Was können Algorithmen?

Sabine Koppelberg

GLIEDERUNG

1. Intuitive Berechenbarkeit - der Begriff des Algorithmus
2. Registermaschinen
3. Rekursive Funktionen
4. Codierungen
5. Registermaschinen-berechenbare Funktionen sind rekursiv. Ein universelles Registermaschinen-Programm
6. Die Churchsche These
7. Das Halteproblem
8. Rekursiv aufzählbare Mengen
9. Turingmaschinen
10. Komplexität
11. Literatur

1. Intuitive Berechenbarkeit – der Begriff des Algorithmus

Der naiv arbeitende Mathematiker stellt sich Funktionen, etwa von der Menge $\mathbb{N}$ der natürlichen Zahlen in sich, gewöhnlich als durch eine Berechnungsvorschrift gegeben vor. Das Ziel dieses Kapitels ist es, zu zeigen, daß sich einerseits der intuitive Begriff der Berechenbarkeit von Funktionen mathematisch präzisieren läßt, andererseits nicht jede, insbesondere nicht jede aus Fragestellungen der Mathematik oder Informatik heraus interessante, Funktion berechenbar ist.
Wir nennen eine k-stellige Funktion $f : \mathbb{N}^k \to \mathbb{N}$ [1] *(intuitiv) berechenbar*, wenn es einen Algorithmus gibt, der, angesetzt auf beliebige Werte $a_1, \ldots, a_k \in \mathbb{N}$, den Funktionswert $f(a_1, \ldots, a_k)$ berechnet. Analog heiße eine k-stellige Relation über $\mathbb{N}$, d.h. eine Teilmenge R von $\mathbb{N}^k$, *(intuitiv) entscheidbar*, falls es einen Algorithmus gibt, der, angesetzt auf $a_1, \ldots, a_k \in \mathbb{N}^k$, entscheidet, ob $(a_1, \ldots, a_k)$ zu R gehört. Dabei ist ein *Algorithmus* ein mechanisches Verfahren, das nach endlich vielen festgelegten Regeln vorgeht. Es soll *deterministisch* sein, d.h. nach

[1] $f : D \to W$ bedeutet, daß f Funktion von D nach W ist. Die Zuordnungsvorschrift wird oft durch $x \mapsto f(x)$ angegeben. Für eine Menge A ist A^k die Menge aller k-tupel $(a_1, \ldots, a_k)$ mit $a_1, \ldots, a_k \in A$.

jedem Rechenschritt soll der nächste eindeutig festliegen und nicht dem Zufall oder der Willkür des Ausführenden überlassen sein (siehe aber 10.4 für einen liberalisierten Algorithmenbegriff). Ferner soll für beliebige Eingaben $a_1, \dots, a_k$ nach endlich vielen Schritten der Wert $f(a_1, \dots, a_k)$ berechnet bzw. die Frage "ist $(a_1, \dots, a_k) \in R$?" entschieden sein.

1.1 Beispiele (a) Die durch

$$(x,y) \mapsto x+y\ , \qquad (x,y) \mapsto x \cdot y$$

gegebenen Funktionen von $\mathbb{N}^2$ nach $\mathbb{N}$ sind berechenbar - die entsprechenden Algorithmen werden (etwa für in Dezimalschreibweise gegebene Zahlen) in der Grundschule gelehrt; ebenso die partiellen (d.h. nicht überall definierten) Funktionen mit

$$(x,y) \mapsto x-y\ , \qquad (x,y) \mapsto x:y\,.$$

(b) Die Funktion, die jedem Zahlenpaar den größten gemeinsamen Teiler zuordnet, wird durch den Euklidischen Algorithmus berechnet.
(c) Die Menge der Primzahlen (eine Teilmenge von $\mathbb{N}$) ist entscheidbar.
(d) Die (zweistellige) Kleiner-Relation auf $\mathbb{N}$

$$R = \{(x,y) \in \mathbb{N}^2 : x < y\}$$ [2]

ist entscheidbar.

Auch auf anderen, „effektiv gegebenen "Bereichen sind die Begriffe der Entscheidbarkeit bzw. Berechenbarkeit sinnvoll. Ein effektiv gegebener Bereich ist dabei eine abzählbare Menge, deren Elemente sich auf „effektive"Weise, etwa durch endliche Folgen von Symbolen über einem endlichen Alphabet, darstellen lassen — siehe Abschnitt 4.

1.2 Beispiele (a) Die Menge $\mathbb{Z}$ der ganzen Zahlen ist ein effektiv gegebener Bereich (z.B. lassen sich ganze Zahlen durch Paare, bestehend aus einem Vorzeichen $+$ oder $-$, und einer natürlichen Zahl darstellen), und die vier Grundrechenarten sind auf $\mathbb{Z}$, soweit definiert, berechenbar. Das Gleiche gilt für den (durch Vorzeichen und ein Paar von natürlichen Zahlen) effektiv gegebenen Bereich $\mathbb{Q}$ der rationalen Zahlen.
(b) Für festes $n \in \mathbb{N}$ sind die Mengen $\mathrm{Mat}(n,n)$ aller rationalen $n \times n$-Matrizen und $\mathbb{Q}^n$ aller n-tupel über $\mathbb{Q}$ effektiv gegebene Bereiche; die Menge

$$M = \{(A,b) : A \in \mathrm{Mat}(n,n), b \in \mathbb{Q}^n\ ,\ \text{das lineare Gleichungssystem } A \cdot x = b \text{ hat genau eine Lösung } x\}$$

