ÜBUNGEN ZU "GRUNDBEGRIFFE DER MATHEMATISCHEN LOGIK" 2011SS

JAKOB KELLNER

Einfache Zahlentheorie

Sei $\mathbb{N} = \{0, 1, 2, \dots\}$ die Menge der natürlichen Zahlen (mit 0).

Bsp 1. Eine Polynomielle Paarfunktion.

Zeige:

> $f : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ definiert durch $f(x, y) = \frac{1}{2}(x + y)(x + y + 1) + y$ ist eine Bijektion.

Man kann die Zahl $f(x, y)$ als den "Code" von (x, y) bezeichnen. Stellen Sie sich $\mathbb{N} \times \mathbb{N}$ als Gitterpunkte im zweidimensionalen Cartesischen Koordinatensystem vor. Welche Reihung der Gitterpunkte ergibt diese Codierung? (D.h., welcher Punkt hat den Code 0, welcher 1 etc?)

Bsp 2. Kodierung endlicher Folgen.

Sei $p_0 = 2$, $p_1 = 3$, und allgemein p_n die $(n + 1)$-te Primzahl. Sei $\mathbb{N}^{<\omega}$ die Menge aller endlichen Folgen natürlicher Zahlen. Sie enthält also z.B. die leere Folge $()$, die Folgen (n) der Länge 1 für jedes $n \in \mathbb{N}$, die Folgen (n, m) der Länge 2 für jedes Paar $n, m \in \mathbb{N}$, etc.

Beweise:

> Die Abbildung $f : \mathbb{N}^{<\omega} \to \mathbb{N}$ definiert durch $(x_0, \dots, x_k) \mapsto p_0^{x_0} \cdot p_1^{x_1} \cdots p_k^{x_k - 1} - 1$ ist eine Bijektion.

Bsp 3. Der Chinesische Restsatz:

Beweise:

> Seien $m_1, \dots, m_n$ paarweise teilerfremde natürliche Zahlen, und $a_1, \dots, a_n$ beliebige natürliche Zahlen, dann gibt es eine natürliche Zahl b so dass $b \equiv a_i \mod m_i$ für $i = 1, \dots, n$.

Hüllenoperatoren

In der Vorlesung wird die Klasse der primitiv rekursiven Funktionen als die kleinste Klasse definiert, die bestimmte Funktionen erhält und unter bestimmten Operationen abgeschlossen ist.

Bsp 4. Hüllenoperatoren

Zeige/argumentiere:

- Diese Definition "funktioniert" (d.h., es gibt diese kleinste Klasse).
- Man kann diese Klasse "von oben" (als Schnitt aller abgeschlossenen Klassen) definieren sowie auch "von unten" (induktiv).
- Die Klasse der primitiv rekursiven Funktionen ist abzählbar unendlich.

Bem.: Solche Konzepte finden sich in der Mathematik ständig; in dieser einfachsten Form z.B. bei Vektorräumen in dem von einer Menge erzeugten linearen Teilraum, bei Körpern in dem von einer Menge erzeugten Unterkörper, bei Ringen in dem von einer Menge erzeugten Ideal.

Bsp 5. Cantorsche Diagonalisierung

Zeige: Es gibt eine Funktion $f : \mathbb{N} \to \mathbb{N}$ die nicht primitiv rekursiv ist.

Date: Version: 17. Mai 2011 23:22.

Primitiv rekursive Funktionen

Bsp 6. Subtraktion 1
Zeige:

$f : \mathbb{N} \to \mathbb{N}$ definiert durch $f(n) = \max(0, n-1)$ ist primitiv rekursiv.

Bsp 7. Subtraktion 2
Zeige:

$f : \mathbb{N}^2 \to \mathbb{N}$ definiert durch $f(n,m) = \max(0, n-m)$ ist primitiv rekursiv.

Bsp (wird ausgelassen). Exponentation
Zeige:

$f : \mathbb{N}^2 \to \mathbb{N}$ definiert durch $f(n,m) = n^m$ ist primitiv rekursiv.

Bsp 8. Division
Zeige:

$f : \mathbb{N}^2 \to \mathbb{N}$ definiert durch $f(n,m) = \lceil \frac{m}{n} \rceil$ für $n > 0$ (und 0 für $n = 0$) ist primitiv rekursiv.

(Dabei ist $\lceil r \rceil$ die kleinste ganze Zahl größer-gleich r.)

Bsp 9. Fallunterscheidungen
Zeige:

Wenn $f1_1, f_2, g : \mathbb{N}^k \to \mathbb{N}$ primitiv rekursiv sind, dann auch $h : \mathbb{N}^k \to \mathbb{N}$ definiert durch

$$h(\bar{x}) = \begin{cases} f_1(\bar{x}) & \text{falls } g(\bar{x}) > 0 \\ f_2(\bar{x}) & \text{sonst.} \end{cases}$$

LOOP Programme

Loop Programme sind im Vorlesungsskript definiert.

