Rekursionstheorie

GeTEXt von
Johannes Renner

13. Oktober 2003

Mitschrift zur Vorlesung von Prof.Dr.W.Thomas an der RWTH-Aachen im SS2003:

- Vertiefung von B.u.K.
- Tieferes Verständnis von „Algorithmus“
- Attraktive Theorie
- Historische Quelle der Informatik

Fehler und Vorschläge bitte an johannes.renner@rwth-aachen.de. Auch falls jemand Lust hat zu helfen das Skript zu vervollständigen (handschriftliche Unterlagen vorhanden).

Inhaltsverzeichnis

1 Nummerierung von Programmen, Funktionen, Mengen

1.1 Grundlage 1

Drei Verwendungen von Algorithmen:

- als Berechnungsverfahren für (partielle) Funktionen
- als Entscheidungsverfahren für Mengen
- als Aufzählungsverfahren für Mengen

Objekte, die manipuliert werden: endlich (z.B. Wörter, Zahlen).
Hier: Bereich $\mathbb{N}$ der natürlichen Zahlen

- Ein Algorithmus $\mathcal{A}$ *berechnet* die partielle Funktion $f : \mathbb{N}^n \to \mathbb{N} \Longleftrightarrow$ bei Eingabe $\bar{x} \in \mathbb{N}^n$ terminiert A, gdw. $\bar{x} \in Def(f)$ und $\mathcal{A}$ liefert $f(\bar{x})$ als Wert.
- $\mathcal{A}$ *entscheidet* $A \subseteq \mathbb{N}^n \Longleftrightarrow$ bei Eingabe $\bar{x} \in \mathbb{N}^n$ terminiert $\mathcal{A}$, und zwar mit Ausgabe 1(Ja), falls $\bar{x} \in A$ und 0(Nein), sonst.
- $\mathcal{A}$ *zählt* $A \subseteq \mathbb{N}^n$ *auf* $\Longleftrightarrow$ $\mathcal{A}$ liefert (gestartet ohne Eingabe) nach und nach in irgendeiner Reihenfolge als Ausgaben genau die Elemente von A.

Bei Existenz solcher Algorithmen sagen wir:

- f „berechenbar“
- A „entscheidbar“
- A „aufzählbar“

1.Bemerkung(Charakterisierung von „entscheidbar“ durch „aufzählbar“):
$A \subseteq \mathbb{N}^n$ entscheidbar $\Longleftrightarrow$ A aufzählbar und $\mathbb{N} \backslash A$ aufzählbar.
Beweis: Folie.

2.Bemerkung(Charakterisierung von „aufzählbar“ durch „berechenbar“):
$A \subseteq \mathbb{N}^n$ aufzählbar $\Longleftrightarrow A = Def(f)$ für eine berechenbare Funktion $f : \mathbb{N}^n \to \mathbb{N}$.

1.2 Grundlage 2

Der Begriff Algorithmus (zur Berechnung von Fkt.) ist mathematisch präzisierbar durch folgende Formalismen:

- λ-Kalkül (Church, 1936)
- Rekursive Funktionen (Gödel, Herbrand, Kleene, 1930-1936)
- Turingmaschinen (Turing, 1936)
- Post-Systeme (Post, 1936)
- Registermaschinen (Shepherdson, Sturgis, 1963)

$f : \mathbb{N}^n \to \mathbb{N}$ heisst *rekursiv*, wenn f durch einen dieser fünf Formalismen definierbar/berechenbar ist.
Analog: „rekursiv aufzählbar“ für „aufzählbar“, „rekursiv“ für „entscheidbar“.

Im folgenden: Registermaschinen (unbeschränkte Registermaschinen, URM)

- Variablen $X_1, X_2, \ldots$ mit Werten in $\mathbb{N}$
- Ein URM-Programm ist eine Liste $P = (I_1, \ldots, I_k)$ von Instruktionen mit
  $I_k = k \quad stop$ und für $1 \leq j \leq k$:

  $I_j = j \quad Incr(X_i)$, oder
  $I_j = j \quad Decr(X_i)$, oder
  $I_j = j \quad if\ X_i > 0\ goto\ l\ [else\ goto\ j+1], 1 \leq l \leq k.$
- Konfiguration K ist Folge (Progr. Zeilennummer, Wert v. X_1, Wert v. X_2,...):
  $K = (k_0, k_1, k_2, \ldots)$, mit $k_i =$ Wert von $X_i, i > 0$ und $1 \leq k_0 \leq k$.
- Folgekonfiguration K' von K ist wie üblich definiert.
- $P(x_1, \ldots, x_n) \downarrow \Longleftrightarrow$ beginnend mit Konfiguration $(1, x_1, \ldots, x_n, 0, \ldots)$ terminiert P, d.h. erreicht Konfiguration $(k, y_1, y_2, \ldots)$, wobei y_1 das Ergebnis ist.
  In diesem Fall schreibe: $P(x_1, \ldots, x_n) \downarrow y_1$, sonst $P(x_1, \ldots, x_n) \uparrow$.
- $f : \mathbb{N}^n \to \mathbb{N}$ ist URM-berechenbar $\Longleftrightarrow$ es existiert ein URM-Programm P, das f berechnet. D.h. für $\bar{x} \in Def(f)$ gilt $P(x_1, \ldots, x_n) \downarrow f(\bar{x})$, sonst $P(x_1, \ldots, x_n) \uparrow$.

1.3 Grundlage 3

Programme (TM,...) sind endliche Objekte und damit nummerierbar.

1. **Paarfunktion**:
   $[\quad] : \mathbb{N}^2 \to \mathbb{N}$ bijektiv, mit

   $$[x, y] = 2^x \cdot (2y + 1) - 1$$

   Beachte: Jede Zahl ≥ 1 ist in Primfaktorenzerlegung schreibbar als
   $2^x\cdot$<Produkt ungerader Primzahlen>.
   $[z]_1$ = Ergebnis von 2 in PFZ von $z + 1$,
   $[z]_2$ analog.

2. **Folgenkodierung**:
   Sei p_i die i-te Primzahl ($p_0 = 2, p_1 = 3, \ldots$)
   Bsp.: Kodiere $(3, 0, 2)$ durch $p_0^4 \cdot p_1^1 \cdot p_2^3 = 2^4 \cdot 3 \cdot 5^3$
   Allgemeine Definition:
   $<>: \mathbb{N}^* \to \mathbb{N}$ durch

   $$\begin{array}{rcl} < \epsilon > & = & 1 \\ < x_0, \ldots, x_n > & = & p_0^{x_0+1} \cdot \ldots \cdot p_n^{x_n+1} \end{array}$$

   Bem.: $<>$ ist eine Bijektion.

Definition(URM-Programm-Codierung):

$$\begin{array}{rcl} code(Incr(X_i)) & = & [0, i] \\ code(Decr(X_i)) & = & [i, 0] \\ code(if\ X_i > 0\ goto\ l) & = & [i, l] \\ code(stop) & = & 0 \end{array}$$

Für $P = (I_1, \ldots, I_k)$ setze $code(P) =< code(I_1), \ldots, code(I_k) >$.

Beispiel:

P_∞: 1 $Incr(X_1)$
2 $if\ X_1 > 0\ goto\ 1$
3 $stop$

$\to code(P_\infty) =< [0, 1], [1, 1], 0 >=< 2, 5, 0 >= 2^3 \cdot 3^6 \cdot 5^1$

Nicht jede Zahl kodiert ein Programm.

Definition:
Für $e \in \mathbb{N}$, setze

$$P_e = \begin{cases} P, & \text{falls } code(P) = e \\ P_\infty, & \text{sonst} \end{cases}$$

Für $e \in \mathbb{N}, n \geq 1$ definiere $\varphi_e^{(n)} : \mathbb{N}^n \to \mathbb{N}$ durch

$$\varphi_e^{(n)}(x_1, \ldots, x_n) = \begin{cases} y, & \text{falls } P_e(x_1, \ldots, x_n) \downarrow y \\ \perp, & \text{falls } P_e(x_1, \ldots, x_n) \uparrow \end{cases}$$

Für $n = 1$ schreibe φ_e.