[2] $\{x \in M : E(x)\}$ bezeichnet die Menge aller Elemente von M, die die Eigenschaft $E(x)$ besitzen.

ist entscheidbar, und die Funktion, die jedem Paar $(A, b) \in M$ die Lösung x zuordnet, ist berechenbar - etwa nach dem Gaußschen Algorithmus.
(c) Die Menge FOR aller aussagenlogischen Formeln über einer abzählbaren Menge von Aussagenvariablen ist ein efffektiv gegebener Bereich, und die Teilmengen VAL der allgemeingültigen bzw. SAT der erfüllbaren Formeln werden durch das bekannte Wahrheitstafelverfahren entschieden.

Ein informales und auch recht unkonstruktives Argument zeigt, daß nicht alle Funktionen, etwa von $I\!N$ nach $I\!N$, berechenbar sein können. Nach Cantors Diagonalverfahren ist nämlich die Menge K aller Funktionen von $I\!N$ nach $I\!N$ überabzählbar: Wäre K abzählbar, etwa $K = \{f_n : n \in I\!N\}$, so wäre die durch

$$d(n) = f_n(n) + 1$$

definierte Diagonalfunktion nicht in K, ein Widerspruch. Andererseits ist die Menge aller Algorithmen und damit die Menge der berechenbaren Funktionen abzählbar: Wir können uns die Algorithmen als in deutscher Sprache mit endlich vielen mathematischen Sonderzeichen geschrieben vorstellen; ein Algorithmus soll eine endliche Folge von Anweisungen sein. Damit sind Algorithmen i.w. endliche Symbolfolgen über einem endlichen Alphabet, und die Menge aller Algorithmen ist abzählbar. - In Abschnitt 7 werden wir übrigens eine präzisierte und ins Konstruktive gewendete Fassung dieses Arguments benutzen.
Die Berechenbarkeit einer gegebenen Funktion f läßt sich überzeugend durch die Angabe eines Algorithmus zur Berechnung von f nachweisen. Aber wie zeigt man Nicht-Berechenbarkeit? Hierzu ist offenbar eine mathematische Präzisierung des Berechenbarkeitsbegriffs notwendig. Zunächst macht man sich klar, daß nur der Begriff der berechenbaren Funktion präzisiert zu werden braucht, denn eine k-stellige Relation R auf $I\!N$ ist genau dann entscheidbar, wenn die durch

$$\chi_R(x_1 \ldots x_k) = \begin{cases} 1 & \text{falls } (x_1 \ldots x_k) \in R \\ 0 & \text{falls } (x_1 \ldots x_k) \notin R \end{cases}$$

definierte charakteristische Funktion $\chi_R : I\!N^k \to I\!N$ von R berechenbar ist.
Bei dem heute jedermann selbstverständlichen Einsatz von Computern wäre eine naheliegende Definition der Berechenbarkeit: f heißt berechenbar, wenn sich ein Algorithmus zur Berechnung von f in einer genügend mächtigen Programmiersprache schreiben läßt. Aber auch hier stellen sich die Fragen: In welcher Sprache? Leisten alle Programmiersprachen das Gleiche, oder könnten noch zu entwickelnde Sprachen möglicherweise mehr Funktionen berechnen als die heute bekannten?
In Abschnitt 3 dieses Kapitels wird die Klasse der rekursiven Funktionen definiert; Abschnitt 6 beantwortet unsere Frage in Form folgender Behauptung.

Churchsche These Die berechenbaren Funktionen sind genau die rekursiven.

Nach der Klärung des Berechenbarkeitsbegriffs (bzw. aufgrund der Churchschen These) ist es dann möglich, auf präzise Art die Unentscheidbarkeit konkret gegebener Probleme zu zeigen. Wir tun das in Abschnitt 7 nur für ein einziges, für die theoretischen Grundlagen der Informatik wichtiges Problem, das Halteproblem. Jedoch haben sich zahlreiche weitere Probleme aus Mathematik und Informatik als unentscheidbar erwiesen. Wir wollen hier ein berühmtes mathematisches Problem nennen, das sich ziemlich leicht formulieren läßt.

1.3 Beispiel (10. Hilbertsches Problem, 1900 von Hilbert ausgesprochen) Wir bezeichnen mit Pol den (effektiv gegebenen) Bereich aller Polynome mit ganzzahligen Koeffizienten in endlich vielen Variablen, die etwa aus der abzählbaren Menge $x_1, x_2, x_3, \ldots$ stammen mögen. Die Teilmenge L von Pol bestehe aus denjenigen Polynomen, die eine ganzzahlige Nullstelle haben. Z. B. gehört $p(x_1, x_2) = x_1^2 + x_2^2 - 1$ zu L ($x_1 = 1, x_2 = 0$ ist eine ganzzahlige Nullstelle von p), aber $q(x_1, x_2) = x_1^2 + x_2^2 + 1$ nicht. Erst 1970 wurde von dem russischen Mathematiker Yu. V. Matijasevič (nach wesentlichen Vorarbeiten anderer) gezeigt, daß L unentscheidbar ist.