Bsp 10. Ein triviales Programm.
Gegeben sei das folgende LOOP Programm:

```
output=Zero()
loop input_1 times {
  output=Inc(output)
}
output=Inc(output)
```

Welche eindimensionale Funktion $f : \mathbb{N} \to \mathbb{N}$ wird durch dieses Programm berechnet?

Bsp 11. Ein verwirrenderes Programm.
Gegeben sei das folgende LOOP Programm:

```
output=Zero()
loop input_1 times {
  output=Inc(input_1)
  loop input_2 times {
    input_1=Inc(input_1)
  }
}
```

Welche zweidimensionale Funktion $g : \mathbb{N}^2 \to \mathbb{N}$ wird durch dieses Programm berechnet?

Bsp (wird ausgelassen). Addition
Schreibe ein LOOP Programm für die Addition, d.h. die Funktion $f : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ definiert durch $(x, y) \mapsto x + y$.

Bsp 12. Primzahlentest

Schreibe ein LOOP Programm, das auf input $x \in \mathbb{N}$ den output $f(x)$ liefert, wobei f die charakteristische Funktion der Primzahlen ist, d.h. $f(x) = 1$ wenn x Primzahl und $f(x) = 0$ sonst.

Bsp 13. Redundante Befehle

Zeige: Zu jedem LOOP Programm gibt es ein äquivalentes Programm, das nur die Befehle `Inc` und `Zero` (sowie die FOR-Schleife) verwendet.

Bsp 14. Ohne Schleifen

Die Schleife kann man natürlich nicht weglassen. Konkreter: Es gibt ein LOOP Programm das man nicht durch ein anderes ersezten kann, das keine Schleifen verwendet. (Welche Funktionen können LOOP Programme ohne Schleifen berechnen?)

Hyperoperationen

Wir definieren folgende Funktionen:

$$x \uparrow^0 y = x \cdot y, \quad x \uparrow^{n+1} 0 = 1, \quad x \uparrow^{n+1} (y+1) = x \uparrow^n (x \uparrow^{n+1} y).$$

Es gilt also $x \uparrow^1 y = x^y$. Die Funktion $x \uparrow^2 y$ ist der "Potenzturm" "x hoch x hoch x ..." (y mal).

In den folgenden Beispielen werden wir sehen:

- Für jedes fixe n ist die zweistellige Funktion $\uparrow^n$ primitiv rekursiv. (Das ist sehr einfach.)
- Aber die dreistellige Funktion $\uparrow$ ist nicht primitiv rekursiv. (Das ist mühsamer.)

Bsp 15. Zeige: Für jedes n ist die zweistellige Funktion $\uparrow^n$ primitv rekursiv.

Bsp 16. Zeige:

(1) $2 \uparrow^n 1 = 2$
(2) $2 \uparrow^n 2 = 4$
(3) $2 \uparrow^{n+1} 3 = 2 \uparrow^n 4$.

Bsp 17. Zeige: Für jedes n wächst die einstellige Funktionen $2 \uparrow^n : \mathbb{N} \to \mathbb{N}$ (d.h., $y \mapsto 2 \uparrow^n y$) streng monoton.

Hinweis: "doppelte" Induktion, Induktionsanfang in der inneren Induktion durch Bsp 16.

Bsp 18. Zeige: Für alle $n \in \mathbb{N}$ und alle $y \in \mathbb{N}$ mit $y \geq 3$ gilt: $2 \uparrow^{n+1} y \geq 2 \uparrow^n (y+1)$.

Bsp 19. (aufwändigere Zusatzaufgabe) Zeige: Für jede primitiv rekursive Funktion $f : \mathbb{N}^k \to \mathbb{N}$ gibt es ein $n \in \mathbb{N}$, so dass für alle $x_1, \ldots, x_k \in \mathbb{N}$ gilt:

$$f(x_1, \ldots, x_k) \leq 2 \uparrow^n (x_1 + \ldots + x_k + 3).$$

Hinweis:

(1) Zeige: Für jedes y wächst die einstellige Funktione f, die n auf $2 \uparrow^n y$ abbildet, monoton. (Ähnliche doppelte Induktion wie in Bsp 17.)
(2) Zeige: Für $n \geq 1$ ist $2 \uparrow^n (y + c) \geq 2^c \cdot (2 \uparrow^n y)$.
(3) Zeige: Für alle $k, c, n \in \mathbb{N}$ gibt es ein $N \in \mathbb{N}$, so dass gilt:

$$k \cdot \big(2 \uparrow^n (y + c)\big) \leq 2 \uparrow^N (y + 3) \text{ für alle } y \in \mathbb{N}.$$

(Benutze Bsp 18.)
(4) Dann erfolgt der Beweis mit induktion nach Aufbau der primitiv rekursiven Funktion (und unter Verwendung der vorherigen Beispiele).

