $R \subseteq \mathbb{N}^{n+1}$ ist erklärt durch die Funktion $f : \mathbb{N}^{n+1} \to \mathbb{N}$ mit

$$f(\vec{x}, y) = \begin{cases} \text{kleinstes } z \text{ mit } z \leq y \wedge R\vec{x}z & \text{falls } z \text{ existiert} \\ y & \text{sonst} \end{cases}$$

   Schreibe $f(\vec{x}, y) = \mu z \leq y.R\vec{x}z$ $\qquad$ f **ist total.**

Beispiele

6.21 Beispiel

1. Mit $g(y) = y + 1$ ergibt sich

$$\begin{aligned} f(y) &= \sum_{z \le y} g(z) = \frac{(y+1)\cdot(y+2)}{2} \\ h(y) &= \prod_{z \le y} g(z) = fac(y+1) \end{aligned}$$

Mit $g(x, y) = x$ ergibt sich

$$\begin{aligned} f(x,y) &= \sum_{z \le y} g(x,z) = x \cdot (y+1) \\ h(x,y) &= \prod_{z \le y} g(x,z) = x^{y+1} \end{aligned}$$

2. Sei $f(x, y) = \mu_{z \le y}[g(x, z) = 0]$.
Wir betrachten zwei Funktionen g:

$g(x, y) = x - y$ liefert $f(x, y) = \begin{cases} 0 & x > y \\ x & x \le y \end{cases}$

$g(x, y) = \begin{cases} x \cdot y & x + y > 0 \\ \uparrow & \text{sonst} \end{cases}$ liefert $f(x, y) = \begin{cases} 0 & x > 0 \\ \uparrow & x = 0 \end{cases}$

3. Beschränkte Minimierung mit einer Relation
   - $f(x) =$ ganzzahliger Anteil der Quadratwurzel von x
     $= \mu y \le x.(y \cdot y \le x \wedge (y+1) \cdot (y+1) > x)$
   - $x \text{ div } y = \begin{cases} \mu t \le x.(t+1) \cdot y > x & \text{falls } y > 0 \\ 0 & \text{sonst} \end{cases}$
   - $x \bmod y = \begin{cases} x - (x \text{ div } y) \cdot y & \text{falls } y > 0 \\ 0 & \text{sonst} \end{cases}$

6.22 Lemma $\mathcal{P}(\mathbb{N})$ ist abgeschlossen bezüglich beschränkter Summation, beschränkter Multiplikation und den beschränkten Minimierungen.

Beweis

1. Sei $g \in \mathcal{P}(\mathbb{N})$; zu zeigen ist $f, h \in \mathcal{P}(\mathbb{N})$, wobei

$$\begin{aligned} f(x,y) &= \sum_{z \le y} g(x,z) \\ h(x,y) &= \prod_{z \le y} g(x,z) \end{aligned}$$

Es gilt

$$\begin{array}{lllllll} f(x,0) & = & g(x,0) & f(x,y+1) & = & f(x,y) + g(x,y+1) \\ h(x,0) & = & g(x,0) & h(x,y+1) & = & h(x,y) \cdot g(x,y+1) \end{array}$$

also

$$\begin{array}{lll} f = R(g, h_1) & \text{mit} & h_1(x,y,z) = y + g(x, z+1) \\ h = R(g, h_2) & \text{mit} & h_2(x,y,z) = y \cdot g(x, z+1) \end{array}$$

Es gilt $g_i, h_i \in \mathcal{P}(\mathbb{N})$, also $f, h \in \mathcal{P}(\mathbb{N})$

2. Sei $g \in \mathcal{P}(\mathbb{N})$ und $f(x, y) = \mu_{z \le y}[g(x, z) = 0]$. Für den Nachweis, dass $f \in \mathcal{P}(\mathbb{N})$ gilt, muss f in der funktionalen Programmiersprache zu $\mathcal{P}(\mathbb{N})$ programmiert werden. Die Idee hierzu spiegelt die natürliche Berechnung von $f(x, y)$ wider: Wir bilden die Funktion

$$f_0(x,z) = \begin{cases} 1 & g(x,u) > 0 \text{ für alle } u \le z \\ 0 & \text{sonst} \end{cases}$$

und summieren die Werte $f_0(x, z)$ für $z = 0, 1, \ldots, y$ auf. Setze also

$$f_0(x,z) = sgn\left(\prod_{u \le z} g(x,u)\right)$$

dann gilt $f_0 \in \mathcal{P}(\mathbb{N})$ nach 1. und eine kurze Überlegung zeigt

$$f(x,y) = \begin{cases} \sum_{z \le y} f_0(x,z) & f_0(x,y) = 0 \\ 0 & \overline{sgn}(f_0(x,y)) = 0 \end{cases}$$

Also gilt $f \in \mathcal{P}(\mathbb{N})$ nach 1. und Lemma 6.12.

Der Beweis zur beschränkten Minimierung soll noch an folgendem Beispiel verdeutlicht werden. Sei

$$g(x,y) = x - y^2$$

Die folgende Tabelle verdeutlicht die Berechnung von $f(5, y)$

y	$g(5,y)$	$f_0(5,y)$	$f(5,y)$
0	5	1	0
1	4	1	0
2	1	1	0
3	0	0	3
4	0	0	3

3. Ist R primitiv rekursiv und f durch beschränkte Minimierung aus R wie eben definiert, so ist $f \in \mathcal{P}(\mathbb{N})$. Es gilt nämlich
$f(\vec{x}, 0) = 0$

$$f(\vec{x}, y+1) = \begin{cases} f(\vec{x}, y) & \text{falls } R\vec{x}f(\vec{x}, y) \\ y+1 & \text{sonst} \end{cases}$$

Beschränkte Quantifizierung

6.23 Definition

Sei $R \subseteq \mathbb{N}^{n+1}$. Definiere Relationen $T \subseteq \mathbb{N}^{n+1}$ und $S \subseteq \mathbb{N}^{n+1}$ durch **beschränkte All-/Existenz-Quantifizierung** durch

$S\vec{x}b$ gdw es gibt $y \leq b$ mit $R\vec{x}y$ (Schreibe $\exists y \leq b.R\vec{x}y$)

$T\vec{x}b$ gdw für alle $y \leq b$ gilt $R\vec{x}y$ (Schreibe $\forall y \leq b.R\vec{x}y$)

6.24 Lemma

Die primitiv rekursiven Relationen sind abgeschlossen gegenüber beschränkter Quantifizierung.

Beweis: Sei $S\vec{x}b$ gdw $\exists y \leq b.R\vec{x}y$

Dann gilt $\chi_S(\vec{x}, 0) = \chi_R(\vec{x}, 0)$
$\chi_S(\vec{x}, b+1) = \max(\chi_R(\vec{x}, b+1), \chi_S(\vec{x}, b))$

Wegen $T\vec{x}b$ gdw $\neg\exists y \leq b.\neg R\vec{x}y$ folgt die Behauptung.

6.25 Beispiel

Die Teilbarkeitsrelation | ist primitiv rekursiv.

- $x \mid y$ gdw $\exists t \leq y.t \cdot x = y$
  $Rxyz$ gdw $z \cdot x = y$
  Dann ist $x \mid y$ gdw $Sxyy$ gdw $\exists t \leq y.Rxyt$ gdw
  $\exists t \leq y.t \cdot x = y$
- $\{p : p$ ist Primzahl$\}$ ist primitiv rekursiv.

Primitiv rekursive Codier- und Decodierfunktionen

Paarungsfunktionen, Codierung von Zahlenfolgen

6.26 Definition

Die **Cauchysche Paarungsfunktion** $\langle\cdot,\cdot\rangle : \mathbb{N}^2 \to \mathbb{N}$ wird definiert durch

$$\langle x, y\rangle = ((x+y)(x+y+1) \text{ div } 2) + y$$

Sie ist primitiv rekursiv und bijektiv.

(x,y)	$(0,0)$	$(1,0)$	$(0,1)$	$(2,0)$	$(1,1)$	$(0,2)\cdots$
$\langle x,y\rangle$	0	1	2	3	4	$5\cdots$

Abzählen auf geeigneten Diagonalen: $x+y$ konstant.
(x,y) kommt auf der Diagonalen $x+y+1$ vor.

- $x+y$ komplett gefüllte Diagonalen: $1+2+\cdots+(x+y) = (x+y)(x+y+1) \text{ div } 2$
- In der Diagonalen, in der (x,y) steht, kommen noch y viele Punkte vor dem Punkt (x,y).

Also ist $\langle\cdot,\cdot\rangle : \mathbb{N}^2 \to \mathbb{N}$ bijektiv

Definiere folgende **Umkehrfunktionen** first, rest: $\mathbb{N} \to \mathbb{N}$ mit $\langle \text{first}(z), \text{rest}(z)\rangle = z$ und
$\text{first}(\langle x,y\rangle) = x$
$\text{rest}(\langle x,y\rangle) = y$

Paarungsfunktionen Codierung von Zahlenfolgen

6.27 Lemma Die Funktionen $\langle\cdot,\cdot\rangle : \mathbb{N}^2 \to \mathbb{N}$ und first, rest: $\mathbb{N} \to \mathbb{N}$ sind primitiv rekursiv.

Beweis:

Nach Def. von $\langle x,y\rangle = z$, folgt $x \leq z$ und $y \leq z$.

Dann

$$\text{first}(z) = \mu x \leq z.\exists y \leq z.\langle x,y\rangle = z$$

$$\text{rest}(z) = \mu y \leq z.\exists x \leq z.\langle x,y\rangle = z$$

Nach Lemma 6.24 und Lemma 6.22 sind first und rest primitiv rekursiv.

Codierung endlicher Zahlenfolgen

6.28 Definition Sei $x_0, \ldots, x_n$ Zahlenfolge. Die **Codierung** $[x_0, \ldots, x_n] \in \mathbb{N}$ wird induktiv über n definiert durch

- $[\,] = 0$ (Codierung der leeren Folge)
- $[x_0, \ldots, x_n] = \langle x_0, [x_1, \ldots, x_n]\rangle$

Die **Folgenzugriffsfunktion** get : $\mathbb{N}^2 \to \mathbb{N}$ sei definiert durch

- $\text{get}(z, 0) = \text{first}(z)$
- $\text{get}(z, i+1) = \text{get}(\text{rest}(z), i)$

Die Folgenzugriffsfunktion liegt in $\mathcal{P}(\mathbb{N})$.

Wertverlaufsrekursion

Beachte: Die Folgenkodierung ist eindeutig bis auf Nullen am rechten Folgenende, d. h. es gilt $[x_0, \ldots, x_n] = [x_0, \ldots, x_n, 0, \ldots, 0]$. Die Elemente x_0 bis x_n mit $x_n \neq 0$ können eindeutig bestimmt werden.

6.29 Lemma Wertverlaufsrekursion

Sind $g : \mathbb{N}^{n+1} \to \mathbb{N}$ und $h : \mathbb{N}^{n+2} \to \mathbb{N}$ primitiv rekursiv, dann auch $f : \mathbb{N}^{n+1} \to \mathbb{N}$ mit

- $f(\vec{x}, 0) = g(\vec{x}, 0)$
- $f(\vec{x}, y+1) = h(\vec{x}, [f(\vec{x}, y), \ldots, f(\vec{x}, 0)], y)$

Hier greift die Rekursion auf beliebig viele Vorgängerwerte zurück.

Beweis:

Die Hilfsfunktion $F(\vec{x}, y) = [f(\vec{x}, y), \ldots, f(\vec{x}, 0)]$ ist primitiv rekursiv, da

- $F(\vec{x}, 0) = \langle g(\vec{x}, 0), 0\rangle$
- $F(\vec{x}, y+1) = \langle h(\vec{x}, F(\vec{x}, y), y), F(\vec{x}, y)\rangle$

und somit ist

$f(\vec{x}, y) = \text{get}(F(\vec{x}, y), 0) = \text{first}(F(\vec{x}, y))$

primitiv rekursiv.

Beispiel

6.30 Beispiel

1. Sei f definiert durch
   - $f(x,0) = 1$
   - $f(x,y+1) = f(x, y \text{ div } 2) + 1$

   Dann ist $f \in \mathcal{P}(\mathbb{N})$: Wähle im obigen Lemma $h(x,u,y) = succ(\text{get}(u, y \text{ div } 2))$.
2. **Folgenverkettung $*$: $\mathbb{N}^2 \longrightarrow \mathbb{N}$**
   (nicht Multiplikation von Zahlen!)
   $[x_0,\ldots,x_n] * [y_0,\ldots,y_m] = [x_0,\ldots,x_n,y_0,\ldots,y_m]$
   wobei $x_n \neq 0$ (falls $n > 0$).

   Behauptung: $*$ ist primitiv rekursiv.
   Beweis: Betrachte

   $$0 * v = v$$
   $$(u+1) * v = \langle \text{first}(u+1), \text{rest}(u+1) * v \rangle$$

   wegen $\text{rest}(u+1) \leq u$ folgt die Behauptung durch Wertverlaufsrekursion (hier wird die Voraussetzung $x_n \neq 0$ benötigt).
   Es gilt nämlich:

$$[i] = \langle i, 0 \rangle = \underbrace{\frac{i \cdot (i+1)}{2}}_{\geq i} > 0 \quad (i \neq 0)$$

Beispiel (Fort.)

$$\underbrace{[x_0,\ldots,x_n]}_{u+1, x_n \neq 0} = \langle x_0, [x_1,\ldots,x_n] \rangle =$$

$$\underbrace{\frac{(x_0 + [x_1 \ldots x_n])(x_0 + [x_1 \ldots x_n] + 1)}{2}}_{\geq 1} + [x_1,\ldots,x_n]$$

D. h.

- $\text{first}(u+1) = x_0 < u$ (Listenlänge ≥ 2)
- $\text{rest}(u+1) = [x_1,\ldots,x_n] \leq u$

Wir werden diese Funktionen noch benötigen, und zwar bei der Arithmetisierung der While-Programme. Programme sind Folgen von Anweisungen. Wir werden Anweisungen durch Zahlen codieren und dementsprechend Programme durch die Codierungen der Zahlenfolgen darstellen. Um die Interpreterfunktion zu simulieren benötigen wir die Folgenzugriffsfunktion und die Folgenverkettung.