In diesem Kapitel wird eine wesentliche Idealisierung gemacht: Wir versuchen nur eine Antwort auf die Frage, welche Funktionen *prinzipiell*, insbesondere ohne Beschränkung von Speicherplatz und Rechenzeit, berechenbar sind. In der Praxis kann man die Frage nach den zur Berechnung einer Funktion notwendigen Ressourcen natürlich keineswegs vernachlässigen. In Abschnitt 10 gehen wir auf diesen Aspekt kurz ein.

2. Registermaschinen

Wir führen in diesem Abschnitt Registermaschinen als ein Modell der Berechenbarkeit ein sowie eine Sprache RM zur Programmierung von Registermaschinen. Die Sprache RM kann als eine stark abgemagerte Version etwa von BASIC gelten, und die Arbeitsweise einer Registermaschine als die eines sehr simplen Rechners, der unter BASIC läuft. Obwohl die Sprache RM extrem schwach wirkt, wird sich in den Abschnitten 5 und 6 herausstellen, daß (unter Annahme der Churchschen These) jede intuitiv berechenbare Funktion durch ein RM- Programm berechnet werden kann.

Vereinbarungen $\mathbb{N} = \{0, 1, 2, \ldots\}$ ist die Menge der natürlichen Zahlen; insbesondere ist 0 eine natürliche Zahl. k-tupel $(x_1 \ldots x_k)$ kürzen wir, solange die Länge k festliegt, mit $\bar{x}$ ab.

2.1 Aufbau einer Registermaschine Eine Registermaschine hat abzählbar unendlich viele Speicherplätze und kann in jedem dieser Plätze eine (beliebig große) natürliche Zahl speichern. Die Speicherplätze sind, bis auf den Programmzählerplatz PZ, durch Variablen $X_1, X_2, X_3, \ldots\ldots, Y, Z_1, Z_2, Z_3, \ldots\ldots$

benannt. $X_1, X_2, X_3, \ldots\ldots$ heißen die Eingabevariablen, Y ist die Ausgabevariable, $Z_1, Z_2, Z_3, \ldots\ldots$ sind die Hilfsvariablen. Ist zu einem Zeitpunkt t die natürliche Zahl a im Speicherplatz V gespeichert, so bezeichnen wir a als den Wert von V zum Zeitpunkt t. Der Programmzähler speichert die Nummer der jeweils auszuführenden Zeile des Programms.– Man sollte sich hier vergegenwärtigen, daß die Annahme, unendlich viele Speicherplätze zur Verfügung zu haben und in einem Speicherplatz beliebig große Zahlen speichern zu können, eine in der Realität keineswegs erfüllte Idealisierung in sich birgt.

PZ			
X_1	X_2	X_3	$\ldots$
Y			
Z_1	Z_2	Z_3	$\ldots$

2.2 Programme in der Sprache RM Programme in der Sprache RM dürfen folgende *Symbole* bzw. Worte benutzen:

1. Symbole für die in 2.1 genannten Variablen $X_1, X_2, X_3, \ldots\ldots, Y, Z_1, Z_2, Z_3, \ldots\ldots$; zur Bezeichnung einer beliebigen Variablen schreiben wir auch $V, W, \ldots$
2. sog. *Marken* $A_1, A_2, A_3, \ldots$ zur Steuerung von Sprungbefehlen; für beliebige Marken schreiben wir auch $B, C, M \ldots$ und für eine Marke, die das Ende des Programms signalisiert E
3. die Worte „if“, “goto”
4. die acht Sonderzeichen $\leftarrow, +, -, 0, 1, \neq, [,]$.

Die Sprache RM hat die *Befehle*

$V \leftarrow V + 1$	(wobei V beliebige Variable sei)
$V \leftarrow V - 1$	(V beliebige Variable)
if $V \neq 0$ goto M	(V Variable, M beliebige Marke) .

Eine *Programmzeile* ist ein Befehl oder ein Befehl, dem eine in Klammern gesetzte Marke [M] vorangeht.

Ein *Programm* ist eine endliche geordnete Folge $P = (z_1, \ldots, z_n)$ von Programmzeilen; die Zahl n ist die *Länge* von P. Wir stellen uns die Zeilen von P untereinandergeschrieben vor:

$$\begin{array}{c} z_1 \\ z_2 \\ \cdot \\ \cdot \\ \cdot \\ z_n\,. \end{array}$$

Die eben gegebene Definition von RM-Programmen ist unter dem Gesichtspunkt einer möglichst einfachen Formulierung entworfen. Dafür nimmt man einige scheinbare Ungereimtheiten in Kauf. Z. B. dürfen in einem Programm P mehrere Zeilen mit derselben Marke beginnen; es dürfen Zeilen der Form “if $V \neq 0$ goto M” vorkommen, auch wenn keine Zeile von P mit der Marke M beginnt; die Länge n von P darf 0 sein - dann ist P das “leere” Programm.