Bemerkung:
Jede URM-berechenbare Funktion f hat die Form

$$f = f_e^{(n)}, (e \text{ Index von } f)$$

Daraus:
Nummerierung der aufzählbaren Mengen $\subseteq \mathbb{N}^n$:

$$W_e^{(n)} := Def(\varphi_e^{(n)})$$

Dann:
Jede aufzählbare Menge $M \subseteq \mathbb{N}^n$ hat die Form

$$M = W_e^{(n)}, (n = 1 : W_e)$$

Interessante aufzählbare Menge:

$$K = \{e \in \mathbb{N} | e \in W_e\} = \{e | P_e(e) \downarrow\}$$

2 Primitiv rekursive und μ-rekursive Funktionen

Grassmann, Dedekind 1888

- $x + 0 = x, \quad x \cdot 0 = 0$
- $x + S(y) = S(x + y)$
- $x \cdot S(y) = (x \cdot y) + x$

S Nachfolgerfunktion: $\quad 0! = 1, \quad S(y)! = y! \cdot S(y)$

2.1 Schema der primitiven Rekursion

Aus $g : \mathbb{N}^n \to \mathbb{N}$ und $h : \mathbb{N}^{n+2} \to \mathbb{N}$ entsteht $f : \mathbb{N}^{n+1} \to \mathbb{N}$ durch *primitive Rekursion* (prim.Rek.), wenn

$$\left.\begin{array}{rcl} f(\bar{x}, 0) & = & g(\bar{x}) \\ f(\bar{x}, S(y)) & = & h(\bar{x}, y, f(\bar{x}, y)) \end{array}\right\} (*)$$

Bemerkung:

1. $(*)$ definiert f eindeutig.
2. Anwendung des Schemas auf partielle Funktionen möglich.
   $f(\bar{x}, y) = \perp$, falls einer der Werte $f(\bar{x}, 0), \ldots, f(\bar{x}, y-1)$ oder $h(\bar{x}, y, f(\bar{x}, y-1))$ undefiniert ist.

Beispiele:

1. '+' entsteht aus $g = id_{\mathbb{N}}$ und folgendem h durch prim.Rek.: $h(x, y, z) = S(z)$.
2. '$\cdot$' entsteht aus der 1-stelligen Konstante 0 und folgendem h durch prim.Rek.: $h(x, y, z) = z + x$.

2.2 Schema der Einsetzung

Aus $h : \mathbb{N}^m \to \mathbb{N}$ und $g_1, \ldots, g_m : \mathbb{N}^n \to \mathbb{N}$ entsteht $f : \mathbb{N}^n \to \mathbb{N}$ durch *Einsetzung*, falls $f(\bar{x}) = h(g_1(\bar{x}), \ldots, g_m(\bar{x}))$.

Bemerkung: Anwendung auf partielle Funktionen möglich: $f(\bar{x})$ undefiniert, falls ein $g_i(\bar{x})$ undefiniert oder alle $g_i(\bar{x})$ definiert und $h(g_1(\bar{x}), \ldots, g_m(\bar{x}))$ undefiniert.

Basisfunktionen sind:

- Nachfolgerfunktion S,
- für $n \geq 0$ jeweils die n-stellige Konstante, $0 : c_0^{(n)}$,
- für $n \geq 1, 1 \leq i \leq n$, die i-te n-stellige Projektion $p_i^{(n)}(\bar{x}) = x_i$.

Definition: Eine Funktion heisst *primitiv rekursiv* (prim.rek.), falls sie aus Basisfunktionen durch endlichmalige Anwendung von Einsetzung und primitiver Rekursion entsteht.

Beispiele:

1. '+' ist prim.rek.
   '+' entsteht aus $g = p_1^{(1)} (= id_{\mathbb{N}})$ und $h(x, y, z) = S(z)$ durch prim.Rek.
   genauer: $h(x, y, z) = S(p_3^{(3)}(x, y, z))$, h entsteht aus S und $p_3^{(3)}$ durch Einsetzung, ist also prim.rek.
2. ' $\cdot$ ' entsteht aus $c_0^{(1)}$ und $h(x, y, z) = z + x$ durch prim.Rek.
   $h(x, y, z) = +(p_3^{(3)}(x, y, z), p_1^{(3)}(x, y, z))$, also ist h prim.rek.

Argument-Lemma:
Sei $f(x_1, \ldots, x_n) = g(x_{i_1}, \ldots, x_{i_m})(\{i_1, \ldots, i_m\} \subseteq \{1, \ldots, n\})$.
Wenn g prim.rek., dann ist auch f prim.rek.
Begründung: $f(\bar{x}) = g(p_{i_1}^{(n)}(\bar{x}), \ldots, p_{i_m}^{(n)}(\bar{x}))$, Einsetzung mit prim.rek. Funktionen.

Beispiele:

1. **Potenzfunktion**:

$$\begin{aligned} pot(x, y) &= x^y \\ pot(x, 0) &= c_1^{(1)} \\ pot(x, S(y)) &= \underbrace{pot(x, y) \cdot x}_{\text{prim.rek. analog ' $\cdot$ '}} \qquad [x^{4+1} = x^4 \cdot x] \end{aligned}$$

   Zu $c_1^{(1)}(x) = S(c_0^{(1)}(x))$ analog: Jede Konstante $c_k^{(n)}$ ist prim.rek.

2. **Vorgängerfunktion**:
   $V : \mathbb{N} \to \mathbb{N}$, mit
   $$V(x) = \begin{cases} x-1, & \text{falls } x > 0 \\ 0, & \text{sonst} \end{cases}$$
   Notation: $x \dot{-} 1$
   $$\begin{array}{rcl} V(0) & = & c_0^{(0)} \\ V(S(y)) & = & h(y, V(y)) \end{array}$$
   wähle $h = p_1^{(2)}$
3. $x \dot{-} y$ über
4. $|x-y| = (x \dot{-} y) + (y \dot{-} x)$
5. **Signum-Funktion**:
   $$sg(x) = \begin{cases} 0, & \text{falls } x = 0 \\ 1, & \text{falls } x > 0 \end{cases}$$
   prim.rek.:
   $$\begin{array}{rcl} sg(0) & = & c_0^{(0)} \\ sg(S(y)) & = & c_1^{(2)}(y, sg(y)) \end{array}$$
   $$\overline{sg}(x) = \begin{cases} 0, & \text{falls } x > 0 \\ 1, & \text{falls } x = 0 \end{cases}$$
   analog mit $c_1^{(0)}, c_0^{(2)}$.

2.3 Beschränkte Summen, Produkte

Zu $g : \mathbb{N}^{n+2} \to \mathbb{N}$ bilde
$$sum_g(\bar{x}, y) := \sum_{z=0}^{y} g(\bar{x}, y, z)$$
Ist g prim.rek., so auch sum_g.
Analog für Π.

Beweis:
Betrachte $F(\bar{x}, y, t) := \sum_{z=0}^{t} g(\bar{x}, y, z)$:
$$\begin{array}{rcl} F(\bar{x}, y, 0) & = & g(\bar{x}, y, 0) = G(\bar{x}, y) \\ F(\bar{x}, y, S(t)) & = & \underbrace{F(\bar{x}, y, t) + g(\bar{x}, y, S(t))}_{H(\bar{x}, y, F(\bar{x}, y, t))} \text{ prim.rek.} \\ sum_g(\bar{x}, y) & = & F(\bar{x}, y, y) \end{array}$$

2.4 Primitiv rekursive Relationen

Zu $R \subseteq \mathbb{N}^n$ sei $\chi_R : \mathbb{N}^n \to \mathbb{N}$ die charakteristische Funktion von R:

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

R heisst prim.rek., falls χ_R prim.rek.