Simultane Rekursion

6.31 Definition

Eine Funktion $f = (f_1,\ldots,f_m) : \mathbb{N}^n \longrightarrow \mathbb{N}^m$ heißt primitiv rekursiv, falls jede Komponentenfunktion $f_i : \mathbb{N}^n \longrightarrow \mathbb{N}$ $i = 1,\ldots,m$ primitiv rekursiv ist.

6.32 Lemma Simultane Rekursion

Sind $g : \mathbb{N}^{n+1} \longrightarrow \mathbb{N}^m$ und $h : \mathbb{N}^{n+m+1} \longrightarrow \mathbb{N}^m$ primitiv rekursiv, dann ist auch die Funktion $f : \mathbb{N}^{n+1} \longrightarrow \mathbb{N}^m$ mit

- $f(\vec{x},0) \rightrightarrows g(\vec{x},0)$
- $f(\vec{x},y+1) \rightrightarrows h(\vec{x}, f(\vec{x},y), y)$

($\rightrightarrows$ ist als Gleichheit von Vektoren zu lesen).

Beweis:

Die Hilfsfunktion

$$F(\vec{x},y) = [f_1(\vec{x},y),\ldots,f_m(\vec{x},y)]$$

ist primitiv rekursiv.

Simultane Rekursion (Forts.)

Da $F(\vec{x},0) = [g_1(\vec{x},0),\ldots,g_m(\vec{x},0)]$

und

$$F(\vec{x},y+1) = [h_1(\vec{x},F(\vec{x},y),y),\ldots,h_m(\vec{x},F(\vec{x},y),y)]$$

und für festes m die Funktion $[k_1,\ldots,k_m] : \mathbb{N}^n \longrightarrow \mathbb{N}$ primitiv rekursiv ist für $k_i : \mathbb{N}^n \longrightarrow \mathbb{N}, k_i \in \mathcal{P}(\mathbb{N})$.

Aus $f_i(\vec{x},y) = \text{get}(F(\vec{x},y), i)$ $i = 1,\ldots,m$ folgt die Behauptung.

6.33 Beispiel

Sei $paar : \mathbb{N} \longrightarrow \mathbb{N}^2$ die Umkehrung der Cauchyschen Paarungsfunktion. Es gilt

$paar(0) = (0,0)$ und

$$paar(n+1) = \begin{cases} (y+1,0) & \text{falls } paar(n) = (0,y) \\ (x-1,y+1) & \text{sonst, wobei } paar(n) = (x,y) \end{cases}$$

Zeige: $paar(n) = (\text{first}(n), \text{rest}(n))$.

Sind alle totalen „berechenbaren“ Funktionen primitiv rekursiv?

- Numeriere effektiv alle primitiv rekursiven Ausdrücke, z. B. lexikographische Anordnung der Wörter über dem Alphabet $\{NULL, SUCC, PROJ, KOMP, REK, (,), 0, \ldots, 9, , \}$
- Sei π_i der i-te primitiv rekursive Ausdruck in dieser Nummerierung (Länge + Lexikographisch).
- Definiere Diagonalfunktion $d : \mathbb{N} \to \mathbb{N}$ durch

$$d(n) = f^{(1)}_{\pi_n}(n) + 1$$

Dann ist d total und „effektiv berechenbar“.

6.34 Satz Diagonalschluss für primitiv rekursive Funktionen

Die Funktion d ist nicht primitiv rekursiv.

Beweis:

Wäre d primitiv rekursiv, dann gäbe es einen primitiv rekursiven Ausdruck π_n, so dass $d = f^{(1)}_{\pi_n}$. Dann aber

$f^{(1)}_{\pi_n}(n) = d(n) = f^{(1)}_{\pi_n}(n) + 1$ ↯

Universelle Funktionen für $\mathcal{P}(\mathbb{N})$

6.35 Folgerung Aufzählung der einstelligen Funktionen in $\mathcal{P}(\mathbb{N})$

Es gibt keine **universelle** primitiv rekursive Funktion $f : \mathbb{N}^2 \to \mathbb{N}$ für $\mathcal{P}(\mathbb{N})$, d. h. eine Funktion $f \in \mathcal{P}(\mathbb{N})$ mit der Eigenschaft $\forall g \in \mathcal{P}(\mathbb{N}) \exists i \in \mathbb{N} : f(i, \cdot) = g(\cdot)$ (i heißt **Index** für g).

Beweis: Sei $f : \mathbb{N}^2 \to \mathbb{N}$ universelle Funktion für $\mathcal{P}(\mathbb{N})$.

Sei $d'(x) = f(x, x) + 1$.

Angenommen $f \in \mathcal{P}(\mathbb{N})$. Dann $d' \in \mathcal{P}(\mathbb{N})$ und es gibt $i \in \mathbb{N}$ mit $d'(x) = f(i, x)$ für $x \in \mathbb{N}$.

Insbesondere $f(i, i) = d'(i) = f(i, i) + 1$ ↯

Gesucht: Konkrete Funktionen, die nicht primitiv rekursiv sind.

6.36 Beispiel Ackermann Funktion

- $A(0, y) = y + 1$
- $A(x + 1, 0) = A(x, 1)$
- $A(x + 1, y + 1) = A(x, A(x + 1, y))$

A ist eine totale Funktion.

A ist „eine Art“ universelle Funktion für $\mathcal{P}(\mathbb{N})$.

Eigenschaften der Ackermann Funktion

1. $A(x, y) > y$.
2. $A(x, y_1) > A(x, y_2)$, falls $y_1 > y_2$ (Monotonie).
3. $A(x + 1, y) \geq A(x, y + 1)$.
4. $A(x + 2, y) > A(x, 2y)$.
5. A ist total. (wie zeigt man dies!)
6. A ist effektiv berechenbar.

```
proc AKM(in x, y : nat, out z : nat)
  {true}call AKM(x, y, z){z = A(x, y)}
  begin
    if x = 0
      then z := y + 1;
      else
          if y = 0
            then call AKM(pred(x), 1, z);
            else
                 call AKM(x, pred(y), z_1);
                 call AKM(pred(x), z_1, z);
          end;
    end;
  end.
```

Die Prozedur „sollte“ korrekt kommentiert sein! (Nachweis!) über $(Nat, \ldots, A)$.

Eigenschaften der Ackermann Funktion (Forts.)

7. Für jede primitiv rekursive Funktion $f : \mathbb{N}^n \to \mathbb{N}$ gibt es ein $r \in \mathbb{N}$, so dass

$$f(\vec{x}) \leq A(r, \max\{x_1, \ldots, x_n\})$$

Beweis: Induktion über Aufbau von $\mathcal{P}(\mathbb{N})$.

- Grundfunktionen: wähle $r = 0$
  0
  $x_1 + 1 \quad \leq A(0, \max\{x_1, \ldots, x_n\}) = \max\{x_1, \ldots, x_n\} + 1$
  x_i
- Sei $f = g \circ (h_1, \ldots, h_m)$ und $y = \max\{x_1, \ldots, x_n\}$.
  wobei für g durch r, h_i durch s_i beschränkt sei, d. h.
  $g(\vec{x}) \leq A(r, y)$, $h_i(\vec{x}) \leq A(s_i, y)$.

$$\begin{aligned} f(\vec{x}) &= g \circ (h_1, \ldots, h_m)(\vec{x}) = g(h_1(\vec{x}), \ldots, h_m(\vec{x})) = \\ &\leq A(r, \max\{h_1(\vec{x}), \ldots, h_m(\vec{x})\}) \\ &\overset{(2)}{\leq} A(r, \max\{A(s_1, y), \ldots, A(s_m, y)\}) \\ &\overset{(3)}{=} A(r, A(\max\{s_1, \ldots, s_m\}, y)) \\ &\overset{(2)(3)}{\leq} A(\max\{s_1, \ldots, s_m, r\}, A(\max\{s_1, \ldots, s_m, r\} \\ &\qquad +1, y)) \\ &\overset{def}{=} A(\max\{s_1, \ldots, s_m, r\} + 1, y + 1) \\ &\overset{(3)}{\leq} A(\max\{s_1, \ldots, s_m, r\} + 2, y) \end{aligned}$$

Wähle $\max\{s_1, \ldots, s_m, r\} + 2$ als Konstante für f.

Eigenschaften der Ackermann Funktion (Forts.)

- Sei $f = R(g, h)$.
  g sei durch r, h durch s beschränkt.
  Hilfsbehauptung:
  $f(\vec{x}, z) \leq A(\max\{r, s\} + 1, y + z)$ mit
  $y = \max\{x_1, \ldots, x_n\}$.
  Beweis: Induktion über z. (einfach).
  Setze $p := \max\{r, s\} + 1$. Dann

$$A(p, y+z) \overset{(2)}{\leq} A(p, 2\max\{y, z\}) \overset{(4)}{\leq} A(p+2, \max\{y, z\})$$

  $p + 2$ kann als Konstante für f gewählt werden.

8. Die Ackermannfunktion ist nicht primitiv rekursiv, d. h.
   $A \notin \mathcal{P}(\mathbb{N})$.

Beweis:

Angenommen $A \in \mathcal{P}(\mathbb{N})$. Dann ist f mit $f(x, y) = A(x, y) + 1$ primitiv rekursiv. Es gibt (wegen 7.) ein $r \in \mathbb{N}$ mit
$f(x, y) = A(x, y) + 1 \leq A(r, \max\{x, y\})$.
Insbesondere für $x = y = r$

$$f(r, r) = A(r, r) + 1 \leq A(r, r) \qquad \lightning$$

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 $f_\pi^{(n)} : \mathbb{N}^n \to \mathbb{N}$ 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 $f : \mathbb{N}^n \to \mathbb{N}$ 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 $\chi_R : \mathbb{N}^n \to \mathbb{N}$ 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 $f_1, \ldots, f_n : \mathbb{N}^m \to \mathbb{N}$ **totale** μ-rekursive Funktionen (z.B. wenn die f_i primitiv rekursiv sind).

Dann ist auch $R \subseteq \mathbb{N}^m$ mit
$R\vec{x}$ 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:

$R(\vec{x}, y)$ gdw $(\vec{x}, y) \in graph(f)$
gdw $f(\vec{x}) = y$

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 $f : \mathbb{N}^n \to \mathbb{N}$ 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 $R \subseteq \mathbb{N}^{n+1}$ 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.

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

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

μ-rekursive Funktionen

- Fallunterscheidung
- Beschränkte Minimierung

Minimierung

- Reduktion

totale

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

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

entscheidbare Relationen

$\cup$

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

- $\neg, \wedge, \vee$
- beschränkte Quantifizier.

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**:

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

  $\alpha_{SUCC}^{(n)}$ ist das Programm $Y_{SUCC} := succ(X_{SUCC}^1);$

  $\alpha_{PROJ(i)}^{(n)}$ ist das Programm $Y_{PROJ(i)} := X_{PROJ(i)}^i \quad i \leq n$
  $Y_{PROJ(i)} := 0 \quad i > n$

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

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

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

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

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

Programmierbarkeit der μ-rekursiven Funktionen

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

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

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

$I := 0;$
$\vec{X}_G := (\vec{X}_F, I);\ \alpha_G^{(n+1)} \quad \{Y_G = g(\vec{X}, 0)\}$
while $\neg Y_G = 0$ **do**
  $I := succ(I);$
  $\vec{X}_G := (\vec{X}_F, I);\ \alpha_G^{(n+1)} \quad \{Y_G = g(\vec{X}, I)\}$
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 $z[\![\alpha]\!]_A z'$ 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

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

Vereinbarungen:

- Variablen sind aus Menge $Var = \{V_0, V_1, \ldots\}$ 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:

- $code(\varepsilon) = 0$
- $code(0) = [1]$
- $code(V_i) = [2, i]$
- $code(succ(t)) = [3, code(t)]$
- $code(s = t) = [4, code(s), code(t)]$
- $code(\neg B) = [5, code(B)]$
- $code(B \wedge C) = [6, code(B), code(C)]$
- $code(V_i := t;\ ) = [7, i, code(t)]$
- $code($**if** B **then** β **else** γ **end**$;\ ) = [8, code(B), code(\beta), code(\gamma)]$
- $code($**while** B **do** β **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**
  $V_0 := succ(V_0);$
  $V_3 := succ(V_3);$
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** $\tau : \mathbb{N}^2 \to \mathbb{N}$, 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

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

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** $\beta : \mathbb{N}^2 \to \mathbb{N}$, 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): $\sigma : \mathbb{N}^3 \to \mathbb{N}$, 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, $i : \mathbb{N} \to \mathbb{N}$, 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:

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

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

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

Simulation der Speicherinitialisierung und Ausgabefunktion

- **Speicherinitialisierung** für Programm mit Code p, $\mathsf{inp}^{(n)} : \mathbb{N}^{n+1} \to \mathbb{N}$.
  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, $\mathsf{out} : \mathbb{N} \to \mathbb{N}$
  $$\mathsf{out}(\langle p, x\rangle) = x[0]$$
- $\mathsf{inp}^{(n)}$ 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 $\Phi^{(n)} : \mathbb{N}^{n+1} \to \mathbb{N}$ mit

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

Φ heißt **Zeitkomplexitätsfunktion.**

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

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

$\varphi^{(n)}(p, x_1, \ldots, x_n) \downarrow$ 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 $f : \mathbb{N}^2 \to \mathbb{N}$ mit $f \in \mathcal{R}_p(\mathbb{N})$, so dass es für jede μ-rekursive Funktion $g : \mathbb{N} \to \mathbb{N}$ 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.