2.3 Ausführung von Programmen Sei $P = (z_1, \ldots, z_n)$ ein RM-Programm und $(a_1 \ldots a_k)$ ein k-tupel von natürlichen Zahlen; wir erklären, wie eine Berechnung gemäß P mit den Eingabewerten $a_1, \ldots, a_k$ verläuft. Induktiv wird für jedes $t \in \mathbb{N}$ der Inhalt aller Speicherplätze zum Zeitpunkt t definiert:
Zu Beginn der Rechnung, d.h. im Zeitpunkt $t = 0$, gibt man den Eingabevariablen $X_1, \ldots, X_k$ die Werte $a_1, \ldots, a_k$, allen übrigen Variablen (auch der Ausgabevariablen Y) den Wert 0. Der Programmzähler erhält den Wert 1, da mit der ersten Zeile des Programms begonnen werden soll.
Im Zeitpunkt t habe der Programmzähler den Wert i, d.h. es soll die i-te Programmzeile z_i ausgeführt werden. Es sind mehrere Fälle zu unterscheiden.
Fall 1. In z_i steht ein Befehl der Form $V \leftarrow V + 1$. Dann wird im Zeitpunkt $t+1$ der Wert der Variablen V um 1 erhöht, und der neue Wert des Programmzählers ist $i + 1$. Die Werte der übrigen Variablen bleiben unverändert.
Fall 2. In z_i steht ein Befehl der Form $V \leftarrow V - 1$. Im Zeitpunkt $t + 1$ wird der Wert von V um 1 erniedrigt, falls er nicht Null war, sonst bleibt er Null. Der neue Wert des Programmzählers ist wieder $i+1$. Die Werte der übrigen Variablen bleiben unverändert.
Fall 3. In z_i steht ein Befehl “if $V \neq 0$ goto M”. Im Zeitpunkt $t + 1$ bleiben die Werte aller Variablen unverändert. Der neue Wert des Programmzählers ist :

> $i + 1$, falls V den Wert 0 hat
>
> das kleinste j zwischen 1 und n, für das die Zeile z_j mit der Marke M beginnt, falls solch ein j existiert und V nicht den Wert 0 hat
>
> $n + 1$, falls V nicht den Wert 0 hat und keine Zeile z_j mit der Marke M beginnt.

Da die Programmzeile z_{n+1} nicht existiert, bedeutet der letzte Fall ein “Herausspringen” aus dem Programm. Die Berechnung ist dann beendet; der Wert der Ausgabevariablen Y zu diesem Zeitpunkt ist das Ergebnis der Berechnung (gemäß P mit den Eingabewerten $a_1, \ldots, a_k$).

2.4 Beispiel Das folgende Programm, das wir durch $\mathbf{V} \leftarrow \mathbf{0}$ abkürzen wollen, setzt den Wert der Variablen V auf 0:

$$\begin{array}{ll} [M] & V \leftarrow V - 1 \\ & \text{if} \quad V \neq 0 \text{ goto } M\,. \end{array}$$

2.5 Makros Bereits geschriebene Programme können als Unterprogramme, sogenannte *Makros*, in komplexeren Programmen benutzt werden. Dabei sind einige Vorsichtsmaßnahmen nötig, die wir nur andeuten. Z.B. könnten ein Makro Q und ein Programm P, in das man Q einsetzen möchte, einige Variablen und Marken gemeinsam benutzen. Beim Einsetzen von Q in P wird man dann zweckmäßigerweise einige oder alle diese Variablen und Marken in Q systematisch umbenennen, so daß ihre Bedeutungen innerhalb von Q nicht mit denen innerhalb von P kollidieren. Wir geben einige nützliche Makros an, die ihrerseits freizügig von früheren Gebrauch machen.

(a) goto **M** sei folgendes Makro, das die Hilfsvariable Z benutzt:

$$\begin{array}{l} Z \leftarrow Z+1 \\ \text{if} \quad Z \neq 0 \text{ goto } M\,. \end{array}$$

(b) Das Makro $\mathbf{V} \leftarrow \mathbf{V}+\mathbf{W}$ ersetzt den Inhalt von V durch die Summe der Inhalte von V und W:

$$\begin{array}{ll} [A] & \text{if} \quad W \neq 0 \text{ goto } B \\ & \text{goto } C \\ & \\ [B] & W \leftarrow W-1 \\ & Z \leftarrow Z+1 \\ & V \leftarrow V+1 \\ & \text{goto } A \\ [C] & \text{if} \quad Z \neq 0 \text{ goto } D \\ & \text{goto } E \\ [D] & Z \leftarrow Z-1 \\ & W \leftarrow W+1 \\ & \text{goto } C\,. \end{array}$$

Im Programmteil hinter der Marke B wird W zu V addiert und gleichzeitig nach Z kopiert; im Programmteil hinter D wird Z nach W zurückkopiert. Der Effekt dieser Maßnahme ist, daß nach Ausführung des Programms W den ursprünglichen Wert hat, der möglicherweise später noch benutzt wird.