Bsp 20. Zeige:

- Die dreistellige Funktion $\uparrow : \mathbb{N}^3 \to \mathbb{N}$, $(x, y, n) \mapsto x \uparrow^n y$ ist nicht primitiv rekursiv.
- Die einstellige Funktion f die n auf $2 \uparrow^n n$ abbildet ist nicht primitiv rekursiv.

(Benutze Bsp 19.)

Bsp 21. Ist für jedes y die einstellige Funktion f die n auf $2 \uparrow^n y$ abbildet primitv rekursiv? (Beweis.)

Universelle Programme und Halteproblem

In der Vorlesung wurden (totale) rekursive Funktionen (d.h., totale μ-rekursive Funktionen) und GOTO Programme $\mathcal{M}$ definiert. Jedem solchen Programm $\mathcal{M}$ (gemeinsam mit einer Stelligkeit k) entspricht eine (im Allgemeinen partielle[1]) Funktion $f_k^{\mathcal{M}} : \mathbb{N}^k \to \mathbb{N}$.

Etwas genauer: Sei $\mathcal{M}$ ein GOTO Programm, k eine natürliche Zahl und $\bar{x} \in \mathbb{N}^k$. Wenn das Programm $\mathcal{M}$ auf input $\bar{x}$ hält, dann ist $f_k^{\mathcal{M}}(\bar{x})$ definiert und gleich dem output. Ansonsten ist $f_k^{\mathcal{M}}(\bar{x})$ undefiniert.

Es stellt sich heraus:

- Sei $\mathcal{M}$ ein GOTO Programm, $k \in \mathbb{N}$ und sei $f_k^{\mathcal{M}}$ total. Dann ist $f_k^{\mathcal{M}}$ (total) rekursiv.
- Sei $f : \mathbb{N}^k \to \mathbb{N}$ rekursiv. Dann gibt es ein GOTO Programm $\mathcal{M}$ so dass $f = f_k^{\mathcal{M}}$.
- (Normalformensatz) Für jedes k gibt es eine einstellige partiell rekursive Funktionen g und eine $k+2$-stellige rekursive funktion h mit der folgenden EIgenschaft: Für jedes GOTO-Programm $\mathcal{M}$ gibt es eine natürliche Zahl e so dass

  $$f_k^{\mathcal{M}}(\bar{x}) = g(\mu(t : h(\bar{x}, e, t)) > 0)$$

  (Genauer: $f_k^{\mathcal{M}}(\bar{x})$ ist genau dann definiert wenn es ein t gibt mit $k(\bar{x}, e, t) > 0$, in diesem Fall ist $f_k^{\mathcal{M}}(\bar{x})$ gleich $g(t)$ für das minimale t mit dieser Eigenschaft.)

Bsp 22. Cantorsche Diagonalisierung

Es gibt überabzählbar viele Funktionen von $\mathbb{N}$ nach $\mathbb{N}$. Das wird durch den Cantorschen Diagonalbeweis gezeigt: Angenommen $(f_i)_{i\in\mathbb{N}}$ wäre eine komplette Liste aller Funktionen. Setzte $g(n) := f_n(n) + 1$. Dann kann g nicht eines der f_i sein. (Warum?)

Wir wissen aber, dass es nur abzählbar viele rekursive Funktionen von $\mathbb{N}$ nach $\mathbb{N}$ gibt. (Warum?)

Versuche den Cantorschen Diagonalbeweis auf GOTO Programme anzuwenden: Sei f_i die rekursive Funktion, die "vom i-ten Programm" berechnet wird. Setzt $g(n) := f_n(n) + 1$. Ist g GOTO berechenbar? (Warum/warum nicht?) Wie passt das zur Behauptung, dass es nur abzählbar viele rekursive Funktionen gibt?

Bsp 23. Universelles Programm (GOTO Interpreter in GOTO Sprache)

Zeige: Es gibt ein "universelles" GOTO Programm $\mathcal{U}$ mit folgender Eigenschaft: Für jedes GOTO-Programm $\mathcal{M}$ gibt es ein $e \in \mathbb{N}$ (den "source code" von $\mathcal{M}$, als Zahl kodiert) so dass für jedes $x \in \mathbb{N}$ der output des Programmes $\mathcal{M}$ auf input x dasselbe ist wie der output des Programmes $\mathcal{U}$ auf (zweidimensionalen) Input (e, x) (insbesondere: der eine Wert ist definiert genau dann wenn es der andere ist). In anderen Worten:

$$(\forall x \in \mathbb{N})\ f_2^{\mathcal{U}}(e, x) = f_1^{\mathcal{M}}(x)$$

Hinweis: Das ist eine sehr einfache Anwendung der oben angeführten Tatsachen, und verlangt kaum Verständnis von GOTO Programmen.

Bemerkung: Das Resultat sieht kompliziert aus, ist aber inhaltlich nicht weiter überraschend: Man kann in der GOTO Sprache einen GOTO-Interpreter schreiben. (Dasselbe gilt natuerlich für Basic, C, Java oder jede andere "vernünftige" Programmiersprache die Ihnen einfällt.)