Beispiele:

- $x \neq y$ prim.rek.: $\chi_{\neq}(x, y) = sg(|x - y|)$
  ($<$ ähnlich)
- $x = y$ prim.rek.: $\chi_{=}(x, y) = \overline{sg}(|x - y|)$
- Sind $R_1, R_2 \subseteq \mathbb{N}^n$ prim.rek., so auch $R_1 \cup R_2$, $R_1 \cap R_2$, $\mathbb{N}^n \backslash R_1$:

  $$\chi_{R_1 \cup R_2}(\bar{x}) = sg(\chi_{R_1}(\bar{x}) + \chi_{R_2}(\bar{x}))$$

  ($\cap$, $\backslash$ ähnlich)

2.5 Beschränkte Quantifizierung

Zu $R \subseteq \mathbb{N}^{n+2}$ bilde $Q_E \subseteq \mathbb{N}^{n+1}$ und $Q_A \subseteq \mathbb{N}^{n+1}$:

$$Q_E\ \bar{x}, y \Longleftrightarrow \exists\, z \leq y, \text{ so dass } R\ \bar{x}yz$$

Ist R prim.rek., so auch Q_E, Q_A.

Begründung: $\chi_{Q_E}(\bar{x}, y) = sg(\sum_{z=0}^{y} \chi_R(\bar{x}, y, z))$. Für Q_A analog mit Π.

Beispiele:

- $x|y$ ist prim.rek.: $x|y \Longleftrightarrow \exists z\ x \cdot z = y$.
- $P\,x$(x ist Primzahl) ist prim.rek.:
  $P\,x \Longleftrightarrow x \neq 0 \wedge x \neq 1 \wedge \forall z \leq x\ (z = 1 \vee z = x \vee \neg z|x)$

2.6 Beschränkte Minimierung

[R prim.rek. $\to$ f prim.rek.]
Zu $R \subseteq \mathbb{N}^{n+1}$ definiere $f : \mathbb{N}^{n+1} \to \mathbb{N}$ durch

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

kurz $f(\bar{x}, y) = \mu z \leq y \; R\bar{x}yz$.

Behauptung: R prim.rek. $\Longrightarrow f$ prim.rek.

Beweis:
$F(\bar{x}, y, t) = \mu z \leq t \; R\bar{x}yz$
F prim.rek., dann auch f, fertig.

$$\begin{aligned} F(\bar{x}, y, 0) &= 0 = c_0^{(n+1)}(\bar{x}, y) \\ F(\bar{x}, y, S(t)) &= \begin{cases} F(\bar{x}, y, t), & \text{falls } \exists z \leq t \; R\bar{x}yz \\ S(t), & \text{sonst.} \end{cases} \\ &= F(\bar{x}, y, t) \cdot \chi_Q(\bar{x}, y, t) + S(t) \cdot \chi_{\bar{Q}}(\bar{x}, y, t) \\ &=: h(\bar{x}, y, t, F(\bar{x}, y, t)) \text{ prim.rek.} \end{aligned}$$

2.7 Bemerkungen zu prim.rek. Funktionen

1. f prim.rek. $\Longleftrightarrow f$ ist *LOOP-berechenbar*, d.h. durch ein sog. LOOP-Programm berechenbar.
   LOOP-Programme haben

   - Wertzuweisungen: $X_i := X_i + 1, \; X_i := X_i - 1$
   - Tests: $X_i > 0$?

   $$\begin{aligned} < LP > ::= \; & Wertzuweisung \,|\, < LP >; < LP > \,|\, if \; Test \; then \; < LP > else < LP > \,| \\ & loop \; X_i \; begin < LP > end \end{aligned}$$

   Semantik von „*loop* X_i *begin P end*“:
   führe P so oft durch wie anfangs Wert von X_i
   Beispiel: $X_1 := X_1 + X_2 \equiv$ „ *loop* X_2 *begin* $X_1 := X_1 + 1$ *end*“
   Bem.: WHILE-Programme entstehen durch Hinzunahme von

   $$While \; Test \; do \; P \; end$$

2. Es gibt viele Varianten von Rekursionsschemata, die auf prim.Rek. zurückführbar sind:

   a) **Wertverlaufsrekursion**:

   $$\begin{aligned} f(\bar{x}, 0) &= g(\bar{x}) \\ f(\bar{x}, S(y)) &= h(\bar{x}, y, < f(\bar{x}, 0) >, \ldots, < f(\bar{x}, y) >) \end{aligned}$$

b) **Simultane Rekursion**:

$$\begin{array}{rcl} f_1(\bar{x},0) & = & g_1(\bar{x}) \\ f_2(\bar{x},0) & = & g_2(\bar{x}) \\ f_1(\bar{x},S(y)) & = & h_1(\bar{x},y,f_1(\bar{x},y),f_2(\bar{x},y)) \\ f_2(\bar{x},S(y)) & = & h_2(\bar{x},y,f_1(\bar{x},y),f_2(\bar{x},y)) \end{array}$$

g_1, g_2, h_1, h_2 prim.rek. $\Longrightarrow f_1, f_2$ prim.rek.

c) **Rekursion mit Parameterwechsel**:

$$\begin{array}{rcl} f(x,0) & = & g(x) \\ f(x,S(y)) & = & h(x,y,f(w(x),y)) \end{array}$$

g, h, w prim.rek. $\Longrightarrow f$ prim.rek.

3. **Ackermann-Funktion** $A : \mathbb{N}^2 \to \mathbb{N}$, eine durch Rekursion definierte totale, nicht prim.rek. Funktion.

$$\begin{array}{rcl} A(0,y) & = & S(y) \\ A(S(x),0) & = & A(x,1) \\ A(S(x),S(y)) & = & A(x,A(S(x),y)) \end{array}$$

Auswertungsbeispiel:

$$\begin{array}{rcl} A(1,3) & = & A(0,A(1,2)) \\ & = & A(0,A(0,A(1,1))) \\ & = & A(0,A(0,A(0,A(1,0)))) \\ & = & A(0,A(0,A(0,A(0,1)))) \\ & = & 5 \end{array}$$

Behauptung: Die Auswertung ist eindeutig und terminiert für alle x,y.

Wertabschätzungen:

$$\begin{array}{ll} A(0,y) > y & A(3,y) > 2^y \\ A(1,y) > S(y) & A(4,y) > \left.2^{2^{\cdot^{\cdot^{2}}}}\right\}y \\ A(2,y) > 2y & A(5,4) > 10^{10000} \end{array}$$

Satz: A ist **nicht** prim.rek.

Beweisansatz:
Lemma: Zu jeder prim.rek. Funktion $f : \mathbb{N}^n \to \mathbb{N}$ existiert k_f mit $A(k_f, max(\bar{x})) > f(\bar{x})$.
(Beweis über Aufbau der prim.rek. Funktionen.)

Es gibt Hierarchie der „mehrfach rekursiven Funktionen“ (Rekursion in m Argumenten für $m \geq 2$), siehe Buch von R.Peter.

2.8 μ-rekursive Funktionen

Zu $g : \mathbb{N}^{n+1} \to \mathbb{N}$ entsteht $f : \mathbb{N}^n \to \mathbb{N}$ durch Anwendung des μ-Operators (unbeschränktes Minimun), falls

$$f(\bar{x}) = \begin{cases} \text{kleinstes } y \text{ mit } g(\bar{x}, y) = 0, & \text{falls } \forall z < y : g(\bar{x}, y) \text{ definiert und } y \neq 0 \\ \perp, & \text{sonst.} \end{cases}$$

kurz: $f(\bar{x}) = \mu y \; g(\bar{x}, y) = 0$.

Beispiele:

- $g(x, y) = |x - y|$ $\quad f(x) = \mu y \; |x - y| = 0$
  $f(x) = x$
- $g(x, y) = x + y$ $\quad f(x) = \mu y \; x + y = 0$
  $$f(x) = \begin{cases} 0, & x = 0 \\ \perp, & x \neq 0 \end{cases}$$

f heisst *μ-rekursiv*, falls f aus den Basisfunktionen durch Anwendung von Einsetzung, prim.Rek. und μ-Operator entsteht.