(c) $\mathbf{V} \leftarrow \mathbf{W}$ ist ein Makro, das W nach V kopiert; der Wert von W bleibt erhalten:

$$\begin{array}{l} V \leftarrow 0 \\ V \leftarrow V+W\,. \end{array}$$

(d) $\mathbf{Y} \leftarrow \mathbf{X_1}+\mathbf{X_2}$ addiert die Werte von X_1 und X_2 und speichert das Ergebnis in Y:

$$\begin{array}{l} Y \leftarrow X_1 \\ Y \leftarrow Y+X_2\,. \end{array}$$

2.6 Die von einem Programm berechnete Funktion; partielle Funktionen Sei $P = (z_1, \ldots, z_n)$ ein RM-Programm und k eine natürliche Zahl. Wir

definieren die durch P berechnete Funktion φ_P aus $I\!N^k$ nach $I\!N$ (die Stellenzahl k berücksichtigen wir in der Notation nicht) folgendermaßen. Für $\overline{a} = (a_1, \ldots, a_k)$ sei, falls die Berechnung endet, $\varphi_P(\overline{a})$ das Ergebnis der Berechnung gemäß P mit dem Eingabetupel $\overline{a}$, wie es in 2.3 definiert wurde, d.h. $\varphi_P(\overline{a})$ ist der Wert der Ausgabevariablen Y am Ende der Berechnung. Endet die Berechnung nicht, so ist $\varphi_P(\overline{a})$ undefiniert.
φ_P ist also möglicherweise nicht für alle Eingabetupel $\overline{a}$ definiert. Wir bezeichnen Funktionen f, die auf einer beliebigen Teilmenge von $I\!N^k$ definiert sind und nach $I\!N$ gehen, als *partielle* Funktionen aus $I\!N^k$ nach $I\!N$ und schreiben

$$f : I\!N^k \hookrightarrow I\!N\,.$$

Eine auf ganz $I\!N^k$ definierte Funktion bezeichnen wir als *total*.
Eine Funktion $f : I\!N^k \hookrightarrow I\!N$ nennen wir *RM-berechenbar*, wenn es ein Programm P gibt, so daß f die von P berechnete k-stellige Funktion φ_P ist. Z.B. berechnet das Programm

$$\begin{array}{ll} [M] & V \leftarrow V + 1 \\ & \text{if} \quad V \neq 0 \text{ goto } M \end{array}$$

die "leere", d.h. nirgends definierte einstellige Funktion. Beispiel (d) in 2.5 zeigt, daß die Additionsfunktion auf $I\!N$ RM-berechenbar ist. Auch die Multiplikation und die Potenzierung auf $I\!N$ sind RM-berechenbar - die entsprechenden Programme sollte der Leser, nach dem Vorbild von 2.5(d), durch iterierte Anwendung der Addition bzw. der Multiplikation selbst schreiben können.

2.7 Weitere Makros für Funktionen und Prädikate (a) Ist f die durch das Programm P berechnete k-stellige Funktion, so können wir P als Makro auffassen, das wir mit $\mathbf{W} \leftarrow \mathbf{f}(\mathbf{V_1}, \ldots, \mathbf{V_k})$ bezeichnen.
(b) Die k-stelligen Relationen auf $I\!N$ entsprechen eindeutig ihren charakteristischen Funktionen, d.h. den Funktionen von $I\!N^k$ in die Menge $\{0, 1\}$. Solche Funktionen nennen wir auch *Prädikate*; ihre Werte 1 bzw. 0 identifizieren wir mit den Wahrheitswerten wahr und falsch:

$$1 = \text{wahr}, \qquad 0 = \text{falsch}\,.$$

Für ein Prädikat $p : I\!N^k \to \{0, 1\}$ und $\overline{a} = (a_1 \ldots a_k) \in I\!N^k$ schreiben wir auch $p(\overline{a})$ ("p trifft auf $\overline{a}$ zu") statt $p(\overline{a}) = 1$. Wird p durch das Programm P berechnet, so können wir das Makro

$$\begin{array}{l} Z \leftarrow p\,(X_1 \ldots X_k) \\ \text{if} \quad Z \neq 0 \text{ goto } M \end{array}$$

definieren; wir bezeichnen es mit if $\mathbf{p}\,(\mathbf{X_1} \ldots \mathbf{X_k})$ goto $\mathbf{M}$. Die Wirkung ist klar: trifft $p(\overline{a})$ zu, so ist der Funktionswert $p(\overline{a}) = 1$ und es wird zur Marke M gesprungen; andernfalls wird zur nächsten Programmzeile gegangen.
(c) Das Programm

$$\begin{array}{l} \text{if} \quad X \neq 0 \text{ goto } E \\ Y \leftarrow Y + 1 \end{array}$$

berechnet die Funktion **neg:** $I\!N \to \{0,1\}$ mit $\text{neg}(0) = 1$ und $\text{neg}(x) = 0$ für $x \neq 0$. Auf die Menge $\{0,1\}$ der Wahrheitswerte eingeschränkt, berechnet neg den negierten Wahrheitswert.
(d) if $\mathbf{V} = \mathbf{0}$ goto **M** sei folgendes Makro, das einen Sprung zu M bewirkt, falls V den Wert Null hat:

$$\begin{array}{l} Z \leftarrow \ \text{neg}\,(V) \\ \text{if} \quad Z \neq 0 \text{ goto } M\,. \end{array}$$

(e) Ausnutzung der bisher geschriebenen Makros zeigt, daß Prädikate wie $p(x_1, x_2) \iff x_1 \leq x_2$ [3], $q(x_1, x_2) \iff x_1 = x_2$, $r(x) \iff x_2$ ist Primzahl usw. RM-berechenbar sind.