Bsp 24. (Universelle rekursive Funktion?)

Gibt es eine "universelle" zweistellige rekursive Funktion $f : \mathbb{N}^2 \to \mathbb{N}$ so dass es für jede rekursive Funktion g ein e gibt so dass für alle $x \in \mathbb{N}$

$$f(e, x) = g(x)?$$

Hinweis: GOTO berechnbar und rekursiv scheinen zwar "dasselbe" zu sein, aber vielleicht sollten Sie zuerst die folgenden Beispiele rechnen bevor Sie die Frage beantworten...

[1]Eine partielle Funktion $f : A \to B$ ist eine Funktion deren Definitionsbereich eine Teilmenge A' von A ist (man sagt $f(a)$ ist definiert gdw $a \in A'$, sonst ist $f(a)$ undefiniert). Wenn $A' = A$ ist, dann heisst f total. Zwei partielle funktionen f, g sind gleich, $f = g$, wenn f und g auf genau demselben Definitionsbereich definiert sind und dort dieselben Werte haben. Das ist stärker als: "Wenn $f(a)$ (definiert und gleich) b ist, dann ist auch $g(a) = b$". (Letzteres wäre ja zB auch für folgende Funktionen erfüllt: f ist nirgendwo definiert, g ist die Identität.

Bsp 25. (Halteproblem)

Sei $\mathcal{U}$ das universelle Programm aus Bsp 23. Setze $\varphi_e(x) := f_2^{\mathcal{U}}(e, x)$. D.h.: $\varphi_e(x)$ ist der output des GOTO Programms mit source code e auf input x (wenn definiert, und undefiniert sonst). Setze

$$H := \{e : \varphi_e(e) \text{ ist definiert}\}.$$

(Das ist eine Variant der "Haltemenge".)

Zeige: Die charakteristische Funktion von H (die x auf 1 abbildet wenn $x \in H$ und auf 0 sonst) ist nicht rekursiv.

Bemerkung: Anders ausgedrückt: Es gibt kein GOTO Programm $\mathcal{M}$ das auf input x (immer terminiert und) 1 ausgibt wenn $x \in H$ und 0 sonst.

Hinweis: Ein Diagonal-Argument, wie im Beweis dass es überabzählbar viele Folgen gibt: Ansonsten wäre die Funktion f rekursiv die n auf $\varphi_n(n) + 1$ abbildet wenn $n \in H$ und auf 0 sonst.

Bsp 26. Rekursionstheorie mit totalen Funktionen.

Angenommen $(\psi_e)_{e\in\mathbb{N}}$ ist eine Aufzählung von bestimmten (totalen) einstelligen rekursiven Funktionen, für die es einen "Interpreter" gibt: Eine zweistellige rekursive Funktion U' so dass $U'(e, x) = \psi_e(x)$ für alle $e, x \in \mathbb{N}$. Zeige: Es gibt eine (totale) rekursive Funktion f die ungleich jedem ψ_e ist.

Bemerkung: Dieses Beispiel deutet an, dass man in gewissem Sinn keine Programmiersprache definieren kann in der verhindert wird dass man versehentlich "Endlosschleifen" programmieren kann, aber gleichzeitig alle (totalen) rekursiven Funktionen programmieren kann.

WHILE Programme

WHILE-Programme sind definiert wie LOOP-Programme, nur dass zusätzlich noch die folgende neue Art von Schleifen erlaubt ist.

```
while x do {
  ...
}
```

Wieder kann `x` durch einen beliebigen Variablennamen ersetzt werden, und die drei Punkte stehen für ein beliebiges WHILE-Programm. Der Schleifenkörper wird nur dann ausgeführt, wenn der Wert der Variable `x` nicht Null ist. Nach Ausführung des Schleifenkörpers wird `x` aber erneut inspiziert. Wenn der Wert dann immer noch nicht Null ist, wird die Schleife erneut ausgeführt.

WHILE-Programme sind näher an modernen Programmiersprachen als GOTO-Programme. Sie sind in der Regel übersichtlich und leichter zu verstehen.

Bsp 27. WHILE kann man durch GOTO ersetzen

Zeige: Zu jedem WHILE-Programm gibt es ein äquivalentes GOTO-Programm.

Bsp 28. GOTO kann durch (eine) WHILE Schleife ersetzt werden

Zeige: Zu jedem GOTO Programm gibt es ein äquivalentes WHILE Programm. Man kommt sogar immer mit einer einzigen WHILE Schleife (und dazu weiteren FOR Schleifen) aus.

Bemerkung: Für eine (totale) funktion f ist also folgendes äquivalent: f ist rekursiv; f ist durch ein WHILE Programm berechenbar; f ist durch ein GOTO Programm berechenbar. (Und es stellt sich heraus dass jedes "normale" Computermodell bzw Programmiersprache denselben Berechenbarkeitsbegriff ergibt; zB Java, Prolog, Lisp, C++, Turing Maschinen etc etc. Man setzt rekursiv daher oft einfach mit "berechenbar" gleich.)