Beispiel: „$3x + 1$“-Funktion

$$\begin{aligned} h(x) &= \begin{cases} x/2, & \text{falls } x \text{ gerade} \\ 3x + 1, & \text{falls } x \text{ ungerade} \end{cases} \\ g(x, y) &= h^y(x) \dot{-} 1 \\ f(x) &= \mu y \; g(x, y) = 0 \end{aligned}$$

Offen: Ist f total?

Bemerkung: f μ-rekursiv $\Longleftrightarrow$ f durch ein WHILE-Programm berechenbar?
Intuitive Korrespondenz:

$$\begin{array}{rcl} \text{Einsetzung} & \sim & \text{Verkettung} \\ \text{prim.Rek.} & \sim & \text{LOOP} \\ \mu\text{-Operator} & \sim & \text{WHILE} \end{array}$$

Bemerkung: Die Ackermann-Funktion ist μ-rekursiv.

Frage 1: Erfassen die μ-rekursiven Funktionen alle URM-berechenbaren Funktionen?

Frage 2: Nötige Anwendungen des μ-Operators in Anzahl beschränkbar?

2.9 Kleene'sches Normalformtheorem (KNFT)

Es gibt eine prim.rek. Funktion $U : \mathbb{N} \to \mathbb{N}$ und zu $n \geq 1$ jeweils ein $n+2$-stelliges prim.rek. Prädikat $T_n \subseteq \mathbb{N}^{n+2}$, so dass:
Jede URM-berechenbare Funktion $f : \mathbb{N}^n \to \mathbb{N}$ lässt sich darstellen als

$$f(\bar{x}) = U(\mu y \; T_n e\bar{x}y) \quad [:= \overline{sg}(\chi_{T_n}(e\bar{x}y)) = 0]$$

für geeignetes e, nämlich Code eines URM-Programms, das f berechnet.
kurz: $\varphi_e^{(n)}(\bar{x}) = U(\mu y \; T_n e\bar{x}y)$.

Beweis:
Definition von T_n:

$T_n \; e\bar{x}y \iff e$ ist Code eines URM-Programms (P_e) und P stoppt auf Eingabe $\bar{x}$ nach $[y]_1$ Schritten mit Konfiguration, deren Code $[y]_2$ ist. $(y = [[y]_1, [y]_2])$
Benutze Code $< l, z_1, \ldots, z_m >$ für Konfiguration $K = (l, z_1 = Wert(X_1), \ldots, z_m = Wert(X_m))$
Genügt Betrachtung von $X_1, \ldots, X_m$ für $m = m_n(e) =$ Max. von Stellenzahl n und grösstem in P_e auftretendem Variablenindex (m_n prim.rek.).
U extrahiert aus y das Ergebnis $(Wert(Y_1))$ der kodierten Konfiguration:
$U(y) =< [y]_2 >_1$, U prim.rek.

T_n prim.rek.:

1. Zeige, dass das Prädikat $Ce \iff e$ ist Programmcode prim.rek.
2. S_n „Schrittfunktion“ prim.rek. liefert Code von Konfiguration zu geg. Schrittzahl.
3. T_n

zu 1. Prüfe, ob $e =< code(I_1), \ldots, code(I_k) >$
im einzelnen: $< e >_{len(e)-1} = 0$

Für alle $x < len(e) - 1$ muss gelten $< e >_x \neq 0$, und:
wenn $[< e >_x]_1 \neq 0, [< e >_x]_2 \neq 0$ dann $1 \leq [< e >_x]_2 \leq len(e)$ prim.rek. gemäss Schema der beschränkten Allquantifizierung.

Zuvor: Test, ob e Folge kodiert.$[len(e) \neq 0]$

zu 2. Definiere:

$$S_n(e, \bar{x}, z) = \begin{cases} 0, & \text{falls nicht } Ce \\ 0, & Ce, Pe \text{ auf } \bar{x} \text{ stoppt vor Schritt } z \\ \text{Code der } z\text{-ten Konf.} & \\ \text{von } Pe \text{ auf } \bar{x}, & \text{sonst.} \end{cases}$$

S_n prim.rek.:

$$\begin{aligned}
S_n(e,\bar{x},0) &= \begin{cases} 0, & \text{falls } \neg Ce \\ p_0^{1+1} \cdot p_1^{x_2+1} \cdot \dots \cdot p_n^{x_n+1} \cdot p_{n+1}^{1} \cdot \dots \cdot p_{m_n(e)}^{1}, & \text{falls } Ce \end{cases} \\
&= c_0^{n+1} \cdot \overline{sg}(\chi_c(e)) + p_0^{1+1} \cdot \dots \cdot p_{m_n(e)}^{1} \cdot \chi_c(e)
\end{aligned}$$

$$< 1, x_1, \dots, x_n | 0, \dots, \underbrace{0}_{m_n(e)} >$$

$$S_n(e,\bar{x},S(z)) = \begin{cases} h_1(e,\bar{x},z,S_n(e,\bar{x},z)), & \text{Konfig. } S_n(e,\bar{x},z) \text{ verlangt } Incr \\ h_2, & \text{analog für Dekrement} \\ h_3, & \text{analog für Sprunganweisung} \\ 0, & S_n(e,\bar{x},z) \text{ kodiert Stopkonfig.} \end{cases}$$

Fallunterscheidung prim.rek. wie bei z=0.

Fall h_1:

$$\left.\begin{aligned} < e >_{aktuelleAnweisung} &= < e >_{<S_n(e,\bar{x},z)>_0 - 1} \\ &= [0,y] \end{aligned}\right\} (*)$$

$$h_1(e,\bar{x},S(z)) = S_n(e,\bar{x},z) \cdot \underbrace{p_0}_{\text{nächste Anw.}} \cdot \underbrace{p_k}_{\text{Werterhöhung}} \quad \text{mit } k = \mu y \leq e \ (*)$$

zu 3. $T_n e\bar{x}y$:

e ist Programmcode, $S_n(e,\bar{x},[y]_1) = [y]_2$ mit: $\underbrace{[y]_2 \text{ kodiert Stopkonfiguration}}_{<[y]_2>_0 = len(e)}$.

Bemerkungen zum T_n-Prädikat:

- $\exists y\ T_n e\bar{x}y \Longrightarrow$ solches y eindeutig bestimmt.
- $\exists y\ T_n e\bar{x}y \Longleftrightarrow e$ kodiert Programm, P_e stoppt auf $\bar{x}$.

Folgerungen aus KNFT:

1. Jede μ-rekursive Fkt. ist definierbar mit nur einer Anwendung des μ-Operators. Dazu: Aus Definition von f gehe zu URM-Programm über (induktiv), dann Anwendung von KNFT.
2. Programmiersprachliche Deutung:
   Jedes WHILE-Programm (mit LOOP- und WHILE-Schleifen) lässt sich umformen in ein WHILE-Programm mit nur einem WHILE-Statement.
   Beweis:
   WHILE-Programm $\Rightarrow$ URM-Programm $\Rightarrow$ KNFT-Form $\Rightarrow$ WHILE-Programm.

KNFT besagt: Jede URM-berechenbare Funktion ist μ-rekursiv.

Bemerkungen zu Äquivalenzbeweisen:

- **A**. Operationelle, nichtstrukturierte Modelle: URM-Programme, Flussdiagramme, TM's
- **B**. Kompositionelle Modelle: WHILE-Programme, rek. Funktionen, λ-Kalkül

Simulationen:

$$\begin{array}{rcl} A \longrightarrow A & : & \text{schrittweise Simulation} \\ B_1 \longrightarrow B_2 & : & \text{induktiv über } B_1 \\ B \longrightarrow A & : & \text{induktiv über } B \\ A \longrightarrow B & : & \text{Kodierung von } A\text{-Konfiguration durch Update-Schritte in } B \end{array}$$

B-Programm $\simeq$ B-Interpreter für A-Programme.