$code(call \dots)$

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)

6.4 Grundzüge der Rekursionstheorie

Folgerungen aus der Existenz universeller Funktionen

$\Phi^{(n)} : \mathbb{N}^{n+1} \to \mathbb{N}$	Zeitkomplexitätsfunktion
$\varphi^{(n)} : \mathbb{N}^{n+1} \to \mathbb{N}$	Universelle Funktion

6.53 Lemma Eigenschaften von $\Phi^{(n)}$ bzw. $\varphi^{(n)}$:

$\Phi^{(n)}$ und $\varphi^{(n)}$ haben denselben Definitionsbereich.

Die Relationen **Beschränkte Laufzeit** $BLZ \subset \mathbb{N}^{n+2}$

$BLZ = \{(p, \vec{x}, b) \in \mathbb{N}^{n+2} : \Phi^{(n)}(p, \vec{x}) \leq b\}$

und **Beschränkte Berechenbarkeit** $BBER \subset \mathbb{N}^{n+3}$

$BBER =$

$\{(p, \vec{x}, b, y) \in \mathbb{N}^{n+3} : \Phi^{(n)}(p, \vec{x}) \leq b \wedge \varphi^{(n)}(p, \vec{x}) = y\}$

sind primitiv rekursiv.

Beweis:

Definitionsbereiche gleich folgt aus Definition von $\Phi^{(n)}$ bzw. $\varphi^{(n)}$.

Relationen primitiv rekursiv, da

$$BLZ = \exists t \leq b.\ \underbrace{\mathsf{first}(i^t(\mathsf{inp}^{(n)}(p, \vec{x}))) = 0}_{rel(p,\vec{x},t)\ \text{primitiv-rekursiv}}$$

bzw.

$$BBER = \exists t \leq b.\ \underbrace{(\mathsf{first}(i^t(\mathsf{inp}^{(n)}(p, \vec{x}))) = 0 \wedge \mathsf{out}(i^t(\mathsf{inp}^{(n)}(p, \vec{x}))) = y)}_{rel(p,\vec{x},t,y)\ \text{primitiv-rekursiv}}$$

Folgerungen

6.54 Satz s-m-n Theorem

Zusammenhang zwischen universellen Funktionen. Zu jedem Paar $m, n \in \mathbb{N}$ mit $n \geq 1$ gibt es eine primitiv rekursive Funktion $s_{m,n} : \mathbb{N}^{m+1} \to \mathbb{N}$, so dass für alle $p \in \mathbb{N}, \vec{x} \in \mathbb{N}^n$ und $\vec{y} \in \mathbb{N}^m$ gilt:

$$\varphi^{(n+m)}(p, \vec{x}, \vec{y}) \quad = \varphi^{(n)}(s_{m,n}(p, \vec{y}), \vec{x}) \quad = \varphi^{(n)}_{s_{m,n}(p,\vec{y})}(\vec{x})$$

Beweis:

In Anhängigkeit von $\vec{y}$ ist der Programmcode p so abzuändern, dass die Werte $\vec{y}$ nicht über die Eingabe eingelesen, sondern im Programm zugewiesen werden. Hierbei ist zu beachten, dass die Eingabezahlen durch geeignete Terme dargestellt werden.

Ist $p = code(\alpha)$ muss $s_{m,n}(p, \vec{y})$ der Code zum Programm

$V_{n+1} := succ^{y_1}(0);\ \dots;\ V_{n+m} := succ^{y_m}(0);\ \alpha$ sein.

Sei

$s_{m,n}(p, \vec{y}) := [[7, n+1, h(y_1)], \dots, [7, n+m, h(y_m)]] * p$

Mit $*$ die Folgenverkettungsfunktion von Beispiel 6.30 und $h : \mathbb{N} \to \mathbb{N}$ wobei

$h(0) = [1]$ ($code$ von 0), $\quad h(y+1) = [3, h(y)]$,

d. h. $h(y) = code(succ^y(0))$

Folgerungen - Programmtransformatoren

Beachte:

Bei obiger Situation gilt auch für beliebige $n \in \mathbb{N}$ und l mit $1 \leq l \leq m$ die Aussage

$\varphi^{(n+m)}(p, \vec{x}, \vec{y}) = \varphi^{(n+l)}(s_{m,n}(p, \vec{y}), \vec{x}, \vec{z})$

für alle $\vec{z} \in \mathbb{N}^l$.

6.55 Lemma

Es gibt eine primitiv rekursive Funktion $compose : \mathbb{N}^2 \to \mathbb{N}$, so dass für alle $p, q \in \mathbb{N}$ gilt

$$\varphi_{compose(p,q)}(x) = \varphi_p(\varphi_q(x))$$

Beweis:

Die Funktion $f(x, p, q) = \varphi_p(\varphi_q(x))$ ist μ-rekursiv. Sei a Index zu einem f berechnenden Programm. D. h.

$$\varphi^{(3)}_a(x, p, q) = f(x, p, q) \text{ für } (x, y, q) \in \mathbb{N}^3 = \varphi^{(1)}_{s_{2,1}(a,p,q)}(x)$$

D. h. $compose(p, q) = s_{2,1}(a, p, q)$ ist primitiv rekursiv.

Lässt sich auf andere Operationen übertragen:

Primitive Rekursion, Beschränkte Minimierung, Minimierung, Wertverlaufsrekursion.

Rekursiv aufzählbare Relationen

Wir haben bisher Relationen betrachtet, die entweder primitiv rekursiv oder rekursiv entscheidbar waren. Eine weitere Klasse von Relationen sind die effektiv aufzählbaren Relationen.

6.56 Definition

Eine Relation $R \subseteq \mathbb{N}^n$ heißt **rekursiv aufzählbar**, wenn es eine berechenbare (μ-rekursive) Funktion f mit Definitionsbereich R gibt, d. h.

$R \subseteq \mathbb{N}^n$ rekursiv aufzählbar $\mathrm{gdw}\ \exists f \in \mathcal{R}_p(\mathbb{N}) : dom(f) = R$
also $(f(\vec{x}) \downarrow\ \mathrm{gdw}\ R\vec{x}\ \ (x \in \mathbb{N}^n))$

6.57 Lemma

$R \subseteq \mathbb{N}^n$ ist entscheidbar $\mathrm{gdw}\ R$ und $\neg R$ rekursiv aufzählbar.

Beweis:

Sei R entscheidbar. Dann ist auch $\neg R$ entscheidbar.

R ist Definitionsbereich der Funktion

$$f(\vec{x}) = \begin{cases} 1 & \text{falls } R\vec{x} \\ \uparrow & \text{sonst} \end{cases}$$

$f \in \mathcal{R}_p(\mathbb{N})$ (Beachte dabei $\uparrow\ (x) = \uparrow$ alle x ist in $\mathcal{R}_p(\mathbb{N})$).

R.a. Relationen - Charakterisierungen

Seien R und $\neg R$ rekursiv aufzählbar und a bzw. b Programmindizes von Funktionen, die als Definitionsbereich R und $\neg R$ haben.

Sei

$$f(\vec{x}) = \mu t.(\Phi_a^{(n)}(\vec{x}) \leq t \vee \Phi_b^{(n)}(\vec{x}) \leq t)$$

$f \in \mathcal{R}_p(\mathbb{N})$ und total, da jedes $\vec{x}$ entweder in $R = dom(\varphi_a^{(n)})$ oder in $\neg R = dom(\varphi_b^{(n)})$ liegt.
Weiterhin ist $R\vec{x}\ \ \mathrm{gdw}\ \ \Phi_a^{(n)}(\vec{x}) \leq f(\vec{x})$ entscheidbar.

6.58 Lemma R.a. Relationen und entscheidbare Relationen
$R \subseteq \mathbb{N}^n$ ist rekursiv aufzählbar gdw es gibt eine entscheidbare Relation $S \subseteq \mathbb{N}^{n+1}$ mit $\quad R\vec{x}\ \ \mathrm{gdw}\ \ \exists y\ S\vec{x}y.$

Beweis:

- Sei R rekursiv aufzählbar. Sei a Index mit $R = dom(\varphi_a^{(n)})$. Dann gilt

  $$R\vec{x}\ \ \mathrm{gdw}\ \ \varphi_a^{(n)}(\vec{x}) \downarrow\ \mathrm{gdw}\ \ \exists y\ \Phi_a(\vec{x}) \leq y$$

  Die Relation $S\vec{x}y\ \mathrm{gdw}\ \Phi_a(\vec{x}) \leq y$ ist entscheidbar (sogar primitiv rekursiv entscheidbar).
- Gelte $R\vec{x}\ \mathrm{gdw}\ \exists y\ S\vec{x}y$ mit S entscheidbar. Dann ist $f(\vec{x}) = \mu y.S\vec{x}y$ berechenbar und hat als Definitionsbereich genau R.

(Lemma gilt auch, wenn entscheidbar durch primitiv rekursiv ersetzt wird.)

Zusammenhänge

6.59 Lemma

$f : \mathbb{N}^n \to \mathbb{N}$ ist berechenbar gdw der Graph von f rekursiv aufzählbar ist. D. h.

$graph(f) = \{(\vec{x}, y) \in \mathbb{N}^{n+1} : f(\vec{x}) \downarrow \wedge f(\vec{x}) = y\}$ rekursiv aufzählbar.

Beweis:

• Sei $f = \varphi_a^{(n)}$. Dann gilt für alle $\vec{x}, y$.

$$(\vec{x}, y) \in graph(f)\ \mathrm{gdw}\ \exists b \underbrace{(\Phi_a^{(n)}(\vec{x}) \leq b \wedge \varphi_a^{(n)}(\vec{x}) = y)}_{S\vec{x}yb\ \text{entscheidbar nach 6.53}}$$

also rekursiv aufzählbar nach Lemma 6.58.

• Sei $graph(f)$ rekursiv aufzählbar: $graph(f) = dom(\varphi_b^{(n+1)})$, dann gilt

$$f(\vec{x}) = \mathrm{first}(\mu y.(\Phi_b^{(n+1)}(\vec{x}, \mathrm{first}(y)) \leq \mathrm{rest}(y)))$$

Dafür betrachte man $y = \langle z, t\rangle$ als Codierung eines Paares (z, t). Ist $(\vec{x}, z) \in graph(f)$, so gibt es ein t mit $\Phi_b^{(n+1)}(\vec{x}, z) = t$ und der μ-Operator liefert y mit $y = \langle z, t\rangle$. Für $f(\vec{x}) = \uparrow$ gibt es kein solches y.

Folgerungen - Weitere Charakterisierungen

6.60 Lemma

Sei f eine totale Funktion. Dann gilt $f \in \mathcal{R}_p(\mathbb{N})\ \mathrm{gdw}\ graph(f)$ ist entscheidbar.

Beweis:

Sei f total, berechenbar. Dann ist die Menge $\{(\vec{x}, y) : f(\vec{x}) = y\}$ entscheidbar, d. h. $graph(f)$ ist entscheidbar (siehe auch Beispiel 6.44).

Ist $graph(f)$ entscheidbar, so ist auch $graph(f)$ rekursiv aufzählbar und somit f berechenbar.

Einfacher: $f(\vec{x}) = \mu y.\chi_{graph(f)}(x, y) = 1$.

6.61 Lemma Effektive Aufzählungen r.a. Relationen

$R \subseteq \mathbb{N}^n$ ist rekursiv aufzählbar $\mathrm{gdw}\ R = \varnothing$ oder es gibt totale berechenbare Funktionen $f_1, \ldots, f_n : \mathbb{N} \to \mathbb{N}$ mit
$R = \{(f_1(i), \ldots, f_n(i)) : i \in \mathbb{N}\}$.

Man kann also R mit Hilfe der Funktionen f_i „effektiv“ aufzählen.

Ist $n = 1$, so ist $R = im(f_1)$, d. h. R ist Bild einer totalen berechenbaren Funktion.

Weitere Charakterisierungen (Fort.)

Beweis:

Sei $R \subseteq \mathbb{N}^n$ rekursiv aufzählbar, $R \neq \varnothing$. Sei $R = dom(\varphi_a^{(n)})$, $\vec{b} \in R$ fest. Definiere Funktionen $f_i \quad i = 1, ..., n$ durch

$$f_i(x) = \begin{cases} x[i] & \text{falls } \Phi_a^{(n)}(x[1], \ldots, x[n]) \leq x[0] \\ b_i & \text{sonst} \end{cases}$$

Durchläuft x alle natürlichen Zahlen, so werden alle möglichen Parameter für $\Phi_a^{(n)}$ durchlaufen. Die Relation $\Phi_a^{(n)}(x[1], \ldots, x[n]) \leq x[0]$ ist nach Lemma 6.53 primitiv rekursiv. Somit sind die f_i primitiv rekursiv, also total, und liefern alle Werte von R.

Umgekehrt ist $R = \varnothing$, so ist R rekursiv aufzählbar als Definitionsbereich der nirgends definierten Funktion. Sei also $R = \{(f_1(i), \ldots, f_n(i)) : i \in \mathbb{N}\}$ $f_i \in \mathcal{R}_p(\mathbb{N})$ totale Funktionen, dann liefert

$$R\vec{x} \text{ gdw } \exists i(x_1 = f_1(i) \wedge \cdots \wedge x_n = f_n(i))$$

R ist rekursiv aufzählbar nach Lemma 6.58.

Abschlusseigenschaften r.a. Relationen

6.62 Lemma Abschlusseigenschaften

Die r.a. Relationen sind

a) Abgeschlossen gegen **existentielle Quantifizierung**
$R \subseteq \mathbb{N}^{n+1}$ $n \geq 1$ sei rekursiv aufzählbar. Dann auch S mit

$$S\vec{x} \text{ gdw } \exists y R\vec{x}y$$

b) Abgeschlossen gegen **Vereinigung und Durchschnitt**
Sind $R, S \subseteq \mathbb{N}^n$ rekursiv aufzählbar. Dann sind auch $R \cup S$, $R \cap S$ rekursiv aufzählbar.