Zusammenfassung der Begriffe und Konzepte der letzten Beispiele

- Ein (GOTO) Programm kann als Zeichenfolge (Menge von Programmzeilen) aufgefaßt werden; äquivalent dazu können wir diese Zeichenfolge in eine einzige natürliche Zahl e "kodieren". Wir **identifizieren** im Weiteren das Programm und den Kode. (Die Zuordnung vom Programm zum Kode ist injektiv.) **Source code** ist einfach eine andere Bezeichnung für ein Programm (oder, äquivalent, für den Kode des Programms).

(Die kodierung eines Programms in eine Zahl wurde in der Vorlesung durchgeführt, sie wird unter anderem in Normalformensatz verwendet.)

- Zu jedem Programm (=jedem Kode) e gibt es eine (eindeutige) von e berechnete (einstellige) Funktion. Im weiteren nennen wir diese Funktion φ_e oder φ_e^1.
  Natürlich berechnet e für jedes k eine k-stellige Funktion φ_e^k. (Da ein Programm e nur endlich viele (Input)variablen verwendet, sagen wir `input_1` bis `input_N`, kann φ_e^k natürlich höchstens von $x_1, \ldots, x_N$ abhängen, auch wenn k größer als N ist.)
- Eine partielle Funktion $f : \mathbb{N}^k \to \mathbb{N}$ heißt **GOTO berechenbar**, wenn es ein GOTO Programm e gibt mit $f = \varphi_e^k$. (Insbesondere: e hält auf Input n gdw $f(n)$ ist definiert). Statt GOTO berechenbar sagen wir auch **partiell rekursiv** oder einfach **berechenbar**.
- Jedes GOTO Programm e berechnet also (genau eine) einstellige partiell rekursive Funktion φ_e^1. Und zu jeder partiell rekursiven Funktion f gibt es ein Programm e mit $f = \varphi_e^1$. Die Zuordnung von source code zur berechneten Funktion ist aber nicht injektiv! Zu jedem f gibt es unendlich viele verschiedene e mit $f = \varphi_e^1$.
- Eine **totale**, rekursive Funktion heißt **rekursiv** (oder auch **total rekursiv**). Die total rekursiven Funktionen sind also eine Teilmenge der partiell rekursiven. Und auch zu jeder total rekursiven Funktion f gibt es nicht nur ein, sondern unendlich viele Programme e die f berechnen (d.h., $f = \varphi_e^1$).
- Natürlich gibt es nur abzählbar unendlich viele partiell rekursive Funktionen. (Warum?) Es gibt also jedenfalls Funktionen (auch totale) die nicht partiell rekursiv sind.
- Bsp 23 zeigt: Es gibt ein universelles Programm $\mathcal{U}$ das gegeben einen source code e und einen input x die Berechnung des Programmes e auf x simuliert (d.h., den output $\varphi_e^1(x)$ liefert).
- Daraus folgt also sofort: Es gibt eine universelle **partiell** rekursive Funktion $G : \mathbb{N}^2 \to \mathbb{N}$, d.h.: $G(e, x) = \varphi_e^1(x)$.
  Kann man eine partiell rekursive Diagonalfunktion $g(n) = G(n, n) + 1$ definieren? (Ja.) Was folgt daraus? (Nichts. Kein Widerspruch, weil G partiell ist.)
- Es gibt aber keine universelle total rekursive Funktion, d.h. eine zweistellige total rekursive Funktion G' so daß es für jede partiell rekursive Funktion f ein e gibt mit $G'(e, \cdot) = f(\cdot)$. (Ansonsten liefert die Diagonalfunktion g eben doch einen Widerspruch.)
- Das Halteproblem ist unlösbar: Es gibt kein Programm das entscheidet ob $G(n, n)$ definiert ist oder nicht.
  (Ansonsten könnte man die Diagonalfunktion g modifizieren so daß $g(n) = 0$ falls $G(n, n)$ undefiniert. Wäre das Halteproblem entscheidbar, dann wäre dieses g rekursiv und würde einen Widerspruch liefern.)

Rekursive und rekursiv aufzählbare Mengen

- Eine Menge $A \subseteq \mathbb{N}^k$ (für ein $k > 0$) heißt rekursiv, wenn ihre charakteristische Funktion (die natürlich total ist) eine rekursive Funktion ist.
- Eine Menge $A \subseteq \mathbb{N}^k$ heißt rekursiv aufzählbar (engl.: recursively enumerable, r.e.), wenn es ein rekursives $B \subseteq \mathbb{N}^{k+1}$ gibt so daß A die Projektion von B
- Sei f eine k-stellige partiell rekursive Funktion. $\mathrm{dom}(f) \subseteq \mathbb{N}^k$ ist die Menge aller Werte auf denen f definiert ist.