2.10 Existenz nicht-rekursiver Mengen

$$K = \{e|\exists y\ T_1 eey\} = \{e|\varphi_e(e)\downarrow\} = \{e|e \in W_e (= Def(\varphi_e)\}$$

Annahme: K rekursiv, λ_K rekursiv.

Definition:

$$\begin{array}{rcl} f(x) & = & \left\{ \begin{array}{ll} 0, & x \notin K \\ \perp, & x \in K \end{array} \right. \quad \text{(rekursiv)} \\ f(x) & = & \mu y\ (\chi_K(x)) + y = 0 \end{array}$$

wähle zu f ein e mit $f(x) = \varphi_e(x)$

$$\varphi_e(x)\downarrow \Leftrightarrow \varphi_e(x) = 0 \Leftrightarrow \varphi_x(x)\uparrow$$

$x = e$ Widerspruch!

$$Tot = \{e|\varphi_e \text{ total}\} = \{e|\forall x \exists y\ T_1 exy\}$$

Behauptung: Tot ist nicht rekursiv.

Zu e stellen wir folgendes Programm her:

$$\left.\begin{array}{c} X1 := 0 \\ X1 := e \\ P_e \end{array}\right\} P_{f(e)}, \quad f \text{ rekursiv}$$

$$
\begin{aligned}
P_e(e)\downarrow &\iff P_{f(e)}(x)\downarrow, \forall x \\
e \in K &\iff f(e) \in Tot \\
\chi_K(x) &= \chi_T ot(f(x))
\end{aligned}
$$

Da K nicht rekursiv ist folgt: Tot ist nicht rekursiv.

3 Universelle Funktionen und Selbstbezüglichkeit

Definiere $\quad \varphi^{n+1}(e, \bar{x}) = U(\mu y \; T_n e\bar{x}y)$

Bemerkung: Nach Def. ist φ^{n+1} rek. Für jedes n-stellige rek. f ex. e, so dass $f(\bar{x}) = \varphi^{n+1}(e, \bar{x})$.

Definition: Sei $\mathcal{K}$ eine Klasse von n-stelligen Funktionen.
$F : \mathbb{N}^{n+1} \to \mathbb{N}$ heisst *universell* für $\mathcal{K}$, falls

- für jedes $f \in \mathcal{K}$ ex. e mit $f(\bar{x}) = F(e, \bar{x})$
- für jedes $e \in \mathbb{N}$ ist die Funktion $\lambda\bar{x} \; F(e, \bar{x}) \in \mathcal{K}$

Satz: φ^{n+1} ist universell für Klasse RF_n der n-stelligen rekursiven Funktionen, und ist selbst rekursiv.

Definition:

- TRF_n sei Klasse der n-stelligen totalen rek. Funktionen
- PRF_n sei Klasse der n-stelligen prim.rek. Funktionen

Satz: Es gibt für TRF_n und PRF_n keine universelle Fkt., die selbst total und rekursiv, bzw. prim.rek. ist.

Beweis: TRF_1 (PRF_1 analog)
Annahme: $F : \mathbb{N}^2 \to \mathbb{N}$ universell für TRF_1 total, rek.
$f(x) = F(x, x) + 1$ total und rekursiv. Wähle e mit $F(e, x) = f(x)$
$x = e$: $F(e, e) = f(e) = F(e, e) + 1$

Konsequenz: Die totalen berechenbaren Funktionen haben keinen effektiven Programmierformalismus durch nummerierbare Programme P_i, so dass:

- Jedes P_i berechnet totale Funktion

- Jede totale berechenbare Funktion wird durch ein P_i berechnet
- Die Auswertungsfunktion $F(i,x) = P_i(x)$ ist berechenbar

Bemerkung: Man kann für die rekursiven Funktionen (partiell!) so **nicht** schliessen: kein Widerspruch durch Existenz einer universellen Funktion $f(x) = \phi^{(2)}(x,x)+1 \in RF_1$
wähle $e : f(x) = \phi^{(2)}(e,x)$
dann: $\phi^{(2)}(e,e) = f(e) = \phi^{(2)}(e,e) + 1$
$\rightarrow$ **kein** Widerspruch, da $\phi^{(2)}(e,e) = \perp$ sein kann.

Historische Bemerkung:
Idee der Diagonalisierung geht auf Cantor zurück:

$$|\mathbb{R}| > |\mathbb{N}|$$

Hier: Die Menge der unendlichen 0-1-Folgen ist überabzählbar.
Anmerkung: Abzählbarkeit diser Menge liefert Schema aller 0-1-Folgen:

$$\begin{array}{rcl} \beta_0 & = & \beta_0(0)\beta_0(1)\beta_0(2)\dots \\ \beta_1 & = & \beta_1(0)\beta_1(1)\beta_1(2)\dots \\ & \dots & \end{array}$$

Jede Folge kommt als ein β_i vor.
Setze $\beta(i) := \overline{\beta_i(i)}$
wähle e, so dass $\beta = \beta_e$

$$\beta_e(e) = \beta = \beta_e(e)$$

3.1 s-m-n-Theorem

Für $m,n \geq 1$ gibt es totale rekursive Funktion $s_n^m : \mathbb{N}^{m+1} \rightarrow \mathbb{N}$ mit

$$\varphi_e^{(m+n)}(\bar{x},\bar{y}) = \varphi_{s_n^m(e,\bar{x})}(\bar{y})$$

Beweis: Zu $e,\bar{x}$ setze $s_n^m(e,\bar{x})$ fest als Code für folgendes Programm $P[e,\bar{x}]$.
$P[e,\bar{x}]$: Bringe Argumente $\bar{y}$ von $X_1,\dots,X_n$ in Variablen $X_m+1,\dots,X_m+n$

$$X_1 := x_1,\dots,X_m := x_m; P_e$$

Dann ist die Gleichung erfüllt. Nach Def. ist s_n^m rekursiv.

3.2 Der Satz von Rice

$A \subseteq \mathbb{N}$ repräsentiere *semantische Programmeigenschaft*, wenn es eine Funktionenklasse $\mathcal{F}$ gibt mit

$$e \in A \Longleftrightarrow \varphi_e \in \mathcal{F}$$

Beispiel:

- „P_e hat 17 Zeilen" ist **nicht** semantisch
- „P_e terminiert immer" ist semantisch (mit $\mathcal{F} = Tot$)

A heisse *trivial*, falls $A = \emptyset$ oder $A = \mathbb{N}$.

Satz von Rice: Jede nichttriviale Menge, die semantische Programmeigenschaft repräsentiert, ist **unentscheidbar**.

Beweis: Zu A betrachte $\mathcal{F}$, wähle

$$\begin{array}{ll} a \in A & \varphi_a \in \mathcal{F} \\ b \notin A & \varphi_b \notin \mathcal{F} \end{array}$$

Benutze $K = \{x | \varphi_x(x) \downarrow\}$ unentscheidbar.
Ziel: Finde tot. berechenbares f:

$$x \in K \Longleftrightarrow f(x) \in A$$

Dann kann A nicht entscheidbar sein. Fertig.

Betrachte $c_\perp^{(1)} \notin \mathcal{F}$ (2.Fall analog)
Definiere

$$g(x,y) = \begin{cases} \varphi_a(y), & x \in K \\ \perp, & \text{sonst} \end{cases}$$

Es gilt: $x \in K \Longleftrightarrow s_1^1(x) \in A$

Dann: $x \in K \Rightarrow \varphi_{s_1^1(x)}(y) = \varphi_a(y) \Rightarrow s_1^1(x) \in A$ (da $a \in A$) $(*)$

Erinnerung: $x \in K \Longleftrightarrow \exists z \; T_1 xxz$

g rek., $g(x,y) = \varphi_e^{(2)}(x,y),\ smn : \varphi_e(x,y) = \varphi_{s_1^1(x)}(y), s_1^1$ tot.rek.