Beweis:

a) Sei R rekursiv aufzählbar. Also $R\vec{a} \text{ gdw } \exists z T\vec{a}z$ für eine entscheidbare Relation T. Dann ist $S\vec{x}$ gdw
$\exists y R\vec{x}y \text{ gdw } \exists y \exists z T\vec{x}yz \text{ gdw } \exists b\, T\vec{x}\text{first}(b)rest(b)$

b) Sei $R\vec{x}$ gdw $\exists y\ T\vec{x}y$ und $S\vec{x}$ gdw $\exists y\ V\vec{x}y$ mit T, V entscheidbare Relationen. Dann gilt
$(R \wedge S)\vec{x}$ gdw
$\exists y \exists z\ (T\vec{x}y \wedge V\vec{x}z)$ gdw $\exists u\ (T\vec{x}\text{first}(u) \wedge V\vec{x}rest(u))$
und
$(R \vee S)\vec{x}$ gdw
$\exists y \exists z\ (T\vec{x}y \vee V\vec{x}z)$ gdw $\exists u\ (T\vec{x}\text{first}(u) \wedge V\vec{x}rest(u))$

Zusammenfassung Charakterisierungen der r.a Relationen

Sei $R \subseteq \mathbb{N}^n$, dann sind äquivalent:

- $R \subseteq \mathbb{N}^n$ ist rekursiv aufzählbar.
- $R = dom(f)$ für ein $f \in \mathcal{R}_p(\mathbb{N})$.
- $R = \varnothing \vee R = im(f_1, \ldots, f_n)$ für $f_i \in \mathcal{R}(\mathbb{N})$.
- $R = \varnothing \vee R = im(f_1, \ldots, f_n)$ für $f_i \in \mathcal{P}(\mathbb{N})$.
- R endlich $\vee$ $R = im(f_1, \ldots, f_n)$ für $f_i \in \mathcal{R}(\mathbb{N})$ mit f_i injektiv.
- $R\vec{x} \text{ gdw } \exists y\ S\vec{x}y$ für eine entscheidbare Relation S.
- $R\vec{x} \text{ gdw } \exists y\ S\vec{x}y$ für eine primitiv rekursive Relation S.

R ist entscheidbar gdw R und $\neg R$ sind rekursiv aufzählbar.

Rekursiv aufzählbare Relationen sind abgeschlossen gegen $\cup, \cap, \exists$.

Was mit $\neg$ und $\forall$?

Weitere Funktionsdefinitionen

6.63 Lemma Funktiondefinition auf r.a. Relationen

Sei $g : \mathbb{N}^n \to \mathbb{N}$ berechenbar, $R \subseteq \mathbb{N}^n$ rekursiv aufzählbar. Dann ist auch

$$f(\vec{x}) = \begin{cases} g(\vec{x}) & \text{falls } R\vec{x} \\ \uparrow & \text{sonst} \end{cases}$$

berechenbar.

Beweis: Es gilt $f(\vec{x}) = sgn(succ(h(\vec{x}))) \cdot g(\vec{x})$, sofern $dom(h) = R$.

Da R rekursiv aufzählbar ist, gibt es ein $h \in \mathcal{R}_p(\mathbb{N})$ mit $dom(h) = R$, also auch $f \in \mathcal{R}_p(\mathbb{N})$.

Führt jede Definition einer Funktion durch Fallunterscheidung mit paarweise disjunkten r.a. Relationen wieder zu berechenbaren Funktionen?

Sind die r.a. Relationen eine echte Obermenge der entscheidbaren Relationen. Gibt es r.a. Relationen die nicht rekursiv entscheidbar sind?

Unentscheidbare Relationen

Wichtige Probleme

- Das **allgemeine Halteproblem:** $K_0 \subseteq \mathbb{N}^2$

$$K_0 = \{(a, x) \in \mathbb{N}^2 : \varphi_a(x) \downarrow\}$$

- Das **spezielle Halteproblem (Selbstanwendungsproblem):** $K \subseteq \mathbb{N}$

$$K = \{a \in \mathbb{N} : \varphi_a(a) \downarrow\}$$

- Sei $f : \mathbb{N} \to \mathbb{N}$ mit $f \in \mathcal{R}(\mathbb{N})$, d. h. total.
  $\mathbf{Spez}(f) = \{a \in \mathbb{N} : \varphi_a = f\}$ Menge der „Indizes von f"
- $\mathbf{\ddot{A}quiv} \subseteq \mathbb{N}^2$ Äquivalenz von Indizes
  $\ddot{A}quiv = \{(a, b) \in \mathbb{N}^2 : \varphi_a = \varphi_b\}$

Beachte: K_0 und K sind rekursiv aufzählbar. Wie stehen diese Probleme in Beziehung?

Erinnerung: $R \subseteq \mathbb{N}^n$, $S \subseteq \mathbb{N}^l$. S kann auf R **rekursiv reduziert** werden (schreibe $\mathbf{S \leq_m R}$), falls es totale berechenbare Funktionen $f_1, \ldots, f_n : \mathbb{N}^l \to \mathbb{N}$ gibt, so dass gilt

$$S\vec{x} \text{ gdw } Rf_1(\vec{x}) \cdots f_n(\vec{x})$$

Wir hatten gezeigt: Ist R entscheidbar, so auch S.

Eigenschaften der Many-one Reduzierbarkeit

6.64 Lemma Es gilt

- $\leq_m$ ist reflexiv und transitiv.
- Ist $S \leq_m R$, R entscheidbar, so ist S entscheidbar.
- Ist $S \leq_m R$, R rekursiv aufzählbar, so ist S rekursiv aufzählbar.
- $S \leq_m R$, so ist $\neg S \leq_m \neg R$.

Beweis:

Sei $R = dom(f)$, dann ist $S = dom(f \circ (f_1, \ldots, f_n))$.

Die anderen Eigenschaften sind leicht zu beweisen.

6.65 Lemma

Es gilt

• $K \leq_m K_0$ • $K \leq_m spez(f)$ • $Spez(succ) \leq_m \ddot{A}quiv$

Beweis:

• Es ist $a \in K$ gdw $(a, a) \in K_0$ $(f_1, f_2$ Identität auf $\mathbb{N})$.

• Sei $f : \mathbb{N} \to \mathbb{N}$ total berechenbar. Definiere

$$\psi(x, p) = \begin{cases} f(x) & \text{falls } p \in K \\ \uparrow & \text{sonst} \end{cases}$$

ψ ist berechenbar, also gibt es ein $a \in \mathbb{N}$ mit $\varphi_a^{(2)} = \psi$.

Rekursiv aufzählbare Relationen, die nicht entscheidbar sind

Nach Definition von ψ gilt

$p \in K \Rightarrow \varphi_a^{(2)}(x, p) = f(x)$

$p \notin K \Rightarrow \varphi_a^{(2)}(x, p) \uparrow$

Nach s-m-n Theorem gibt es primitiv rekursive Funktion

$g(p) = s_{1,1}(a, p) : \varphi_a^{(2)}(x, p) = \varphi_{g(p)}(x)$.

• Dann ist $p \in K$ gdw $g(p) \in Spez(f)$.

• Sei $a \in \mathbb{N}$ mit $\varphi_a = succ$:

$b \in Spez(succ)$ gdw $(a, b) \in \ddot{A}quiv$.

6.66 Satz

$K = \{a \in \mathbb{N} : \varphi_a(a) \downarrow\}$ ist nicht entscheidbar.

Beweis: Angenommen K wäre entscheidbar. Wende Diagonalisierungsargument an: Sei $f : \mathbb{N} \to \mathbb{N}$ mit

$$f(x) = \begin{cases} \varphi_x(x) + 1 & \text{falls } x \in K \\ 0 & \text{sonst} \end{cases}$$

$f \in \mathcal{R}_p(\mathbb{N})$ und f ist total. Es gibt einen Index $p \in \mathbb{N}$ für f, d. h. $f = \varphi_p$. Insbesondere $f(p) = \varphi_p(p)$. Dies ist ein Widerspruch, da

- Ist $p \in K$, so $\varphi_p(p) \downarrow$ und $f(p) = \varphi_p(p) + 1$ ↯
- Ist $p \notin K$, so $\varphi_p(p) \uparrow$ und $f(p) = 0$ ↯

Folgerungen

6.67 Folgerung Es gilt

- $K_0, Spez(f)(f \in \mathcal{R})$ und $\ddot{A}quiv$ sind unentscheidbar.
- $\neg K, \neg K_0$ sind nicht rekursiv aufzählbar.
- Die rekursiv aufzählbaren Relationen sind nicht abgeschlossen gegenüber Komplementbildung und Allquantifizierung.
- Die entscheidbaren Relationen sind nicht abgeschlossen gegenüber existentielle- und Allquantifizierung.

6.68 Lemma

Sei R rekursiv aufzählbar. Dann gilt $R \leq_m K_0$.

Beweis: Sei $R \subseteq \mathbb{N}^n$ rekursiv aufzählbar. $R = dom(\varphi_a^{(n)})$ für ein $a \in \mathbb{N}$.

Es gilt

$$\begin{aligned} \vec{x} \in R \quad &\text{gdw} \quad \varphi_a^{(n)}(\vec{x}) \downarrow \\ &\text{gdw} \quad \varphi_{s_{n-1,1}(a,x_2,\ldots,x_n)}(x_1) \downarrow \\ &\text{gdw} \quad (s_{n-1,1}(a, x_2, \ldots, x_n), x_1) \in K_0 \end{aligned}$$

Seien $f_1(x_1, \ldots, x_n) = s_{n-1,1}(a, x_2, \ldots, x_n)$ und $f_2(x_1, \ldots, x_n) = x_1$. $f_1, f_2 \in \mathcal{P}(\mathbb{N})$, also $R \leq_m K_0$.

Die Relation K_0 ist also die „schwerste" r.a. Relation.

Vollständige rekursiv aufzählbare Relationen

6.69 Definition

Eine Relation $S \subseteq \mathbb{N}^n$ heißt vollständig bzgl. $\leq_m$, falls S rekursiv aufzählbar ist, und für jede andere rekursiv aufzählbare Relation $R \subseteq \mathbb{N}^m$ $(m \geq 1)$, $R \leq_m S$ gilt.

S ist eine „schwerste" rekursiv aufzählbare Relation.

6.70 Satz Existenz vollständiger, r.a. Relationen.

K_0 ist vollständig für die rekursiv aufzählbaren Relationen.

6.71 Lemma

Ist S vollständig für rekursive aufzählbare Relationen und gilt $S \leq_m R$ für R rekursiv aufzählbar, dann ist auch R vollständig.

Beweis:

Sei T rekursiv aufzählbar, dann ist $T \leq_m S \leq_m R$. D. h. $T \leq_m R$.

6.72 Folgerung K ist auch vollständig.

Anwendung des s-m-n-Theorems. Zu zeigen:
$K_0 = \{(a,x) \in \mathbb{N}^2 : \varphi_a(x) \downarrow\} \leq_m K = \{a : \varphi_a(a) \downarrow\}$
Betrachte $Rzax$ gdw K_0ax. Dann ist R rekursiv aufzählbar, d. h. $R = dom(\varphi_b^{(3)})$. Das s-m-n-Theorem liefert:

$$\varphi_b^{(3)}(z,a,x) \downarrow \text{ gdw } \varphi_{s_{2,1}(b,a,x)}(z) \downarrow$$

Vollständige rekursiv aufzählbare Relationen Die Sätze von Rice

Dann gilt mit $z = s_{2,1}(b,a,x)$

$$\begin{aligned} \varphi_a(x) \downarrow \quad & \text{gdw } \varphi_b^{(3)}(s_{2,1}(b,a,x),a,x) \downarrow \\ & \text{gdw } \varphi_{s_{2,1}(b,a,x)}(s_{2,1}(b,a,x)) \downarrow \\ & \text{gdw } Ks_{2,1}(b,a,x) \end{aligned}$$

Da $h(a,x) = s_{2,1}(b,a,x)$ primitiv rekursiv ist, gilt die Behauptung $K_0 \leq_m K$.

Methoden für den Nachweis von Unentscheidbarkeit und nicht rekursive Aufzählbarkeit.

6.73 Satz (Rice) Entscheidbare Indexmengen

Sei $S \subset \mathcal{R}_p^{(1)}(\mathbb{N})$ Menge einstelliger Funktionen. Dann ist die **Indexmenge** der Funktionen in S, die Menge $S_\mu = \{a \in \mathbb{N} : \varphi_a \in S\}$, genau dann entscheidbar, wenn $S = \varnothing$ oder $S = \mathcal{R}_p^{(1)}(\mathbb{N})$.
Also ist eine Indexmenge $R \subseteq \mathbb{N}$ entscheidbar gdw $R = \varnothing$ oder $\mathbb{N}$.

Beweis:

Ist $S = \varnothing$ oder $S = \mathcal{R}_p^{(1)}(\mathbb{N})$, so ist $S_\mu = \varnothing$ oder $S_\mu = \mathbb{N}$ und somit entscheidbar.
Sei $S \neq \varnothing, \neq \mathcal{R}_p^{(1)}(\mathbb{N})$. Angenommen S_μ ist entscheidbar.
O.b.d.A. $\uparrow \notin S$ (sonst statt S_μ wähle $\neg S_\mu$).

Die Sätze von RICE (Fort.)