In der Vorlesung wird gezeigt: Für $A \subseteq \mathbb{N}$ ist Folgendes äquivalent:

- A ist r.e.
- A ist das Bild einer partiell rekursiven Funktion. D.h., es gibt ein f partiell rekursiv so dass $m \in A$ genau dann wenn es ein n gibt mit $f(n) = m$ (und insbesondere $f(n)$ definiert.
- A ist entweder leer oder das Bild einer (totalen) rekursiven Funktion.

Bsp 29. Weiter Äquivalenz von r.e.

Zeige: $A \subseteq \mathbb{N}^k$ ist r.e. genau dann wenn $A = \mathrm{dom}(f)$ für eine partiell rekursive Funktion $f : \mathbb{N}^k \to \mathbb{N}$ ist. (D.h., $n \in A$ gdw $f(n)$ definiert.)

Hinweis: Wenn A r.e., dann ist A die Projektion einer rekursiven Menge $B \subseteq \mathbb{N}^{k+1}$. Dann kann man $f(\bar{x})$ zum Beispiel definieren als das kleinste y so daß $(\bar{x}, y) \in B$. Für die andere Richtung, verwende den Normalformensatz.

Bsp 30. Noch eine Äquivalenz von r.e.
Zeige: $A \subseteq \mathbb{N}^k$ ist r.e. genau dann wenn A die Projektion einer **primitiv** rekursiven Menge $B \subseteq \mathbb{N}^{k+1}$ ist.
Hinweis: Wieder einmal Normalformensatz.

Bsp 31. Eine universelle r.e. Menge
Zeige: Es gibt eine r.e. Menge $B \subseteq \mathbb{N}^2$ so daß es für jede r.e. Menge $A \subseteq \mathbb{N}$ ein e gibt mit $A = \{n : (e, n) \in B\}$.

Bsp 32. Eine universelle rekursive Menge?
Gibt es eine universelle rekursive Menge, d.h. eine rekursive Menge $B \subseteq \mathbb{N}^2$ so daß es für jede rekursive Menge $A \subseteq \mathbb{N}$ ein e gibt mit $A = \{n : (e, n) \in B\}$?

Bsp 33. Halteproblem 2
Zeige: Die Haltemenge H aus Bsp 25 ist r.e., aber nicht rekursiv.

Bsp 34. Halteproblem 3
Zeige:

- Die Menge $H^* = \{(e, x) : x \in \mathrm{dom}(\varphi_e)\} \subseteq \mathbb{N}^2$ ist r.e., aber nicht rekursiv.
  (D.h., es ist unentscheidbar ob ein Programm m auf input x hält oder nicht.)
- Die Menge $H^0 = \{e : 0 \in \mathrm{dom}(\varphi_e)\}$ ist nicht rekursiv.
  (D.h., es ist unentscheidbar ob ein Programm m auf input 0 hält oder nicht.)
  Hinweis: Gegeben ein source code e, modifiziere e zu einem neuen source code e' so daß e' auf input 0 denselben output hat wie der ursprünglich source code e. Die Abbildung von e nach e' ist rekursiv.

Rekursive Definition der partiell rekursiven Funktionen

In der Vorlesung wurden (totale) rekursive Funktionen definiert durch Abschluss unter gewissen Operationen (Einsetzung, primitve rekursion, μ-Rekursion). Wir können natürlich auch die partiell rekursiven Funktionen auf solche Art definieren, zB folgendermassen: Die Menge Rekursiv* ist die kleinste Menge von partiellen Funktionen, welche alle primitiv rekursiven Funktionen enthält und abgeschlossen ist unter:

- Zusammensetzung (von partiellen Funktionen): Wenn $f: \mathbb{N}^\ell \to \mathbb{N}$ und $g_1, \ldots, g_\ell: \mathbb{N}^k \to \mathbb{N}$ alle in Rekursiv* sind, dann auch die durch
  $$h(\bar{x}) = f(g_1(\bar{x}), \ldots, g_\ell(\bar{x}))$$
  gegebene partielle Funktion $h: \mathbb{N}^k \to \mathbb{N}$. Dabei ist $h(\bar{x})$ definiert genau dann wenn alle Funktionswerte $g_1(\bar{x}), \ldots, g_\ell(\bar{x})$ definiert sind.
- μ-Rekursion (von partiellen Funktionen): Wenn die partielle Funktion $f: \mathbb{N}^{k+1} \to \mathbb{N}$ in Rekursiv* ist, dann ist auch die folgendendermaßen definierte Funktion $g: \mathbb{N}^k \to \mathbb{N}$ in Rekursiv*: $g(\bar{x}) = y$ genau dann wenn
  $$f(\bar{x}, 0) = 0, \ldots, f(\bar{x}, y-1) = 0, f(\bar{x}, y) > 0$$
  (insbesondere müssen all diese Funktionswerte definiert sein!) Wenn es kein solches y gibt dann ist $g(\bar{x})$ undefiniert. Wir schreiben auch $g(\bar{x}) = \mu y(f(\bar{x}, y) > 0)$.