Es ist jetzt eine Fleiß-bzw. Übungsaufgabe, weitere Funktionen und Prädikate als RM-berechenbar nachzuweisen. Die Behauptung vom Anfang des Abschnitts, jede intuitiv berechenbare Funkt sei RM-berechenbar, erhält damit bereits einige Plausibilität.

3. Rekursive Funktionen

Wir definieren in diesem Abschnitt die Klasse der rekursiven Funktionen und erhalten damit einen anderen, abstrakten Zugang zum Begriff der Berechenbarkeit. Es ist leicht einzusehen, daß jede rekursive Funktion intuitiv berechenbar, genauer sogar RM-berechenbar ist (Satz 3.7). Die Umkehrung gilt ebenfalls, ist aber schwieriger zu zeigen; ihr Beweis wird in Abschnitt 5 skizziert. Die Grundidee bei der Definition der rekursiven Funktionen ist, zuerst einige besonders einfache berechenbare Ausgangsfunktionen zu wählen (3.1) und diese dann unter Prozessen abzuschließen, die von berechenbaren Funktionen wieder zu berechenbaren führen (3.2 bis 3.4). Es soll hier bereits betont werden, daß die (in 3.5 endgültig definierten) rekursiven Funktionen i.a. nur partiell sind; dies ist wegen der angestrebten Äquivalenz von Rekursivität und RM- Berechenbarkeit auch nicht anders zu erwarten.

3.1 Ausgangsfunktionen Wir bezeichnen die folgenden Funktionen als Ausgangsfunktionen:

1. $n : I\!N \to I\!N$ mit $n(x) = 0$ (die Nullfunktion)
2. $s : I\!N \to I\!N$ mit $s(x) = x + 1$ (die Nachfolgerfunktion)
3. (für $1 \leq i \leq m$) $\quad pr_i^m : I\!N^m \to I\!N$ mit $pr_i^m(\overline{x}) = x_i$ (die Projektion auf die i-te Koordinate).

[3] $A \iff B$ bedeutet, daß die Aussagen A und B äquivalent sind.

Diese Funktionen sind jedenfalls RM-berechenbar: n wird durch das Programm $Y \leftarrow 0$ aus 2.4 berechnet, s durch das Programm

$$\begin{array}{l} Y \leftarrow X \\ Y \leftarrow Y+1 \end{array}$$

pr_i^m durch das Makro $Y \leftarrow X_i$ (siehe 2.5(c)).

Vereinbarung Wir werden es häufig mit partiellen Funktionen zu tun haben. Dabei ist folgende Schreibweise nützlich. Sind $t(\overline{x})$ und $t'(\overline{x})$ Terme für partielle Funktionen, die von dem Argumentetupel $\overline{x}$ abhängen, so bedeute die Schreibweise $t(\overline{x}) \simeq t'(\overline{x})$: $t(\overline{x})$ und $t'(\overline{x})$ sind beide undefiniert oder beide definiert und gleich. Ist $t(\overline{x})$ aus Teiltermen zusammengesetzt, etwa $t(\overline{x}) = f(t_1(\overline{x}), t_2(\overline{x}), \ldots)$, so ist $t(\overline{x})$ genau dann definiert, wenn $y_1 = t_1(\overline{x}), y_2 = t_2(\overline{x}), \ldots$ alle definiert sind und auch $f(y_1, y_2, \ldots)$ definiert ist.

3.2 Einsetzung Seien $f : I\!N^k \hookrightarrow I\!N$ und, für $1 \leq i \leq k$, $g_i : I\!N^n \hookrightarrow I\!N$ gegebene partielle Funktionen. Wir sagen, daß die Funktion
(1) $h : I\!N^n \hookrightarrow I\!N$ mit $h(x) \simeq f(g_1(x) \ldots g_k(x))$
durch *Einsetzung* aus f und den g_i entsteht.
Mit Hilfe der Projektionsfunktionen aus 3.1 kann man auch Einsetzungen, bei denen die g_i nicht alle dieselbe Stellenzahl haben oder die Variablen x_i permutiert sind, in der Normalform (1) schreiben. Sei z.B. für $\overline{x} = (x_1, x_2, x_3)$ die Funktion $t(\overline{x})$ definiert durch

$$t(\overline{x}) \simeq f(g(x_2), h(x_3, x_1), k(x_1, x_2, x_3))\,;$$

eine Normalform für t ist

$$t(\overline{x}) \simeq f(g(pr_2^3(\overline{x})),\ h(pr_3^3(\overline{x}),\ pr_1^3(\overline{x})),\ k(\overline{x}))\,.$$

3.3 Rekursion Seien $f : I\!N^k \hookrightarrow I\!N$ und $g : I\!N^{k+2} \hookrightarrow I\!N$ gegebene Funktionen. Die Funktion $h : I\!N^{k+1} \hookrightarrow I\!N$ mit

$$h(0, \overline{x}) \simeq f(\overline{x}), \quad h(t+1, \overline{x}) \simeq g(t, h(t, \overline{x}), \overline{x})$$

geht aus f und g durch *Rekursion* hervor. h ist also durch Induktion über t definiert; das Variablentupel $\overline{x}$ spielt die Rolle eines festen Parameters.
$h(t, \overline{x})$ ist genau dann definiert, wenn für alle Werte $s < t$ bereits $h(s, \overline{x})$ definiert ist und schließlich $g(t, h(t, \overline{x}), \overline{x})$ (im Sinne der Vereinbarung oben) definiert ist.