Nach Vor. gibt es eine Funktion $f \in S$.
Definiere

$$F(x,y) = \begin{cases} f(y) & \text{falls } Kx \\ \uparrow & \text{sonst} \end{cases}$$

F ist berechenbar, also gibt es ein $a \in \mathbb{N}$ mit $F(x,y) = \varphi_a^{(2)}(y,x) = \varphi_{s_{1,1}(a,x)}^{(1)}(y)$. Die Funktion $g(x) = s_{1,1}(a,x)$ ist primitiv rekursiv und es gilt

$x \in K \Rightarrow \varphi_{g(x)} = f \in S$ gdw $g(x) \in S_\mu$
$x \notin K \Rightarrow \varphi_{g(x)} = \uparrow \notin S$ gdw $g(x) \notin S_\mu$
Es gilt somit $x \in K$ gdw $g(x) \in S_\mu$, d. h. $K \leq_m S_\mu$ ↯

Nichttriviale Indexmengen sind also nicht rekursiv entscheidbar. Sind sie überhaupt rekursiv aufzählbar?

6.74 Definition

$f : \mathbb{N} \to \mathbb{N}$ heißt **endliche Restriktion** von $g : \mathbb{N} \to \mathbb{N}$, falls $dom(f)$ endlich ist und $f \sqsubseteq g$, d. h.

$$dom(f) \text{ endlich und } f(x) \downarrow \Rightarrow (g(x) \downarrow \wedge f(x) = g(x))$$

Die Sätze von Rice (Fort.)

6.75 Satz (Rice-Shapiro) Rekursiv aufzählbare Indexmengen

Sei $S \subset \mathcal{R}_p^{(1)}(\mathbb{N})$. Ist die Menge $S_\mu = \{a \in \mathbb{N} : \varphi_a \in S\}$ rekursiv aufzählbar, dann folgt für $f \in \mathcal{R}_p^{(1)}(\mathbb{N})$:
- $f \in S$ gdw es gibt eine endliche Restriktion g von f in S.

Insbesondere sind $spez(f)$ (f total berechenbar) und somit auch $\ddot{A}quiv$ nicht rekursiv aufzählbar.

Es gibt keine effektive Aufzählung aller while-Programme, die nur primitiv rekursive oder nur totale μ-rekursive-Funktionen berechnen.

Beweis:

Sei S_μ rekursiv aufzählbar und $f \in \mathcal{R}_p^{(1)}(\mathbb{N})$.
• Sei g endliche Restriktion von f und $g \in S$.
Angenommen $f \notin S$. Betrachte die Funktion:

$$F(x,y) = \begin{cases} f(y) & \text{falls } Kx \\ g(y) & \text{sonst} \end{cases}$$

Behauptung: F ist berechenbar. Da

$$F(x,y) = \begin{cases} g(y) & \text{falls } y \in dom(g) \\ sgn(succ(\varphi_x(x))) \cdot f(y) & \text{sonst} \end{cases}$$

Beweisfortsetzung

Sei $a \in \mathbb{N}$ mit $F(x,y) = \varphi_a^{(2)}(y,x) = \varphi_{s_{1,1}(a,x)}(y)$

$x \in K \Rightarrow F(x,\cdot) = f$

$x \notin K \Rightarrow F(x,\cdot) = g$

d. h. $x \notin K$ gdw $s_{1,1}(a,x) \in S_\mu$, d.h. S_μ ist nicht r.a. $\lightning$

• Sei umgekehrt $f \in S$. Angenommen es gebe keine endliche Restriktion von f in S.

Betrachte die Funktion:

$$F(x,y) = \begin{cases} f(y) & \text{falls } \neg\Phi_x(x) \leq y \\ \uparrow & \text{sonst} \end{cases}$$

F ist berechenbar.
Sei $a \in \mathbb{N}$ mit $F(x,y) = \varphi_a^{(2)}(y,x) = \varphi_{s_{1,1}(a,x)}(y)$.

Es gilt:

• $x \in K \Rightarrow F(x,\cdot)$ ist eine endlichen Restriktion von f:

Da $\varphi_x(x)\downarrow$, d. h. $\Phi_x(x) = t_0$ und für

$y < t_0 \quad F(x,y) = f(y)$

$y \geq t_0 \quad F(x,y) \uparrow$

• $x \notin K \Rightarrow F(x,\cdot) = f$, da $\neg\Phi_x(x) \leq y$ gültig.

$x \notin K$ gdw $s_{1,1}(a,x) \in S_\mu$, d.h. S_μ ist nicht r.a. $\lightning$

Nicht rekursiv aufzählbare Mengen

6.76 Folgerung Der Satz erlaubt uns zu zeigen, dass gewisse Indexmengen nicht r.a. sind.
Ist nämlich A eine Indexmenge partiell rekursiver Funktionen und $p \in A$ für die gilt:
a) Es gibt ein $q \in \neg A$ mit $\varphi_p \sqsubseteq \varphi_q$ oder
b) Es gibt kein Index einer endliche Restriktion von φ_p in A.

Insbesondere sind folgende Mengen A_i nicht rekursiv aufzählbar:

$$\begin{aligned}
A_0 &= \{x \in \mathbb{N} \mid \varphi_x = \uparrow, \text{d. h. } dom(\varphi_x) = \varnothing\} \\
A_1 &= \{x \in \mathbb{N} \mid dom(\varphi_x) \text{ endlich}\} \\
A_2 &= \{x \in \mathbb{N} \mid im(\varphi_x) \text{ endlich}\} \\
A_3 &= \{x \in \mathbb{N} \mid dom(\varphi_x) = \mathbb{N}, \text{d. h. } \varphi_x \text{ total}\} \\
A_4 &= \{x \in \mathbb{N} \mid im(\varphi_x) = \mathbb{N}, \text{d. h. } \varphi_x \text{ surjektiv}\} \\
\neg A_5 &= \{x \in \mathbb{N} \mid a \in dom(\varphi_x) \ a \text{ fest, d. h. } \varphi_x(a)\downarrow\} \\
\neg A_6 &= \{x \in \mathbb{N} \mid a \in im(\varphi_x) \ a \text{ fest, d. h. } \exists y \varphi_x(y) = a\}
\end{aligned}$$

Wende Satz 6.75 an.

Beachte $\neg A_0, \neg A_5, \neg A_6$ sind rekursiv aufzählbar.

Existenz von Programmen (berechenbare Funktionen) mit bestimmten Eigenschaften

6.77 Satz Fixpunktsatz - Rekursionssatz

a) **FPS.** Zu jedem $f \in \mathcal{R}^{(1)}(\mathbb{N})$ (totale μ-rekursive Funktion) gibt es ein $p \in \mathbb{N}$ mit $\varphi_{f(p)} = \varphi_p$.
D.h. die „Programme" p und $f(p)$ berechnen die gleiche Funktion.
Es muss **nicht** $f(p) = p$ gelten.

b) **RS.** Zu jeder Funktion $G \in \mathcal{R}^{(2)}(\mathbb{N})$, $G : \mathbb{N}^2 \to \mathbb{N}$, gibt es ein $q \in \mathbb{N}$ mit $\varphi_q = G(\cdot, q)$.
D. h. $\varphi_q(x) = G(x,q)$ für alle $x \in \mathbb{N}$.

Beweis:

a) Betrachte die Funktion $F(x,y) = \varphi_{f(s_{1,1}(y,y))}(x)$

$F \in \mathcal{R}^{(2)}(\mathbb{N})$. Sei q Index von F, d. h.

$$F(x,y) = \varphi_q^{(2)}(x,y) \underset{s-m-n}{=} \varphi_{s_{1,1}(q,y)}(x)$$

Setze $p = s_{1,1}(q,q)$. Dann gilt
$\varphi_{f(p)}(x) = \varphi_{f(s_{1,1}(q,q))}(x) = F(x,q) = \varphi_{s_{1,1}(q,q)}(x)$
$= \varphi_p(x)$.
Beachte: wählt man für $f(y) = s_{1,1}(y,y)$, so gibt es ein p mit $\varphi_{s_{1,1}(p,p)} = \varphi_p$.

Anwendungen

b) Sei $G \in \mathcal{R}_p^{(2)}(\mathbb{N})$ und a Index von G, d. h.
$G(x,y) = \varphi_a^{(2)}(x,y) \underset{s-m-n}{=} \varphi_{s_{1,1}(a,y)}(x)$.
Sei $f(y) = s_{1,1}(a,y)$. Nach a) gibt es $q \in \mathbb{N}$ mit

$$\varphi_q(x) = \varphi_{f(q)}(x) = \varphi_{s_{1,1}(a,q)}(x) = G(x,q)$$

6.78 Folgerung Existenz spezieller Programme!

• Es gibt ein while-Programm, das für jede Eingabe seine eigene Codezahl ausgibt.
Sei $G(x,y) = f^{(2)}_{proj(2)}(x,y) = y$. Dann liefert RS $q \in \mathbb{N}$ mit

$$\varphi_q(x) = G(x,q) = q$$

• Es gibt eine primitiv rekursive Funktion f mit $f : \mathbb{N} \to \mathbb{N}$, so dass $\forall x \in \mathbb{N}.\ dom(\varphi_{f(x)}) = \{x^2\}$.
„Implizite Definition von Programmen"
Sei $h(x,y) = \mu z.(\chi_{\{x^2\}}(y) = 1) = \begin{cases} 0 & y = x^2 \\ \uparrow & \text{sonst} \end{cases}$
$h \in \mathcal{R}_p$, d. h. $h(x,y) = \varphi_a^{(2)}(y,x) = \varphi_{s_{1,1}(a,x)}(y)$.
Wähle $f(x) = s_{1,1}(a,x)$.

• FPS liefert sogar Existenz von $p_0 \in \mathbb{N}$ mit
$\varphi_{p_0} = \varphi_{f(p_0)}$, d. h. $dom(\varphi_{p_0}) = dom(\varphi_{f(p_0)}) = \{p_0^2\}$

6.5 Die Churchsche These

Maschinennahe Programme: Register- und Turing-Maschinen.

Bisher: $\mathcal{R}_p(\mathbb{N})$=While-programmierbar über N.
=While-rekursiv programmierbar über N.

Wichtige Ergebnisse

Existenz **universeller** berechenbarer-Funktionen:
$\Phi^{(n)} : \mathbb{N}^{n+1} \to \mathbb{N}$ **Zeitkomplexitätsfunktion.**

- $\Phi^{(n)}(p, \vec{x}) = \mu t.\mathsf{first}(i^t(\mathsf{inp}^{(n)}(p, \vec{x}))) = 0$
  $\mathsf{inp}^{(n)}, i, \mathsf{first} \in \mathcal{P}(\mathbb{N})$

$\varphi^{(n)} : \mathbb{N}^{n+1} \to \mathbb{N}$ **universelle Funktion** für n-stellige Funktionen.

- $\varphi^{(n)}(p, \vec{x}) = \mathsf{out}(i^{\Phi^{(n)}(p,\vec{x})}(\mathsf{inp}^{(n)}(p, \vec{x})))$
  Kleenesche Normalform für μ-rekursive Funktionen.
- $\Phi^{(n)}, \varphi^{(n)} \in \mathcal{R}_p(\mathbb{N})$ (d. h. berechenbar)
- $\forall f \in \mathcal{R}_p^{(n)}(\mathbb{N}) \quad \exists p \in \mathbb{N} : f = \varphi_p^{(n)}$

Jede berechenbare Funktion hat einen „Index“.

Sätze: „Programmtransformationen“

s-m-n Theorem: $\exists s_{m,n} \in \mathcal{P}^{m+1}\varphi_p^{n+m}(\vec{x}, \vec{y}) = \varphi^n_{s_{m,n}(p,\vec{y})}(\vec{x})$

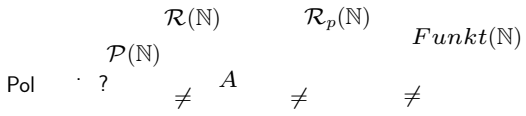
Fixpunktsatz: $\forall f \in \mathcal{R}^{(1)}(\mathbb{N}) \;\; \exists p \in \mathbb{N} \;\; \varphi_{f(p)} = \varphi_p$

Rekursionstheorem: $\forall G \in \mathcal{R}_p^{(2)}(\mathbb{N}) \;\; \exists p \in \mathbb{N} \;\; \varphi_p = G(\cdot, p)$

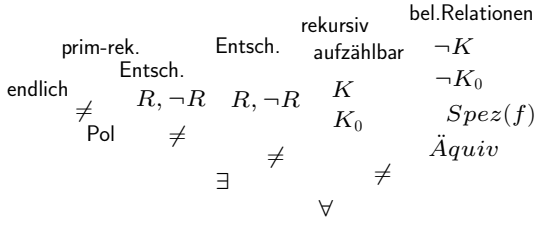
Die Sätze von Rice: Entscheidbarkeit und r.a. von Indexmengen.

Klassifizierung von Funktionen und Relationen

$f : \mathbb{N}^n \to \mathbb{N}$



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



Churchsche These

Die Klasse der effektiv berechenbaren Funktionen ist genau die Klasse der μ-rekursiven Funktionen. Jede Formalisierung von berechenbaren Funktionen liefert die gleiche Klasse.

Wir werden einige dieser Formalisierungen kurz vorstellen.

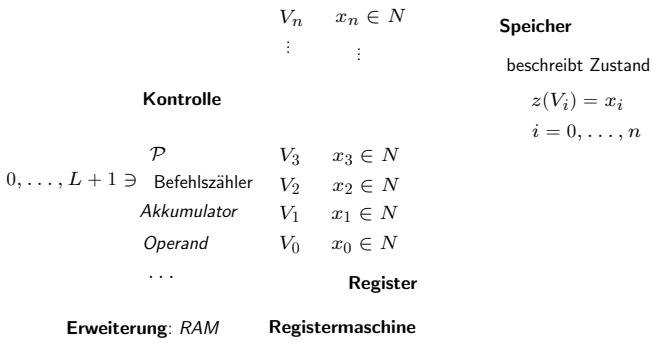
6.79 Definition Register-Maschinen (goto-Programme über N)
Goto-Programme über der Variablenmenge $V = \{V_0, \ldots, V_n\}$ sind markierte Befehlsfolgen der Form

$$\begin{aligned} \mathcal{P} :: \quad & 0 : B_0 \\ & 1 : B_1 \\ & \vdots \\ & L : B_L \end{aligned}$$

Mit **Befehlen** $B_i, i \in \{0, \ldots, L\}$ einer der Formen
$\bullet V_i := s(V_i)$ $\bullet V_i := p(V_i)$ $\bullet$**if** $V_i = 0$ **then goto** l_1 **else goto** l_2
mit $V_i \in V, l_1, l_2 \in \{0, \ldots, L+1\}$ (**Marken**).