Achtung: Wenn $f : \mathbb{N} \to \mathbb{N}$ die konstante Nullfunktion ist, dann ist $f \circ g$, d.h. die Funktion $x \mapsto f(g(x))$, i.A. nicht die konstante Nullfunktion! (Wenn $g(x)$ undefiniert ist für ein bestimmtes x, dann ist auch $f \circ g(x)$ undefiniert.)

Bsp 35. Zeige: Rekursiv* ist das selbe wie "partiell rekursiv" (d.h. durch ein GOTO Programm berechenbar).
Hinweis: Für die eine Richtung, wie immer, Normalformensatz.

Andere Definitions- und Zielbereiche

Sei $f : \mathbb{N}^k \to \mathbb{N}^m$ eine partielle Funktion. Für $1 \leq i \leq m$ Setze $f_i := \pi_i^m \circ f$, eine partielle Funktion von $\mathbb{N}^k$ nach $\mathbb{N}$. Es ist also $f(\bar{x}) = (f_1(\bar{x}), \ldots, f_m(\bar{x}))$.

Wir definieren: $f : \mathbb{N}^k \to \mathbb{N}^m$ ist partiell rekursiv genau dann wenn jedes $f_i : \mathbb{N}^k \to \mathbb{N}$ rekursiv ist.

Fixiere m. In der Vorlesung wurde eine Kodierung $\varphi : \mathbb{N}^m \to \mathbb{N}$ vorgestellt (d.h., φ ist eine bijektiv Funktion, ausserdem ist φ primitiv rekursiv, und für wenn $1 \leq i \leq m$ ist dann ist $(\varphi^{-1})_i := \pi_i^m \circ \varphi^{-1}$ primitiv rekursiv.

Bsp 36. Zeige:

- $f : \mathbb{N}^m \to \mathbb{N}$ ist partiell rekursiv genau dann wenn $f \circ i^{-1} : \mathbb{N} \to \mathbb{N}$ partiell rekursiv ist.
- $f : \mathbb{N}^k \to \mathbb{N}^m$ ist partiell rekursiv genau dann wenn $i \circ f : \mathbb{N}^k \to \mathbb{N}$ rekursiv ist.

Bsp 37. (Zusatzaufgabe) Entscheidungsproblem für diophantische Gleichungen.

Man kann natürlich nicht nur $\mathbb{N}^k$, sondern alle möglichen anderen (abzählbaren) Mengen nach $\mathbb{N}$ kodieren. Zeige:

- Gib einen (informellen) Algorithmus für folgendes Problem an: Gegeben ein Polynom in einer Variable mit ganzzahligen Koeffizienten, hat dieses Polynom eine ganzzahlige Nullstelle? (Anders geschrieben: Gegeben $f \in \mathbb{Z}[X]$, gibt es ein $y \in \mathbb{Z}$ s.d. $f(y) = 0$?)
- Gib eine Kodierung $\varphi : \mathbb{Z}[X] \to \mathbb{N}$ an.
- Argumentiere: Die Menge $A \subseteq \mathbb{Z}[X]$ der Polynome mit ganzzahliger Nullstelle (genauer: das Bild von A unter der Kodierung φ, $\varphi[A] \subseteq \mathbb{N}$) ist rekursiv.
- Gib eine Kodierung $\varphi : \mathbb{Z}[X_1, \ldots, X_9] \to \mathbb{N}$ an. Argumentiere: Die Menge der Polynome in $\mathbb{Z}[X_1, \ldots, X_9]$ mit ganzzahliger Nullstelle ist r.e. (Bem.: Der Satz von Matiyasevich sagt aus, daß diese Menge *nicht* rekursiv ist.)

Mehr über rekursive Funktionen

Bsp 38. Busy Beaver Funktion

Sei $BB(n)$ der größtmögliche Output, der durch irgendein GOTO Programm mit $\leq n$ Programmzeilen auf Input 0 erzeugt wird. ($BB(n)$ ist wohldefiniert, weil es (abgesehen von irrelevanten Variablennamen) nur endlich viele Programme mit mit $\leq n$ Zeilen gibt.) BB ist also eine totale Funktion. Zeige:

(a) BB ist nicht rekursiv.
(b) Sei g eine rekursive Funktion. Dann gibt es ein $x_0 \in \mathbb{N}$ so dass $BB(x) > g(x)$ für alle $x \geq x_0$.

Hinweis zu (b): OBdA (warum?) sei g monoton. Sei g durch das Programm M berechnet. Zeige: Es gibt ein $d \in \mathbb{N}$ und für jedes $l \in \mathbb{N}$ ein Programm M^l mit $l + d$ Zeilen so dass der Output von M^l auf input 0 größer als $g(2l)$ ist.

Bsp 39. Partiell rekursiv

Sei $f : \mathbb{N} \to \mathbb{N}$ eine partielle Funktion so dass $f(x) = 0$ für alle $x \in \mathrm{dom}(f)$. Ist f partiell rekursiv?