3.4 μ-Operator Sei $g : I\!N^{k+1} \hookrightarrow I\!N$ gegeben. Die Funktion $h : I\!N^k \hookrightarrow I\!N$ mit

$$h(\overline{x}) \simeq \text{ das kleinste } t \text{ mit } g(t, \overline{x}) \simeq 1$$

geht aus h durch Anwendung des *μ-Operators* hervor. Wir schreiben

$$h(\overline{x}) \simeq \mu t(g(t,\overline{x}) \simeq 1)\,.$$

$h(\overline{x})$ ist genau dann definiert und hat den Wert t, wenn $g(t,\overline{x})$ definiert und gleich 1, aber für alle $s < t$ $g(s,\overline{x})$ definiert und von 1 verschieden ist. Ist die Funktion g ein Prädikat $p : I\!N^{k+1} \to \{0,1\}$ (insbesondere total, und $p(t,\overline{x}) = 1$ genau dann, wenn $p(t,\overline{x})$ zutrifft), so schreiben wir auch

$$h(\overline{x}) \simeq \mu t\,(p(t,\overline{x}))\,.$$

Man beachte, daß die Ausgangsfunktionen aus 3.1 total sind und die Prozesse der Einsetzung bzw. Rekursion von totalen wieder zu totalen Funktionen führen. Jedoch ist eine Funktion h, die aus einem totalen g durch Anwendung des μ-Operators hervorgeht, i.a. nicht mehr total: $h(\overline{x})$ ist nur dann definiert, wenn ein t mit $g(t,\overline{x}) = 1$ existiert!

3.5 Rekursive Funktionen Die Klasse der *rekursiven Funktionen* (in der Literatur meist partiell rekursive Funktionen genannt) ist definiert als die kleinste Klasse von partiellen Funktionen (beliebiger Stellenzahl), die alle Ausgangsfunktionen enthält und unter den Prozessen der Einsetzung, Rekursion und Anwendung des μ-Operators abgeschlossen ist. Genauer, aber etwas umständlich: Eine Funktion f heißt genau dann rekursiv, wenn es eine endliche Folge $(f_1 \ldots f_n)$ gibt mit

(2) f_1 ist Ausgangsfunktion, und $f_n = f$

(3) für $2 \le i \le n$ ist f_i entweder Ausgangsfunktion oder geht aus einigen der Funktionen $f_1, \ldots, f_{i-1}$ durch Einsetzung oder Rekursion oder Anwendung des μ-Operators hervor.

Zum Beweis der Rekursivität einer Funktion f, die nicht Ausgangsfunktion ist, wird man also Funktionen angeben, die bereits als rekursiv nachgewiesen sind und aus denen f durch Einsetzen, Rekursion oder Anwenden des μ-Operators hervorgeht.

3.6 Beispiele von rekursiven Funktionen Wir geben einige Beispiele von rekursiven Funktionen an.
(a) Durch Induktion über $c \in I\!N$ zeigt man, daß die konstante Funktion $\mathbf{k_c} : I\!N^k \to I\!N$ mit $k_c(\overline{x}) = c$ rekursiv ist. Denn $k_0(\overline{x}) = n(pr_1^k(x))$ (n die Nullfunktion aus 3.1) ist rekursiv; mit k_c ist auch k_{c+1} rekursiv wegen $k_{c+1}(\overline{x}) = s(k_c(\overline{x}))$ (s die Nachfolgerfunktion aus 3.1).
(b) Die Additionsfunktion **add** : $I\!N^2 \to I\!N$ mit $\mathrm{add}(t,x) = t + x$ ist rekursiv, denn add läßt sich durch iteriertes Anwenden der Nachfolgerfunktion s definieren:

$$\mathrm{add}(0,x) = x = pr_1^1(x)\,,\ \mathrm{add}(t+1,x) = s(\mathrm{add}(t,x))\,.$$

Analog sind die Funktionen **mult** und **pot** von $\mathbb{N}^2$ nach $\mathbb{N}$ mit $\mathrm{mult}(t,x) = t \cdot x, \mathrm{pot}(t,x) = x^t$ rekursiv, da mult durch iteriertes Anwenden von add, pot durch iteriertes Anwenden von mult entsteht.
(c) Die Funktionen **vorg** (Vorgänger) von $\mathbb{N}$ nach $\mathbb{N}$ und **diff** (Differenz) von $\mathbb{N}^2$ nach $\mathbb{N}$ mit

$$\mathbf{vorg}(x) = \begin{cases} 0 & \text{für } x = 0 \\ x-1 & \text{für } x > 0 \end{cases} \qquad \mathbf{diff}(t,x) = \begin{cases} 0 & \text{für } x \leq t \\ x-t & \text{für } x > t \end{cases}$$

sowie **neg** (Negation) aus 2.7(c) sind rekursiv wegen $\mathrm{vorg}(0) = 0, \mathrm{vorg}(t{+}1) = t = pr_1^2(t, \mathrm{vorg}(t))$, $\mathrm{diff}(0,x) = x = pr_1^1(x)$, $\mathrm{diff}(t+1,x) = \mathrm{vorg}(\mathrm{diff}(t,x))$, $\mathrm{neg}(t) = \mathrm{diff}(t,1)$.
Für $\mathrm{diff}(t,x)$ schreiben wir kurz $x - t$.