Die intendierte Semantik von $\boldsymbol{s}$**,** $\boldsymbol{p}$ ist die Nachfolger- bzw. die Vorgängerfunktion auf $\mathbb{N}$.

Register-Maschinen Semantik



Interpretersemantik:
$I_{\mathcal{P}}(l, z) : \{0, \ldots, L+1\} \times \mathcal{Z} \to \{0, \ldots, L+1\} \times \mathcal{Z}$
Startzustand: $(0, z)$, Eingaben $z(V_i) = x_i \in \mathbb{N}$

$I_{\mathcal{P}}(l, z) =$

$$\begin{cases} (l+1, z(V_i/z(V_i)+1)) & l : V_i := s(V_i) \in \mathcal{P} \\ (l+1, z(V_i/z(V_i) \dot{-} 1)) & l : V_i := p(V_i) \in \mathcal{P} \\ (l_1, z) & l : \underline{\textbf{if}}\ V_i = 0\ \underline{\textbf{then}}\ \textbf{goto}\ l_1\ \underline{\textbf{else}}\ \textbf{goto}\ l_2 \in \mathcal{P} \\ & \qquad \wedge z(V_i) = 0 \\ (l_2, z) & l : \underline{\textbf{if}}\ V_i = 0\ \underline{\textbf{then}}\ \textbf{goto}\ l_1\ \underline{\textbf{else}}\ \textbf{goto}\ l_2 \in \mathcal{P} \\ & \qquad \wedge z(V_i) \neq 0 \\ (l, z) & l = L+1 \text{ oder } l \text{ kein Label in } \mathcal{P}\ \ (\textbf{Stopp}) \end{cases}$$

Register-Maschinen berechenbare Funktionen

Programm $\mathcal{P}$ stoppt aus Startzustand z gdw keine Befehlsausführung mehr möglich.

Ein- Ausgabevereinbarungen für die Berechnung von Funktionen $f : \mathbb{N}^l \to \mathbb{N}$: $\mathcal{P}$ **berechnet** f gdw

i) Die Rechnung stoppt aus Anfangszustand $z(V_i) = x_i, i = 1, \ldots, l, z(V_i) = 0$ sonst gdw $(x_1, \ldots, x_l) \in dom(f)$.

ii) Gilt $(x_1, \ldots, x_l) \in dom(f)$, $y = f(x_1, \ldots, x_l)$, so stoppt $\mathcal{P}$ in einem Zustand z' mit $z'(V_0) = y$. Also gilt:

$$\exists t \in \mathbb{N} : I^t_{\mathcal{P}}(0, z) = (L + 1, z')$$

6.80 Beispiel Einfache RM bzw. goto-Programme

Sei S festes Register mit Inhalt 0, d. h. $z(V_s) = 0$

a) Register „leeren“

$V \Leftarrow 0 ::$ $0 : V := p(V)$
$1 :$ **if** $V = 0$ **then goto** 2 **else goto** 0

b) $Z \Leftarrow Z + 1$, $Z \Leftarrow Z \dot{-} 1$ sind leicht anzugeben.

Einfache RM bzw. goto-Programme

c) „Inhalt umspeichern“

$Z \Leftarrow Y ::$ — copy Y nach Z — Hilfsregister U — initialisiert mit 0 — unbedingter Sprung: **goto** 1 (Abkürzung)

$0 : Z \Leftarrow 0$
$1 :$ **if** $Y = 0$ **then goto** 5 **else goto** 2
$2 : Y := p(Y)$
$3 : U := s(U)$
$4 :$ **if** $V_s = 0$ **then goto** 1 **else goto** 1
$5 :$ **if** $U = 0$ **then goto** 10 **else goto** 6
$6 : U := p(U)$
$7 : Z := s(Z)$
$8 : Y := s(Y)$
$9 :$ **goto** 5

6.81 Lemma

- Jede μ-rekursive Funktion ist goto-berechenbar.
- Jede goto-berechenbare Funktion ist μ-rekursiv.

Beweisidee:

- Zeige die Grundfunktionen sind goto-berechenbar.

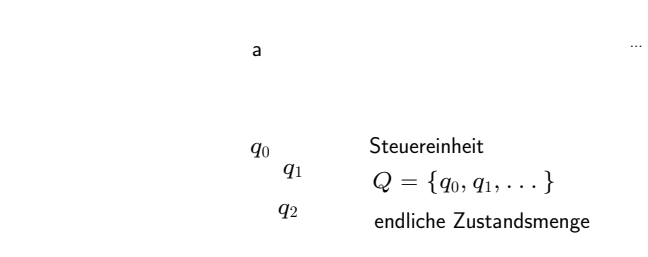
$f = go(h_1, \ldots, h_m), \quad f = R(g, h), \quad f = \mu.g$

Lassen sich durch Goto-Programme berechnen, falls $g, h_1, \ldots, h_m, h$ goto-berechenbar.

- Zeige die Funktion $I_{\mathcal{P}}$ lässt sich durch eine primitiv rekursive Funktion simulieren. Dann Iteration und Minimierung.

Turingmaschinen (nach A. Turing)

$\Sigma = \{b_1, \ldots, b_r\}$ Alphabet, Leersymbol $\square \notin \Sigma$, $a \in \Sigma$



Zu jedem Zeitpunkt sind nur endlich viele Felder nicht mit $\square$ belegt. Es gibt somit stets zusammenhängenden Block endlicher Länge, der das A-Feld enthält und außerhalb davon nur Leerzeichen vorkommen.

Erlaubte Operationen:

In Abhängigkeit vom Zustand und Inhalt des A-Felds schreibe Zeichen ins A-Feld, bewege Lese-Schreibkopf um ein Feld nach links (L), rechts (R) oder bleibe darauf (S), ändere Zustand.

Beschreibung durch „**Übergangsfunktion**“
$\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R, S\}$, wobei Q endliche Zustandsmenge und Γ Bandalphabet sind.

Turingmaschinen (Forts.)

6.82 Definition

Eine Turingmaschine T ist ein 6 -Tupel $T = (Q, \Sigma, \Gamma, \delta, q_0, F)$ mit folgenden Bestandteilen:

- Q ist endliche **Zustandsmenge**.
- Σ Eingabealphabet mit $\square \notin \Sigma$. **Eingabezeichen.**
- Γ Bandalphabet mit $\Sigma \subseteq \Gamma$ und $\square \in \Gamma$. **Bandzeichen**.
- q_0 ist der **Startzustand**.
- $F \subseteq Q$ Menge der **Endzustände**.
- $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R, S\}$ (oft als total verlangt) genügt $dom(\delta) = (Q \backslash F) \times \Gamma$. **Übergangsfunktion**. Wird oft als Tafel oder Tabelle angegeben.

Ein **Bandzustand** von T ist ein Tripel (q, x, β) mit $q \in Q$ (aktueller Zustand), $x \in \mathbb{Z}$ (aktuelle Kopfposition), $\beta : \mathbb{Z} \to \Gamma$ totale Funktion (aktueller Bandinhalt) mit $\beta(y) = \square$ für alle bis auf endlich viele $y \in \mathbb{Z}$.

Turingmaschinen (Forts.)

T überführt den Bandzustand (q, x, β) in den Bandzustand (q', x', β') (**Folgezustand**), falls

- $\delta(q, \beta(x)) = (q', \beta'(x), M)$
- $\beta'(y) = \beta(y)$ für alle $y \neq x$ Folgezustand
- $x' = \begin{cases} x-1 & \text{falls } M = L \\ x+1 & \text{falls } M = R \\ x & \text{falls } M = S \end{cases}$

Eine **Rechnung** von T ist eine endliche Folge von Bandzuständen $(z_0, \ldots, z_n)$, so dass T für alle $0 \leq i < n$ den Zustand z_i in z_{i+1} überführt.

Eine Rechnung heißt haltend, falls $z_n = (q, x, \beta) \wedge q \in F$.

6.83 Beispiel

$\Sigma = \{1, 2\}$, $\Gamma = \Sigma \cup \{\square\}$, $Q = \{q_0, q_1, q_2\}$, $F = \{q_2\}$

δ	$\square$	1	2
q_0	$(q_0, \square, R)$	$(q_1, \square, R)$	$(q_1, \square, R)$
q_1	$(q_2, \square, S)$	$(q_1, \square, R)$	$(q_1, \square, R)$
q_2	$(q_2, \square, S)$	$(q_2, 1, S)$	$(q_2, 2, S)$

Andere Beschreibungen von δ möglich: z.B.

Fünftupel $\{q\ b\ q'\ b'\ M : q \in Q, b \in \Gamma\}$

Beispiele von Turingmaschinen

Beispiel Rechnung:

-2 -1 0 1 2 3 4 5 6 7

1 2 2

q_0

q_2

q_1

Anfangszustand $z_0 = (q_0, 0, \beta)$ wobei $\beta(2) = 1$, $\beta(3) = 2$, $\beta(5) = 2$, sonst $\square$

$z_1 = (q_0, 1, \beta)$
$z_2 = (q_0, 2, \beta)$
$z_3 = (q_1, 3, \beta_1)$ mit $\beta_1(3) = 2 = \beta_3(5)$ sonst $\square$
$z_4 = (q_1, 4, \beta_2)$ mit $\beta_3(5) = 0$ sonst $\square$
$z_5 = (q_2, 4, \beta_2)$ $\ni q_2$ haltend oder „Haltezustand“
$z_6 = (q_2, 4, \beta_2)$ „Endzustand“

Wirkung: TM sucht rechts vom A-Feld $w \in \Sigma^*$ als Block und löscht es. Bleibt auf Leerzeichen hinter w stehen, falls $w \in \Sigma^+$ existiert. Stoppt nicht, falls auf AFeld und rechts davon lauter $\square$-Zeichen sind.

Turing-berechenbare Funktionen

Unäre Codierung von Zahlen $n \rightarrow \underbrace{|||\ldots|}_{n}$

6.84 Definition

Eine Funktion $f : \mathbb{N}^n \rightarrow \mathbb{N}$ heißt **Turing-berechenbar**, falls es eine TM T mit Eingabealphabet $\{|, \$\}$ gibt, so dass der Bandzustand $(q_0, 0, \beta)$ mit

- $\beta(i) = \square$ für $i < 0$ und $i > x_1 + x_2 + \cdots + x_n + n$
- $\beta(0) = \beta(x_1 + 1) = \beta(x_1 + x_2 + 2) = \cdots = \beta(x_1 + \cdots + x_n + n) = \$$
- $\beta(i) = |$ für alle anderen i

genau dann zu einer haltenden Rechnung ergänzt werden kann, wenn $f(x_1, \ldots, x_n) \downarrow$ und, ist in diesem Fall (q, i, β') der Zustand, in dem die Rechnung hält, dann ist die Anzahl der Striche $|$, die in $\beta'(i+1), \beta'(i+2), \ldots$ unmittelbar aufeinanderfolgen, gleich $f(x_1, \ldots, x_n)$.

Turing-berechenbare Funktionen (Forts.)

.... $ x_1 viele Striche $ .... $ x_n viele Striche $ ....

q_0

T

.... $f(x_1, \ldots, x_n)$ viele Striche kein Strich ....

$q \in F$

Beispiele

6.85 Beispiel

1. Vorgänger und Nachfolger: $v(x) = n - 1, \ s(n) = n + 1$

v :: \$ n-Striche \$ $\rightsquigarrow$ \$ $n \dot{-} 1$-Striche \$

q_0

$q \in F$

$\delta(q_0, \$) = (q_1, \square, R) \quad \Sigma = \{|, \$\}$
$\delta(q_1, |) = (q_3, \$, S) \quad \Gamma = \{|, \$, \square\}$
$\delta(q_1, \$) = (q_2, \$, L) \quad Q = \{q_0, q_1, q_2, q_3\}$
$\delta(q_2, \square) = (q_3, \$, S) \quad F = \{q_3\}$

s :: \$ n-Striche \$ $\rightsquigarrow$ \$ $n + 1$-Striche \$

q_0

$q \in F$

$\delta(q_0, \$) = (q_0, |, L) \quad \Sigma = \{|, \$\}$
$\delta(q_0, \square) = (q_1, \$, S) \quad \Gamma = \{|, \$, \square\}$
$Q = \{q_0, q_1\}$
$F = \{q_1\}$

Beispiele (2)

2. Suche rechts vom A-Feld erstes Vorkommen von \$, bleibe dort stehen.

nicht \$ \$ $\rightsquigarrow$ nicht \$ \$

q_0

$q \in F$

$\delta(q_0, b) = (q_1, b, R) \quad b \in \Gamma$
$\delta(q_1, b) = (q_1, b, R) \quad b \in \Gamma \backslash \$$
$\delta(q_1, \$) = (q_2, \$, S)$

SL\$. Analog SR\$

3. Verschiebe Block \$$n$-Striche\$ um ein Feld nach links

$0 \ 1 \ 2 \ \cdots n+2 \ \cdots$ 0 $n+1$
\$ n Striche \$ Zeichen $\rightsquigarrow$ \$ n Striche \$ Zeichen

q_0

$q \in F$

Beispiele (2) (Forts.)