Bsp 40. Entscheidungsproblem für partiell rekursive Funktionen

Sind die folgenden Fragen algortihmisch entscheidbar (d.h., rekursiv):

- Gegeben der source code e eines GOTO-Programms (oder äquivalent: die rekursiv*-Konstruktion/Definition einer Funktion), ist die berechnete/definierte Funktion die konstante Nullfunktion?
- Gegeben der source code eines LOOP Programms (oder äquivalent: die primitiv-rekursive Konstruktion/Definition einer Funktion), ist die berechnete/definierte Funktion die konstante Nullfunktion?

Bsp 41. Das S^n_m-Theorem

Man kann natürlich auf Source-codes selbst ebenfalls rekursive Transformationen anwenden, und das Ergebnis ist insgesamt rekursiv. (Wir haben das schon in Bsp 34 verwendet.)

Zeige:

- Sei $f : \mathbb{N} \to \mathbb{N}$ rekursiv. Dann ist die Funktion $g : \mathbb{N}^2 \to \mathbb{N}$ die (m, x) auf $\varphi^1_{f(m)}(x)$ abbildet, partiell rekursiv.
- Sei $g : \mathbb{N}^2 \to \mathbb{N}$ partiell rekursiv. Dann gibt es eine rekursive funktion $f : \mathbb{N} \to \mathbb{N}$ so daß $\varphi^1_{f(m)}(x) = g(m, x)$ für alle m, x.

(Die Definition von φ steht auf Seite 6.)

Bsp 42 Schwierigere Zusatzaufgabe. Fixpunktsatz

Zeige: Für jede rekursive Funktion ("source code Transformation") $f : \mathbb{N} \to \mathbb{N}$ gibt es einen sourcecode e so daß $\varphi^1_e = \varphi^1_{f(e)}$.

Hinweis: Ein Beweis findet sich u.A. im Wikipedia Eintrag zu "Kleene's recursion theorem".

Bsp 43. Einfache Anwendung des Fixpunktsatzes 1

(a) Sei f folgende sourcecode Transformation: Gegeben das Programm (d.h., den sourcecode) e, sei $f(e)$ das folgende neue Programm: Zuerst wird e ausgeführt, dann der output um 1 erhöht. Ist f rekursiv? Wenn ja, wie sieht ein Fixpunkt e bzgl f aus? (D.h. was wissen wir über die Funktion φ_e die $\varphi_e = \varphi_{f(e)}$ erfüllt?)

(b) Sei g folgende sourcecode Transformation: Gegeben das Programm (d.h., den sourcecode) e, sei $g(e)$ das Programm das (konstant) 0 ausgibt wenn $e \in H$ und (konstant) 1 sonst. Ist g rekursiv? Wenn ja, wie sieht ein Fixpunkt e bzgl g aus?

Bemerkung: $h(x) = x + 1$ ist natürlich eine andere Funktion als die oben definierte Funktion f. Was sagt uns der Fixpunktsatz für h? (Hinweis: wenig.)

Bsp 44. Einfache Anwendung des Fixpunktsatzes 2

Zeige: Es gibt ein Programm mit Code e das die konstante e Funktion berechnet (d.h., seinen eigenene source code ausgibt).

Bsp 45 Schwierige Zusatzaufgabe. Satz von Rice

Wir haben gesehen das das Halteproblem unentscheidbar ist. Wir können die (äquivalente) Variante für H_0 folgendermaßen formulieren: Die Menge $H_0 = \{e : \varphi_e(0) \text{ ist definiert}\}$ ist nicht rekursiv. (D.h., es ist unentscheidbar ob, ein source code e auf input 0 terminiert.)

Sei $\mathcal{A}$ eine nichttriviale Eigenschaft partiell rekursiver Funktionen. Anders formuliert: sei PART die Menge der partiell rekursiven Funktionen. Dann ist $\mathcal{A}$ eine Teilmenge von PART; nichtleer (d.h. es gibt mindestens eine partiell rekursive Funktion mit Eigenschaft $\mathcal{A}$) und nicht gleich PART (d.h. es gibt mindestens eine partiell rekursive Funktion die die Eigenschaft $\mathcal{A}$ nicht hat).

Beweise den Satz von Rice: Sei $\mathcal{A} \subset$PART nichttrivial. Dann ist $\{e : \varphi_e \in \mathcal{A}\}$ ist nicht rekursiv.

D.h., für keine nichttriviale Eigenschaft einer Funktion ist es entscheidbar, ob die durch e berechnete Funktion diese Eigenschaft hat oder nicht.

Aussagenlogik

Aus dem Skriptum:

Bsp 46. Skriptum Bsp 3.1

Bsp 47. Skriptum Bsp 3.2

Bsp 48. Skriptum Bsp 3.3

Bsp 49. Skriptum Bsp 3.4

Bsp 50. Skriptum Bsp 3.5