$(*) x \notin K$: $\varphi_{s_1^1(x)} = c_\perp^{(1)}$ ($\notin \mathcal{F}$, 1.Fall), $\quad s_1^1(x) \notin A$

3.3 Rekursionssatz

Sei $f : \mathbb{N} \to \mathbb{N}$ total, rekursiv. Dann existiert e mit $\varphi_e = \varphi_{f(e)}$.

Sabotage: $e \mapsto f(e), \varphi_e \neq \varphi_{f(e)}$

Beweis:
Betrachte

$$g(y,x) = \begin{cases} \varphi_z(x), & \text{falls } \varphi_y(y)\downarrow \text{ und } = z \\ \perp, & \text{sonst} \end{cases}$$

g rekursiv.
$smn : g(y,x) = \varphi_{s_1^1(y)}(x), s_1^1\ tot., rek.$
Also:

$$\varphi_{s_1^1(y)}(x) = \begin{cases} \varphi_{\varphi_y(y)}(x), & \text{falls } \varphi_y(y)\downarrow \\ \perp, & \text{sonst} \end{cases}$$

Definiere zu f gemäss Vorraussetzung eine Fkt. $h : \mathbb{N} \to \mathbb{N}$ mit $h(y) = f(s_1^1(y))$.
Da s_1^1, f tot., rek., ist auch h tot., rek.
wähle i mit $\varphi_i(y) = h(y)$, φ_i total, also $\varphi_i(i)\downarrow$.

$$\varphi_{s_1^1(i)}(x) = \varphi_{\varphi_i(i)}(x) = \varphi_{f(s_1^1(i))}(x)$$

wegen $\varphi_i(y) = h(y) = f(s_1^1(y))$ setze $e = s_1^1(i)$.

1. **Anwendung**: Es gibt ein (URM-)Programm, das für jeden Input ($\in \mathbb{N}$) seine eigene Codezahl e ausgibt.

   Beweis: $g(x,y) = x,\ smn : g(x,y) = \varphi_{s_1^1(x)}(y)$

   Nach Rekursionssatz existiert e mit $\varphi_e = \varphi_{s_1^1(e)}$

   $$\varphi_e(y) = \varphi_{s_1^1(e)}(y) = g(e,y) = e$$

   Also: P_e liefert für Eingabe y die Zahl e als Ausgabe.

2. **Anwendung**: Es existiert e, so dass: $W_e = e (= Def(\varphi_e))$.

   Betrachte

   $$g(x,y) = \begin{cases} 0, & \text{falls } x = y \\ \perp, & \text{sonst.} \end{cases}$$

   $smn : g(x,y) = \varphi_{s_1^1(x)}(y),\ Def(\varphi_{s_1^1(x)}) = x$ nach Definition von g.

   Rekursionstheorem liefert e: $\varphi_e = \varphi_{s_1^1(e)}$

   $$W_e = Def(\varphi_e) = Def(\varphi_{s_1^1(e)}) = e$$

3.4 Bemerkungen zur Selbstbezüglichkeit

1. Gödel's **Unvollständigkeitssatz** (1931) zeigt:

   Das Beweissystem der Peano-Arithmetik (Peano-Axiome für alle 1.Stufe Formeln, Beweiskalkül für Logik 1.Stufe) ist unvollständig: Es gibt einen in $(\mathbb{N}, +, \cdot, 0, 1)$ wahren Satz, der **nicht** herleitbar ist.

Ansatz: Wähle Satz, der sagt „Ich bin nicht beweisbar.“

Durchführung:

a) Kodiere Sätze, Formeln und Beweise durch Zahlen.

b) Konstruktion einer Formel

$$beweis(x, y) := \text{„}y \text{ ist Code eines Beweises für Satz } x\text{“}$$

Hieraus: $bewbar(x) := \exists y \ beweis(x, y)$

c) Betrachte $m \mapsto f(m)$ mit $f(m) =$ Code von $\neg bewbar(m)$
Rekursionssatz liefert e mit $f(e) = e, \quad e =$ Code von $\neg bewbar(e)$

d) Satz mit Nr. e sagt: „Ich bin nicht beweisbar.“

2. **Selbst(re-)produktion** in physikalischen Systemen benutzt Realisierung von Codes, kodierten Objekten.

Operationen:

Konstruktor	$Kon : e \mapsto P_e$
Kopierer	$Kop : e \mapsto$ neue Kopie von e
Reproduzierer	$Repr : e \mapsto P_e, e$ (Kombination von Kon und Kop)

Falls Reproduzierer einen Code besitzt (e_R) $Repr : e_R \mapsto Repr, e_R$.

Realisierung in biolog. Systemen:
Code eines Objekts(Lebewesen)=Genom(DNA)

$$\begin{aligned} Kon &: \text{ Protheinsynthesemechanismus} \\ Kop &: \text{ RNA-Polymerase} \\ Repr \text{ mit } e_R &: \text{ Zelle und ihr Genom} \end{aligned}$$

Vorraussetzung: Umgebung hält Substrate bereit (Proteinsuppe)

3. Von Neumann-Modell der **zellularen Automaten**.

Zell. Automat: (Z×Z)-Anordnung von Kopien eines einzigen endlichen Automaten. Konfiguration ist endliches Arrangement von Zuständen (sonst: Ruhezustand).

$$C, B \longrightarrow C, B, B \longrightarrow \ldots (C, B \text{ Automaten, auch für } B = C)$$

C wurde konstruiert durch v. Neumann mit 29 Zuständen, durch Conway mit 2 Zuständen.

4 Elementare Sachverhalte über aufzählbare Mengen

Signifikanz: A aufzählbar $\Longleftrightarrow$ die A-Elemente können durch systematischen Prozess sukzessiv erzeugt werden.

Musterbeispiel: Sätze einer axiomatisierten Theorie.

Intuitive **Definition**: $A = \{f(0), f(1), f(2), \ldots\}$ für ein totales, berechenbares f.
Bisher: $A = Def(g)$ für berechenbares g.

Satz: $\emptyset \neq A \subseteq \mathbb{N}$ ist $Bild(f(\mathbb{N}))$ einer tot. rek. Funktion $f \Longleftrightarrow A = Def(g)$ für rekursive Funktion g.

Beweis:

- "$\Longleftarrow$":
  Interessant $|A| = \infty$
  $g(x) = U\mu y\ Texy$

  Idee: Gehe Paare (x, y) mit $Texy$ durch und produziere jeweils x als Wert.

$$\begin{aligned} f(0) &= [\mu t\ Te\underbrace{[t]_1}_{x}\underbrace{[t]_2}_{y}]_1 \\ f(z+1) &= [\mu t\ t > f(z) \wedge Te[t]_1[t]_2]_1 \end{aligned}$$

- "$\Longrightarrow$":
  Zu f wähle Index e

$$\begin{aligned} z \in A &\Longleftrightarrow \exists x\ f(x) = z \\ &\Longleftrightarrow \exists x \exists y\ (Texy \wedge U(y) = z) \\ &\Longleftrightarrow \mu y\ (Texy \wedge U(y) = z) \\ &\Longleftrightarrow \exists t\ (Te[t]_1[t]_2 \wedge U([t]_2) = z) \end{aligned}$$

  $g(z) = \mu t(\ldots), \quad Def(g) = A.$

4.1 Reduktionsbeziehungen $\leq_1, \leq_m$

$$K = \{x | \varphi_e(e) \downarrow\} = \{x | \exists y \; Teey\}$$

Hieraus: K ist rekursiv aufzählbar (recursively enumerable):
Gehe alle Paare (e, y) durch und gebe e aus, wenn $Teey$ gilt.