$\Sigma = \{|, \$\}$
$\Gamma = \Sigma \cup \{\square\}$
$Q = \{q_0, \ldots, q_5\}$
$F = \{q_5\}$

$q_0 \ b \ \$ \ R \ q_1 \quad b \in \Gamma$
$q_1 \ b \ b \ R \ q_2$
$q_2 \ | \ | \ L \ q_3$
$q_2 \ \$ \ \square \ L \ q_4$
$q_3 \ b \ | \ R \ q_1$
$q_4 \ b \ \$ \ R \ q_5$
$q_5 \ b \ b \ S \ q_5$

VL. Analog VR (verschiebe nach rechts).
$\delta(q_2, \$) = (q_4, \square, L)$
q_3 merkt sich |
q_4 merkt sich \$

Strategie:

q_0 0
q_1
q_3 q_2
q_4 q_1

Simulation von RM-Programme durch TM

6.86 Lemma

• Jede RM (goto)-berechenbare Funktion lässt sich durch eine TM berechnen. Also ist jede μ-rekursiv Funktion TM-berechenbar.

• Jede Turing-berechenbare Funktion ist μ-rekursiv.

Beweisidee:

Simuliere Berechnung des goto-Programms über $V_0, \ldots, V_m$, $f : \mathbb{N} \to \mathbb{N}$. Speichere Zustand $z : V \to \mathbb{N}$ als

\$ \$ x_1-Striche \$ x_2-Striche \$ $\cdots$ \$ x_n-Striche \$ \$ $\cdots$ \$

$m + 2$ \$-Zeichen

$z(V_2) = x_2$

q_0

Ein-Befehl wird durch mehrere TM-Schritte simuliert.
Die Zustände entsprechen Marken im Goto-Programm.
$V_i := s(V_i)$

- Verschiebe die Blöcke vor V_i jeweils um ein Feld nach links wie oben.
- Wende s TM an.
- SL\$ i-mal.

Simulation von TM durch μ rekursive Funktionen

Simulation der Überführungsfunktion einer Turing-Maschine durch eine primitiv-rekursive Funktion $i_T : \mathbb{N} \to \mathbb{N}$, die auf geeignet codierten Bandzuständen arbeitet (q, x, β).

Dann wie üblich.

Wir haben somit weitere Charakterisierungen der μ-rekursiven Funktionen, die die Churchsche These untermauern.

Man kann für $\mathcal{P}(\mathbb{N})$ (primitiv-rekursiven Funktionen) ebenfalls eine Charakterisierung mit Hilfe einfacher Programmiersprachen finden.

z. B. For-Programme über $\mathbb{N}$

Anweisungsfolgen:

Anweisung: Zuweisung, Test oder For-Schleife der Form:

$$\underline{\textbf{for}}\ I = 0\ \underline{\textbf{to}}\ J\ \underline{\textbf{do}}\ \alpha\ \underline{\textbf{end}};$$

$I, J \in V$ α For-Programm über V, das keine Zuweisung der Form $I := t$ oder $J := t$ mit Term t enthält (Schleife wird genau $z(J)$ mal ausgeführt, dabei wird stets α ausgeführt und I um 1 erhöht).

6.6 Berechenbarkeit auf Zeichenreihen Wortfunktionen

Wortfunktionen: $f : (\Sigma^*)^n \to \Sigma^*$

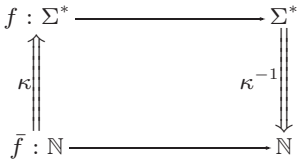
Wortrelationen $R \subseteq (\Sigma^*)^n$ **Sprachen** $R \subseteq \Sigma^*$

Bisher: Funktionen, Relationen auf $\mathbb{N}$: μ-rekursive Funktionen.

Turing-Maschinen und While-Programme sind für beliebige Alphabete bzw. beliebige Strukturen definiert.

Verallgemeinerung der Ergebnisse der Rekursionstheorie, insbesondere über Entscheidbarkeit und Nichtentscheidbarkeit auf Wortfunktionen und Relationen.

1. Möglichkeit: Codierung von Σ^* in $\mathbb{N}$. Einfache effektive Codierungen: z. B. Folgencodierungsfunktion oder Interpretation als Zahl (binäre-, dezimale-Darstellung).



$\bar{f}(n) = \kappa^{-1}(f(\kappa(n)))$

Berechenbarkeit auf Zeichenreihen (2)

Definition $f \in \mathcal{R}_p(\Sigma)$ gdw $\bar{f} \in \mathcal{R}_p(\mathbb{N})$.

Zeige: Unabhängig von der gewählten effektiven Codierung.

2. Möglichkeit: $\Sigma = \{a_1, \ldots, a_n\}$ $(n \geq 1)$

While-Programme:

Betrachte die Algebra

$String = (\Sigma^*, \varepsilon, succ_{a_1}, \ldots, succ_{a_n}, pred)$ mit

- $succ_a(u) = au \quad (a \in \Sigma)$
- $pred(au) = u \quad pred(\varepsilon) = \varepsilon$

Ordnungen auf Σ^*: $\leq_{llex}$ Länge-Lexikographisch,

d. h. $u \leq_{llex} v$ gdw $|u| < |v|$ oder
$|u| = |v| \wedge u \leq_{lex} v$

Wobei $\leq_{lex}$ lex. Ordnung, die von lin. Ordnung auf Σ induziert wird (z. B. $a_1 < a_2 < a_3 \cdots < a_n$).

Beachte: $|\cdot| : \Sigma^* \to \Sigma^* \quad |u| = a_1^{|u|}$ und $\chi_{\leq llex}$ sind while-berechenbar.

3. Möglichkeit: **μ-rekursive Funktionen über Σ^***: $a \in \Sigma$

- $f^{(n)}_{NULL}(\vec{w}) = \varepsilon$, $f^{(n)}_{SUCC_a}(\vec{w}) = aw_1$ $(\vec{w} = (w_1, \ldots, w_n))$
- $f^n_{PROJ(i)}$ wie bisher.

Berechenbarkeit auf Zeichenreihen (3)

Komposition: $f \circ (g_1, \ldots, g_n)$ wie bisher.

Primitive Rekursion: $f = R_\Sigma(g, h_1, \ldots, h_n)$, falls

- $f(\vec{u}, \varepsilon) = g(\vec{u}, \varepsilon)$
- $f(\vec{u}, a_i v) = h_i(\vec{u}, f(\vec{u}, v), v)$

Minimierung: $f(\vec{u}) = \mu_{llex} v \cdot g(\vec{u}, v) = \varepsilon$

$f(\vec{u}) = w$ $llex$-minimal mit $g(\vec{u}, w) = \varepsilon$, sofern ein solches existiert.

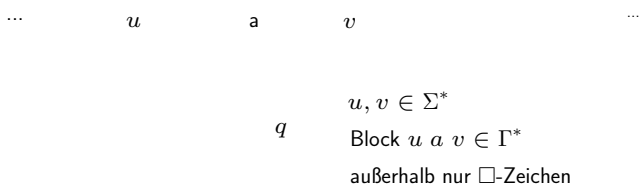
4. Möglichkeit: **RM (Goto-Programme)**: $z(V_i) \in \Sigma^*$

Befehle:

- $X := s(a, X)$ Wirkung wie $succ_a$ in Σ^*
- $X := p(X)$ Wirkung wie pred in Σ^*
- **<u>if</u>** $X = \varepsilon$ **<u>then</u> goto** l_1 **<u>else</u> goto** l_2
- Test (Anfangsbuchstabe ist $a \in \Sigma$):
  <u>if</u> $AB(X) = a$ **<u>then</u> goto** l_1 **<u>else</u> goto** l_2

Berechenbarkeit auf Zeichenreihen (4)

5. Möglichkeit:**Turing-Maschinen**:
$T = (Q, \Sigma, \Gamma \supseteq \Sigma \cup \{\Box\}, \delta, q_0, F \subseteq \Gamma)$



Konfiguration: $u\ q\ a\ v \in \Gamma^* \cdot Q \cdot \Gamma^+ \quad (\Gamma^+ = \Gamma^* \backslash \{\varepsilon\})$
Zwei Konfigurationen heißen **äquivalent**, falls sie sich nur durch Blöcke von $\Box$-Zeichen davor und danach unterscheiden.

Anfangskonfigurationen: $q_0 \Box w \qquad w \in \Sigma^*$

Berechenbarkeit auf Zeichenreihen (5)

Folgekonfigurationen
k' ist Folgekonfiguration von k : $k \underset{T}{\vdash} k'$, falls gilt

k	$\delta(q,a)$	k'	
$u\ q\ a\ v$	(q', a', S)	$u\ q'\ a'\ v$	
$u\ q\ a\ v$	(q', a', R)	$u\ a'\ q'\ v$	$v \in \Gamma^+$
$u\ q\ a$	(q', a', R)	$u\ a'\ q'\ \Box$	
$u\ b\ q\ a\ v$	(q', a', L)	$u\ q'\ b\ a'\ v$	$b \in \Gamma$
$q\ a\ v$	(q', a', L)	$q'\ \Box\ a'\ v$	

Eine **Rechnung** einer TM T ist Folge von Konfigurationen $(k_0, \ldots, k_n)$ mit $k_i \underset{T}{\vdash} k_{i+1}$. Sie ist haltend, falls k_n eine Endkonfiguration ist, d. h. $k_n = u\ q\ v$ mit $q \in F$. Schreibe $k_0 \overset{*}{\vdash}_T k_n$
Eine **Berechnung** einer TM T ist eine Rechnung wobei k_0 eine Anfangskonfiguration $(k_0 = q_0 \Box w) w \in \Sigma^*$ ist.
TM-berechenbare Funktionen: $f : (\Sigma^*)^n \to \Sigma^*$ ist TM-berechenbar, falls es eine TM T gibt die f berechnet, d. h.

a) T stoppt für Anfangskonfiguration $k_0 = q_0 \Box x_1 \Box x_2 \Box \cdots \Box x_n \Box$ gdw $(x_1, \ldots, x_n) \in dom(f)$
b) Gilt $(x_1, \ldots, x_n) \in dom(f)$ und $y = f(x_1, \ldots, x_n)$, so hat T beim Stopp die Konfiguration $\Box^i\ q \Box x_1 \Box \cdots \Box x_n \Box y \Box^j$, für geeignete $i, j \in \mathbb{N}$.

Berechenbarkeit auf Zeichenreihen (6)

6.87 Satz
$f : (\Sigma^*)^n \to \Sigma^*$, dann sind äquivalent

- $f \in \mathcal{R}_p(\Sigma)$, d. h. f ist μ-rekursiv.
- f ist while-programmierbar über String.
- f ist RM-(goto)-berechenbar.
- f ist TM-berechenbar.

Existenz universeller Funktionen, universeller Programme und universeller Maschinen wie bisher.

- Relationen: Entscheidbarkeit, rek-Aufzählbarkeit $R \subseteq (\Sigma^*)^n$
- R entscheidbar gdw $\chi_R \in \mathcal{R}_p(\Sigma)$, $\chi_R(\vec{w}) = \begin{cases} \varepsilon & w \notin R \\ a_1 & w \in R \end{cases}$
- R rekursiv-aufzählbar gdw $R = dom(f)$, $f \in \mathcal{R}_p(\Sigma)$.
- Halteproblem: $K_0 = \{(T, w) \mid T$ mit Anfangskonfiguration $q_0 \Box w$ hält, d. h. Berechnung mit Endkonfiguration$\}$

Ist nicht entscheidbar.

Bisherige Ergebnisse lassen sich übertragen.

Insbesondere: Reduzierbarkeit $\leq_m$.

Turing-Maschinen als Akzeptoren von Sprachen und als entscheidende Automaten

6.88 Definition Akzeptierende und erkennende TM
Sei $T = (Q, \Sigma, \Gamma \supseteq \Sigma \cup \{\Box\}, \delta, q_0, F)$

- T **akzeptiert** die Sprache $L \subseteq \Sigma^*$ gdw für $w \in \Sigma^*$: $q_0 \Box w \overset{*}{\vdash}_T u\ q\ v$ mit $q \in F$ gdw $w \in L$, d. h. es gibt haltende Berechnung aus $q_0 \Box w$ gdw $w \in L$, $L = L(T)$.
- T **entscheidet** die Sprache $L \subseteq \Sigma^*$ gdw für jede Eingabe $w \in \Sigma^*$ hält T: $q_0 \Box w \overset{*}{\vdash}_T u\ q\ v$ mit $q \in F$ und $w \in L$, so $q = q_y \qquad w \notin L$, so $q = q_n$ wobei $q_y, q_n \in F$ spezielle „Ja“-, „Nein“- Zustände sind.

6.89 Lemma

- $L \subseteq \Sigma^*$ ist entscheidbar gdw es gibt eine TM T, die L entscheidet.
- $L \subseteq \Sigma^*$ ist rekursiv aufzählbar gdw es gibt eine TM T, die L akzeptiert, d. h. $L = L(T)$.

Beachte: Andere Konventionen sind möglich. Andere TM: Mehrband TM, δ unvollständig, Band einseitig unendlich, mehrspurig, nicht deterministisch.

Beispiele

Turing-Programme

- Turing Befehl hat die Form
  $B \equiv Op$ $\quad Op \in \Gamma \dot\cup \{R, L, \text{stopp}\}$
  $B \equiv q$ $\quad q \in Q$ unbedingter Sprung
  $B \equiv a, q$ $\quad a \in \Gamma, q \in Q$ bedingter Sprung nach q, falls a in A-Feld
- Turing Programm ist endliche Folge markierter Befehle
  $Q = \{q_0, q_1, \ldots, q_n\}$, $q_i \neq q_j$ für $i \neq j$
  TP:: $q_0 : B_0$ $\quad B_i$ Turing-Befehl
  $q_1 : B_1$
  $\vdots$
  $q_n : B_n$
- Semantik eines T-Programms durch Angabe der TM
  $T = (Q', \Sigma, \Gamma, \delta, q_0, F)$, $a \in \Gamma$, $Q' = Q \cup \{q_{n+1}\}$

$$\begin{array}{lll} \delta(q_i, a) & = (q_{i+1}, a', S) & B_i \equiv a' \in \Gamma \\ & = (q_{i+1}, a, M) & B_i \equiv M \in \{L, R\} \\ & = (q_{i+1}, a, S) & B_i \equiv a', q \;\; a \neq a' \\ & = (q, a, S) & B_i \equiv a, q \\ & = (q_{n+1}, a, S) & B_i \equiv \text{ stopp oder } i = n+1 \end{array}$$

  $F = \{q_{n+1}\}$
- Eigenschaft:Jede TM kann durch ein äquivalentes T-Programm beschrieben werden.