$A, B \subseteq \mathbb{N}$:

$$A \leq_m B \Longleftrightarrow \text{ es ex. rek. tot. } f : \mathbb{N} \to \mathbb{N} \text{ mit: } x \in A \Longleftrightarrow f(x) \in B$$

schreibe $f : A \leq_m B$, $A \leq_1 B$: verlange zusätzlich, dass f injektiv.

Bemerkungen:

1. $\leq_1, \leq_m$ reflexiv, transitiv
2. $A \leq_1 B \Longrightarrow A \leq_m B$
3. $A \leq_1 B \Longrightarrow \overline{A} \leq_1 \overline{B}$, $A \leq_m B \Longrightarrow \overline{A} \leq_m \overline{B}$
4. $A \leq_m B, B$ rek. $\Longrightarrow A$ rek.

   $$\chi_A(x) = \chi_B(f(x)), \text{ für } f : A \leq_m B$$

5. $A \leq_m B, B$ rek. aufzählbar $\Longrightarrow$ A rek. aufzählbar, $f : A \leq_m B$, g rek. mit $Def(g) = B$

   $$g(f(x)) \, def. \Longleftrightarrow f(x) \in B$$

   $Def(f \circ g) = A$, $f \circ g$ rek.

Definition: A heisst 1-*vollständig*/*m-vollständig*:$\Longleftrightarrow$

1. A rek. aufzählbar
2. $\forall B \, (B$ rek. aufzählbar $\Longrightarrow B \leq_1 A / B \leq_m A)$

Satz: K ist m-vollständig.

Beweis: $x \in K \Longleftrightarrow \exists y \; Txxy$
Sei B rek. aufzählbar, zeige $B \leq_m K$: $B = Def(h)$, h rek.
Definiere

$$g(x, y) = \begin{cases} 0, & h(x) \downarrow \\ \perp, & \text{sonst} \end{cases}$$

$smn : g\,(x, y) = \varphi_{s_1^1(x)}(y)$

$$\begin{array}{rcl} x \in B & : & \varphi_{s_1^1(x)} = c_0^{(1)} \\ x \notin B & : & \varphi_{s_1^1(x)} = c^{(}_{\perp}1) \end{array}$$

$$x \in B \iff \varphi_{s_1^1(x)}(\underbrace{s_1^1(x)}_{y}) \downarrow \iff s_1^1(x) \in K$$

Also: $\quad s_1^1 : B \leq_m K$.

Verschärfung: K sogar 1-vollständig.

Aufgabe: Mache s injektiv (dann s^*)
$h(s(x), y)$ = Code von ($P_s(x)$ ergänzt um k Anweisungspaare $incr(X_1)decr(X_1)$ am Anfang)

$$\begin{array}{rrcl} & s^*(0) & = & s(0) \\ & s^*(s(x)) & = & h(s(x), y) \\ \text{für kleinste } y\text{:} & h(s(x), y) & \neq & s^*(0), \ldots, s^*(x) \end{array}$$

Übertragung des Beweises liefert:

$$A \text{ 1-vollständig} \iff A\ m\text{-vollständig.}$$

5 Reduktionsbegriffe und Grade

$A \equiv_1 B \;:\Longleftrightarrow A \leq_1 B, B \leq_1 A$
$A \equiv_m B :\Longleftrightarrow A \leq_m B, B \leq_m A$

1-Grad ist $\equiv_1$-Klasse
m-Grad ist $\equiv_m$-Klasse

Ziel 1: Vergleich von $\equiv_1, \equiv_m$

Ziel 2: Allgemeinere Reduktionsbegriffe

5.1 Satz von Myhill

$A \equiv_1 B \Longleftrightarrow A$ und B sind *rekursiv isomorph*, d.h es gibt eine rek. Bijektion $f : \mathbb{N} \to \mathbb{N}$ mit $x \in A \Leftrightarrow f(x) \in B$.

linke Seite liefert:
inj. $g : \mathbb{N} \to \mathbb{N}$ mit $x \in A \Longleftrightarrow g(x) \in B$
inj. $h : \mathbb{N} \to \mathbb{N}$ mit $y \in B \Longleftrightarrow h(y) \in A$

"$\Longleftarrow$" trivial: $g = f, h = f^{-1}$

5.2 Satz von Cantor-Schröder-Bernstein

Für beliebige Mengen A, B gilt: Falls injektive Funktionen $g : A \to B$ und $h : B \to A$ existieren, dann existiert Bijektion $f : A \to B$.

Beweis verlangt Kombinationen der geg. Funktionen $g : A \leq_1 B, h : B \leq_1 A$ zu Bij. f.

Ansatz: Erzeuge sukzessiv endliche A-B-Korrespondenzen: $(x_0, y_0), (x_1, y_1), \ldots, (x_k, y_k)$ mit

- $i \neq j \Longrightarrow x_i \neq x_j, y_i \neq y_j$

- $x_i \in A \Longleftrightarrow y_i \in B$

so dass schliesslich jedes $x \in \mathbb{N}$ ein x_i und jedes $y \in \mathbb{N}$ ein y_i.

Sorge in Schritt $2n$ dafür, dass $0, \ldots, n$ alle im Vorbereich, in Schritt $2n+1$, dass $0, \ldots, n$ im Nachbereich der A-B-Korrespondenzen auftreten.

Konstruktion: Setze $x_0 = 0, y_0 = g(0)$.
Seien $(x_0, y_0), \ldots, (x_k, y_k)$ konstruiert

- $k+1$ **gerade**:

  Wähle x_{k+1} kleinstes $x \neq x_0, \ldots, x_i$

  (a) falls $g(x_{k+1}) \neq y_0, \ldots, y_k$, setze $y_{k+1} = g(x_{k+1})$
  (b) sonst betrachte $y_i = g(x_{k+1})$

    (a) falls $g(x_i) \neq y_0, \ldots, y_k$ setze $y_{k+1} = g(x_i)$
    (b) sonst betrachte $y_i = g(x_i)$

      (a) falls $g(x_j) \neq y_0, \ldots, y_k$ setze $y_{k+1} = g(x_j)$
      (b) etc.

  Wegen $|y_0, \ldots, y_k| < |x_0, \ldots, x_{k+1}|$ tritt schliesslich Fall (a) ein.

- $k+1$ **ungerade**:

  Analog mit h statt g.
  Klar: Es entsteht eine Bijektion von $\mathbb{N}$ auf $\mathbb{N}$.

  $x_k \in A \Longleftrightarrow y_k \in B$: $\quad k = 0$ wegen $g : A \leq_1 B$
  $k+1$, etwa $k+1$ gerade und $y_{k+1} = g(x_j)$

$$\begin{aligned} x_{k+1} \in A &\Longleftrightarrow g(x_k+1) = y_i \in B \\ &\Longleftrightarrow x_i \in A \\ &\Longleftrightarrow g(x_i) = y_j \in B \Leftrightarrow x_j \in A \\ &\Longleftrightarrow g(x_j) = y_{k+1} \in B \end{aligned}$$

Beispiel:

x	$g(x)$	$h(x)$	k	x_k	y_k
0	2	3	0	0	2
1	5	2	1	3	0
2	0		2	1	5
3	6		3	2	1
4	1		4	4	6
$\vdots$	$\vdots$		$\vdots$	$\vdots$	$\vdots$

5.3 Produktive und kreative Mengen

$A \subseteq \mathbb{N}$ heisst *produktiv*, falls eine rekursive Funktion f existiert mit:

$$W_x \subseteq A \Longrightarrow f(x)\downarrow \text{ und } f(x) \in A \backslash W_x$$

"*f ist produktiv für A*"

Bemerkung:

$$A \text{ produktiv } \Longrightarrow A \textbf{ nicht} \text{ rekursiv aufzählbar}$$

A heisst *kreativ* $\Longleftrightarrow$ A ist rek. aufz., $\bar{A}$ produktiv

Beispiel:
K kreativ: rek. aufz., zeige $\bar{K}$ produktiv:
setze $f(x) = x$, betrachte $W_x \subseteq \bar{K}$.

Beh.: $x \in \bar{K} \backslash W_x$
Genügt $x \notin W_x$ (Dann $x \in \bar{K}, x \notin W_x$).
Wäre $x \in W_x$, auch $W_x \subseteq \bar{K}$ wegen Vorraussetzung.
Widerspruch zu Definition von K.

Satz:

$$A \text{ produktiv, } A \subseteq_m B \Longrightarrow B \text{ produktiv}$$

Vorbereitung für Beweis:
Wenn f total, rekursiv, dann $f^{-1}(W_x) = W_{g(x)}$ für eine rek. tot. Fkt. g.
Dann

$$f(y) \in W_x \Longleftrightarrow y \in W_{g(x)} \quad (*)$$

Definiere rek. Fkt. $h(x, y) = 0$, falls $\varphi_x(f(y))\downarrow$, sonst $\perp$

smn: $\varphi_{g(x)}(y) = h(x, y)$ für geeignetes g

Wegen $(*)$ gilt:

$$\begin{aligned} f(y) \in W_x &\Longleftrightarrow \varphi_x(f(y))\downarrow \\ &\Longleftrightarrow h(x,y) = 0 \\ &\Longleftrightarrow \varphi_{g(x)}(y)\downarrow \\ &\Longleftrightarrow y \in W_{g(x)} \end{aligned}$$

Beweis des Satzes:
Sei h prod. für A: $W_x \subseteq A \Longrightarrow h(x)\downarrow$ und $h(x) \in A - W_x$
Sei $f : A \leq_m B : x \in A \Longleftrightarrow f(x) \in B$, g zu f gewählt gemäss Vorbereitung.

Zeige $g \circ h \circ f$ produktiv für B:
Betrachte $W_x \subseteq B$: Wegen $f : A \leq_m B$ gilt

$$\underbrace{f^{-1}(W_x)}_{W_{g(x)}} \subseteq A$$

Da h produktiv für A: $h(g(x)) \downarrow, h(g(x)) \in A - W_{g(x)}$
Dann $f(h(g(x))) \downarrow$, zeige $f(h(g(x))) \in \underbrace{B}_{1} - \underbrace{W_x}_{2}$

1. klar wegen $h(g(x)) \in A$
2. bekannt: $h(g(x)) \notin W_g(x)$.
   wegen $(*)$ folgt $f(h(g(x))) \notin W_x$

5.4 Verbindungen zum Studium arithmetischer Theorien

Betrachte Sprache 1.Stufe mit $+, \cdot, 0, 1$. Identifiziere Sätze ϕ der Sprache mit Zahlen $\lceil \phi \rceil$.

$$Th = \{\lceil \phi \rceil | (\mathbb{N}, +, \cdot, 0, 1) \models \phi\}$$

Gödel's Kodierungstechnik liefert zu jedem e einen Satz ϕ_e mit

$$\begin{array}{rcl} \exists y\ Teey & \iff & \lceil \phi_e \rceil \in Th \\ (e \in \bar{K} \Leftrightarrow) \neg \exists y\ Teey & \iff & \lceil \phi_e \rceil \notin Th \\ & \iff & \lceil \neg \phi_e \rceil \in Th \end{array}$$

Somit $\bar{K} \leq_m Th$.
Bekannt: $\bar{K}$ produktiv, Somit: Th produktiv.

Proposition 1: Die Menge der wahren Sätze der Arithmetik 1.Stufe ist produktiv und somit **nicht** aufzählbar.

Betrachte PA (Peano-Arithmetik) als Menge der Zahlencodes der aus PA-Axiomen formal herleitbaren Sätze.
Analyse der Gödel'schen Kodierung liefert: Falls $Teem$ zutrifft, so $T\underline{eem}$ herleitbar.
($\underline{e}$ ist Term $\underbrace{1 + \cdots + 1}_{e}$), also auch $\underbrace{\exists y\ T\underline{ee}y \text{ herleitbar}}_{\text{Code } \in PA}$.

Daher $K \leq_m PA, \bar{K} \leq_m \overline{PA}$.

Proposition 2: Die Menge der in PA nicht herleitbaren Sätze ist produktiv, PA selbst also kreativ.

$$S = \{\lceil \phi_e \rceil | \neg \phi_e \text{ in } PA \text{ beweisbar}\}$$

Unterstelle: PA ist widerspruchsfrei.
Nach Def. ist S aufzählbar, ausserdem $S \subseteq \overline{PA}$.
($\neg\phi_e$ in $PA \Longrightarrow \phi_e$ nicht in PA beweisbar)

Da nach Prop. 2 $\overline{PA}$ produktiv, findet man aus (Index von) S ein e_0 mit $\lceil \phi_{e_0} \rceil \in \overline{PA} \backslash S$.

Proposition 3: ϕ_{e_0} nicht beweisbar in PA, $\neg\phi_{e_0}$ nicht beweisbar in PA (Unvollständigkeit von PA).

Ziel: Vergleich von $\equiv_1, \equiv_m$

Proposition:

1. Jede unendliche rek. aufz. Menge hat eine rek. unendliche Teilmenge
2. Jede produktive Menge hat eine unendl. rek. aufz. Teilmenge

Beweis:

1. Übung.
2. Sei A produktiv mit prod. Fkt. f:

$$\begin{array}{rll} \emptyset & \subseteq A & \text{, etwa } \emptyset = W_{x_0} : f(x_0) \in A(\backslash \emptyset) \\ \{x_0\} & \subseteq A & \text{, etwa } \{x_0\} = W_{x_1} : f(x_1) \in A(\backslash x_0) \\ \{x_0, x_1\} & \subseteq A & \text{, etwa } \{x_0, x_1\} = W_{x_2} : f(x_2) \in A(\backslash \{x_0, x_1\}) \\ & \dots & \end{array}$$

($\Longrightarrow \{x_0, x_1, \dots\} \subseteq A$)

$$\begin{array}{rcl} A \textbf{ \textit{immun}} & :\Longleftrightarrow & A \text{ unendlich, enthält keine } \infty \text{ rek. aufz. Teilmenge} \\ A \textbf{ \textit{einfach}} & :\Longleftrightarrow & A \text{ rek. aufz., } \overline{A} \text{ immun} \end{array}$$

Satz: Es gibt eine einfache Menge S.

Beweis:
Betrachte $h(x) = [\mu t\ Tx[t]_1[t]_2 \wedge [t]_1 > 2x]_1$
(erstes z, bei Durchsuchen aller Paare (z, y) mit $Txzy \wedge z > 2x$, wobei $[t]_1 = z, [t]_2 = y$)

$S := Bild(h)$

klar nach Definition: h rek., $Bild(h)$ rek. aufz.

Beachte: S enthält wenigstens ein Element jeder unendlichen rek. aufz. Menge W_x, klar nach Def. von h. (jedes ∞ W_x liefert $z \in W_x, z > 2x$)

Also: $\overline{S}$ enthält kein ∞ W_x ganz.

Somit: $\overline{S}$ enthält keine rek. aufz. Teilmenge.

Fehlt: $\overline{S}$ unendlich.
Nach Def. von S:

Für $x = 0$ wird verlangt $z > 0$.
Für $x = 1$ wird verlangt $z > 2$.
Für $x = 2$ wird verlangt $z > 4$.

Also in $\{0, 1, 2\}$ ist höchstens ein z.
Also in $\{0, 1, .., 4\}$ sind höchstens zwei z.
Also in $\{0, ..., 2k\}$ sind höchstens k S-Elemente(z).

In $\{0, .., 2k\}$ sind jeweils mindestens $k + 1$ $\overline{S}$-Elemente.
Also ist $\overline{S}$ unendlich.

Satz (Trennung von $\equiv_1, \equiv_m$-Klassen):
Sei A einfach, $k \notin A, B = A \cup k$.
Dann gilt:

1. B ist einfach
2. $A \leq_m B$
3. $B \leq_1 A$
4. $A \not\leq_1 B$

Beweis:

Index