Beispiele

Suche Links von AF das erste Vorkommen von □ ... Rechts ...

$SL\,\square:$	L	$SR\,\square:$	R
	$\square, Fin$		$\square, Fin$
	$SL\,\square$		$SR\,\square$
$Fin:$	Stopp	$Fin:$	Stopp

TM, die die Menge der Palindrome über $\{a, b\}^*$ entscheidet
$L = \{w \in \{a, b\}^* : w = w^{mi}\}$

q_0:	R	q_b:	$\square$
	$\square : q_y$		$SR\,\square$
	$a : q_a$		L
	$b : q_b$		$\square : q_y$
			$b : q$
			$a : q_n$
q_a:	$\square$	q:	$\square$
	$SR\,\square$		$SL\,\square$
	L		q_0
	$\square : q_y$		
	$a : q$		
	$b : q_n$		

Diese Turing Programm hält für jede Eingabe $w \in \Sigma^*$ und entscheidet die Menge der Palindrome.

Simulation von TM-Berechnungen durch Wortersetzungssystemen (Σ, Π)

Π ist Menge von Produktionen $l ::= r$, mit $l \in \Delta^+$, $r \in \Delta^*$

Kalkül: $\frac{u\ l\ v}{u\ r\ v}$ für $l ::= r \in \Pi$, $u, v \in \Delta^*$.

Sei

$T = (Q, \Sigma, \Gamma, \delta, q_0, F)$ und $\Delta = Q \dot\cup \Gamma \dot\cup \{\#\}$

Produktionen Π_T:

Für jedes $\delta(q, a) = (q', a', S)$: $\quad q\ a ::= q'\ a' \in \Pi_T$.

Für jedes $\delta(q, a) = (q', a', R)$ und $b \in \Gamma$:

$$q\ a\ b ::= a'\ q'\ b \in \Pi_T$$
$$q\ a\ \# ::= a'\ q'\ \square\ \# \in \Pi_T$$

Für jedes $\delta(q, a) = (q', a', L)$ und $b \in \Gamma$

$$b\ q\ a ::= q'\ b\ a' \in \Pi_T$$
$$\#q\ a ::= \#q'\ \square\ a' \in \Pi_T$$

Offenbar gilt:

$k = u\ q\ v \underset{T}{\vdash} u'\ q'\ v' = k'$, d. h. k' ist Folgekonfiguration von k

gdw $\quad \#u\ q\ v\# \overset{1}{\underset{\Pi_T}{\vdash}} \#\ u'\ q'\ v'\ \#$,

d. h. Rechnungen der TM T können in Π_T simuliert werden.

$\#q_0\ \square\ w\ \# \underset{\Pi_T}{\vdash} \#\ u\ q\ v\ \#$ $\quad$ gdw $\quad q_0\ \square\ w \overset{*}{\underset{T}{\vdash}} u\ q\ v$

für $w \in \Sigma^*$, $u, v \in \Gamma^*$, $q \in Q$.

Das Ableitbarkeitsproblem

6.90 Definition Sei (Σ, Π) ein Wortersetzungssystem.

Das **Ableitbarkeitsproblem** $Abl \subset \Sigma^* \times \Sigma^*$ für (Σ, Π) ist gegeben durch

$$Abl\ x\ y \qquad \text{gdw} \qquad x \underset{\Pi}{\vdash} y$$

(für $x, y \in \Sigma^*$) d.h. „y lässt sich aus x mit Hilfe der Produktionen aus Π ableiten“.

6.91 Satz Unentscheidbarkeit des Ableitbarkeitsproblems

Das Ableitbarkeitsproblem für beliebige Wortersetzungssysteme ist nicht entscheidbar.

Beweis:

Reduziere das Halteproblem für TM auf das Ableitbarkeitsproblem. Die Konstruktion TM $T \longrightarrow$ simulierendes Wortersetzungssystem Π_T ist effektiv. Für $q \in F$ füge noch die Produktionen
$a\ q ::= q$, $q\ a ::= q$, für $a \in \Delta \backslash \{\#\}$ und $\#\ q\ \# ::= q$ hinzu.

Dann gilt: T hält mit Eingabe w

gdw $\exists u, v \in \Gamma^*, q \in F$ mit $q_0\ \square\ w \overset{*}{\underset{T}{\vdash}} u\ q\ v$

gdw $\exists u, v \in \Gamma^*, q \in F$ mit $\#\ q_0\ \square\ w\ \# \underset{\Pi_T}{\vdash} \#\ u\ q\ v\ \#$

gdw $\exists q \in F$ mit $\#\ q_0\ \square\ w\ \# \underset{\Pi_T}{\vdash} q$.

Also ist das Halteproblem auf das Ableitbarkeitsproblem reduzierbar.

Speziellere Ergebnisse (z.B. spezielles Wort ableitbar) sind möglich.

Das Postsche Korrespondenzproblem (PCP)

6.92 Definition

Das Postsche Korrespondenzproblem (**PCP**) besteht aus allen Listen von Wortpaaren

$$\mathcal{L} = (x_1 \sim y_1, \ldots, x_k \sim y_k) \qquad k \geq 1$$

mit nichtleeren Wörtern $x_i, y_i \in \Sigma^*(1 \leq i \leq k)$ zu denen es eine Indexfolge $i_1, \ldots, i_n \in \{1, \ldots, k\}$ mit $n \geq 1$ gibt, so dass

$(*) \qquad x_{i_1} \cdots x_{i_n} = y_{i_1} \ldots y_{i_n}$ gilt.

Schreibe: PCP($\mathcal{L}$). Die Folge $(i_1, \ldots, i_n)$ ist Lösung, falls $(*)$ gilt.

Einschränkungen: z. B. $i_1 = 1$ spezielle PCP (**SPCP**).

Beachte Parameter: $\Sigma, k, x_i, y_i \in \Sigma^+, 1 \leq i \leq k$

Lösung: Liste natürlicher Zahlen aus $\{1, \ldots, k\}$.

Beachte: Zu gegebener Liste $i_1, \ldots, i_n$ ist es einfach zu überprüfen, ob sie eine Lösung ist.

Beispiel

6.93 Beispiel $\Sigma = \{0, 1\}$

- $\mathcal{L}_1 = (0 \sim 0\,0\,0,\ 0\,1\,0\,0 \sim 0\,1,\ 0\,0\,1 \sim 1) \quad k = 3$
  Lösungen können nur mit $i_1 = 1$ oder $i_1 = 2$ beginnen.
  1. Lösung: $i_1 = 1, i_2 = 3 : \underline{0}\ \underline{0\,0\,1} = \underline{0\,0\,0}\ \underline{1}$.
  2. Lösung: $i_1 = 2$, $i_2 = 1$, $i_3 = 1$, $i_4 = 3$.
  $x_2x_1x_1x_3$ $\underline{0\,1}\,0\,0\,\underline{0}\,0\,0\,0\,1$
  $y_2y_1y_1y_3$ $\underline{0\,1}\,\underline{0\,0\,0}\,0\,0\,0\,1$
  d. h. PCP($\mathcal{L}_1$), SPCP($\mathcal{L}_1$)
- $\mathcal{L}_2 = (\underline{1} \sim \underline{1}\,1\,1, \underline{1\,0}\,1\,1\,1 \sim \underline{1\,0},\ 1\,0 \sim 0) \quad k = 3$
  Lösung: $2, 1, 1, 3$ (muss mit 1 oder 2 beginnen).
  $x_2x_1x_1x_3$ $\underline{1\,0}\,1\,1\,1\,\underline{1}\,\underline{1}\,\underline{1\,0}\quad \underline{1}\,\underline{1}\cdots$ keine
  $y_2y_1y_1y_3$ $\underline{1\,0}\,\underline{1\,1\,1}\,\underline{1\,1\,1}\,\underline{0}\quad \underline{1\,1\,1}\,\underline{1\,1\,1}\cdots$ Lösung
  d. h. PCP($\mathcal{L}_2$), $\neg$ SPCP($\mathcal{L}_2$)
- $\mathcal{L}_3 = (\underline{0\,1} \sim \underline{0\,1}\,0, 1\,\underline{0\,0} \sim \underline{0\,0}, 0\,1\,0 \sim 1\,0\,0)$
  Behauptung: $\neg$ PCP($\mathcal{L}_3$):: Lösung muss mit 1 beginnen und mit 2 enden. $t \in \{1, 2, 3\}^+$ $t = 1t'2$.
  $\underline{0\,1}\,\underline{0\,1\,0}\,\underline{0\,1\,0}\cdots$ t' muss mit 3 beginnen
  $\underline{0\,1\,0}\,\underline{1\,0\,0}\,\underline{1\,0\,0}\cdots$
  Keine Fortsetzung möglich, da kein y mit 1 endet $\cdots\underline{1\,0\,0}$ $\cdots\underline{0\,0}$

Beispiel (Forts.)

- $\mathcal{L}_4 = (0\,0\,1 \sim 0, 0\,1 \sim 0\,1\,1, 0\,1 \sim 1\,0\,1, 1\,0 \sim 0\,0\,1)$
  Es gilt PCP($\mathcal{L}_4$) aber $\neg$ SPCP($\mathcal{L}_4$).
  Lösung etwas länger: mit 2 beginnen, mit 3 enden, 1 muss verwendet werden.

$\underline{0\,1}\,\underline{1\,0}\,0\,1\,0\,1\,1\,0$ keine Forts.
$\underline{0\,1\,1}\,\underline{0\,0\,1}\,0\,1\,1\,0\,1\,1\,0\,0\,1$
$\underline{0\,1}\,\underline{1\,0}\,\underline{0\,1}\,1\,0\,1\,0\,0\,1\,0\,0\,1\,0\,1\,1\,0\,0\,1\,1\,0\cdots$
$\underline{0\,1\,1}\,\underline{0\,0\,1}\,\underline{1\,0\,1}\,0\,0\,1\,0\,0\,1\,0\,1\,1\,0\,\underline{0}\,\underline{1}\,\underline{1}\,0\,0\,1\,1\,0\,1\,0\,0\,1$

Unentscheidbarkeit von PCP

6.94 Satz Das PCP ist unentscheidbar.

Beweisidee: Reduziere Ableitbarkeitsproblem für WES auf PCP.

- Sei $\gamma = (\Sigma, \Pi)$ ein Wortersetzungssystem ohne ε-Regeln.
  $Abl_\gamma(u, v)$ gdw $u \underset{\Pi}{\vdash} v$
- Konstruktion: Aus $\gamma, u, v \rightsquigarrow \mathcal{L}_{u,v}$ mit $u \underset{\Pi}{\vdash} v$ gdw PCP($\mathcal{L}_{u,v}$).
  Sei $\gamma = (\Sigma, \Pi)$, $\Pi = \{u_i ::= v_i \mid i = 1, \ldots, n\}$,
  $\Sigma = \{a_1, \ldots, a_r\}$. $\hat{\Sigma} = \{\hat{a}_1, \ldots, \hat{a}_r\}$ Kopie von Σ und $\Gamma = \Sigma \dot\cup \hat{\Sigma} \dot\cup \{+, \hat{+}, [,]\}$.
- $u, v \in \Sigma^*$. Definiere $\mathcal{L}_{u,v}$ über Γ durch

$$\begin{array}{lll}
X_i :: [u\,+, &], & +, \hat{+}, a_1, \ldots, a_r, \hat{a}_1, \ldots, \hat{a}_r, v_1, \ldots, v_n, \hat{v}_1, \ldots, \hat{v}_n \\
\quad \wr & \wr \quad \wr\ \wr\ \wr & \wr\ \wr \quad \wr\ \wr \quad \wr\ \wr \quad \wr \\
Y_i :: [& , \hat{+}v], & \hat{+}, +, \hat{a}_1, \ldots, \hat{a}_r, a_1, \ldots, a_r, \hat{u}_1, \ldots, \hat{u}_n, u_1, \ldots, u_n \\
\text{NR}: 1 & 2 & 3\ 4\ 5\ \ldots\ l\ \ m \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad n
\end{array}$$

$l = 4 + r$, $m = 4 + r + 1$,$n = 4 + 2r + 2n$

Behauptung: PCP($\mathcal{L}_{u,v}$) gdw $u \underset{\Pi}{\vdash} v$ (o.b.d.A. $u \neq v$).

„$\Leftarrow$“ $u \underset{\Pi}{\vdash} v$ etwa $u = w_0 \overset{(1)}{\underset{\Pi}{\vdash}} w_1 \overset{(1)}{\underset{\Pi}{\vdash}} \cdots \overset{(1)}{\underset{\Pi}{\vdash}} w_k = v \quad k > 0$

Es gibt $t_j \in J^*$, $j = 1, \ldots, k$ mit

j-gerade $\quad X(t_j) = [w_0 + \hat{w}_1\hat{+}w_2 + \cdots + \hat{w}_{j-1}\hat{+}w_j$
$\qquad\qquad Y(t_j) = [w_0 + \hat{w}_1\hat{+}w_2 + \cdots + \hat{w}_{j-1}$

j-ungerade $\quad X(t_j) = [w_0 + \hat{w}_1\hat{+}w_2 + \cdots \hat{+}w_{j-1} + \hat{w}_j$
$\qquad\qquad Y(t_j) = [w_0 + \hat{w}_1\hat{+}w_2 + \cdots \hat{+}w_{j-1}$