3.7 Satz *Jede rekursive Funktion ist RM-berechenbar.*
Beweis. Sei f rekursiv; es gibt also eine Folge $(f_1 \ldots f_n)$ mit den in 3.5 genannten Eigenschaften. Wir zeigen durch Induktion über i, daß jedes f_i rekursiv ist; für $i = n$ ergibt sich dann die Rekursivität von $f = f_n$.
Ist die Funktion f_i Ausgangsfunktion, so ist sie nach 3.1 rekursiv; insbesondere ist f_1 rekursiv und der Induktionsanfang klar. Andernfalls geht f_i aus den Funktionen $f_i, \ldots, f_{i-1}$, die nach Induktionsannahme bereits rekursiv sind, durch einen der Prozesse Einsetzung, Rekursion, Anwendung des μ-Operators hervor. Man braucht sich daher nur klarzumachen, daß diese Prozesse von RM-berechenbaren Funktionen wieder zu RM-berechenbaren führen. Seien also g, h und $g_1, \ldots, g_k$ RM-berechenbar.
Entsteht h wie in 3.2 durch Einsetzung aus f und den g_i, so wird h durch folgendes Programm berechnet, das Makros für f und die g_i benutzt:

$$\begin{array}{l} Z_1 \leftarrow g_1(X_1 \ldots X_n) \\ \ldots \\ Z_k \leftarrow g_k(X_1 \ldots X_n) \\ Y \leftarrow f(Z_1 \ldots Z_k)\,. \end{array}$$

Sei die Funktion $h(t, x_1 \ldots x_k)$ wie in 3.3 durch Rekursion aus f und g definiert. Im folgenden Programm zur Berechnung von h benennen wir die Eingabevariablen mit $T, X_1, \ldots, X_k$ (statt, wie in Abschnitt 2, mit $X_1, \ldots, X_{k+1}$). Man beachte, daß die Hilfsvariable Z am Anfang der Rechnung den Wert 0 hat, nach r Rekursionsschritten den Wert r.

$$\begin{array}{ll} & Y \leftarrow f(X_1 \ldots X_k) \\ [\mathrm{A}\ ] & \text{if } T = 0 \text{ goto } E \\ & Y \leftarrow g(Z, Y, X_1 \ldots X_k) \\ & Z \leftarrow Z + 1 \\ & T \leftarrow T - 1 \\ & \text{goto } A\,. \end{array}$$

Geht h aus g wie in 3.4 durch Anwenden des μ-Operators hervor, so läßt sich h durch folgendes Programm berechnen; Y hat wieder am Anfang den Wert 0.

$$\begin{array}{ll} [\mathrm{A}\ ] & Z \leftarrow g(Y, X_1 \dots X_k) \\ & \text{if } Z = 1 \text{ goto } E \\ & Y \leftarrow Y + 1 \\ & \text{goto } A\,. \end{array}$$

□

3.8 Beispiele von rekursiven Prädikaten In 2.8.(b) hatten wir Prädikate definiert als totale Funktionen p von $\mathbb{N}^k$ in die Menge $\{0,1\} \subseteq \mathbb{N}$. Damit ist bereits festgelegt, was rekursive Prädikate sind: Ein Prädikat p ist rekursiv, wenn p (als Funktion von $\mathbb{N}^k$ in $\mathbb{N}$ angesehen) rekursiv ist.
(a) Sind p und q k-stellige rekursive Prädikate, so sind auch die (ebenfalls k-stelligen) aussagenlogischen Verbindungen $\neg, \vee, \wedge$ [4] von p und q rekursiv:

$$\neg p(\overline{x}) \;=\; \mathrm{neg}(p(\overline{x}))$$
$$(p \wedge q)(\overline{x}) = p(\overline{x}) \cdot q(\overline{x})\,, \quad (p \vee q)(\overline{x}) = \neg(\neg p(\overline{x}) \wedge \neg q(\overline{x}))\,.$$

(b) Das Gleichheitsprädikat p mit $p(x,y) \iff x = y$ und analog q und r mit $q(x,y) \iff x < y$, $r(x,y) \iff x \leq y$ sind rekursiv wegen

$$\begin{aligned} x = y &\iff \mathrm{neg}(y - x) = 1 \wedge \ \mathrm{neg}(x - y) = 1 \\ x < y &\iff \mathrm{neg}(y - x) = 0 \\ x \leq y &\iff x = y \vee x < y\,. \end{aligned}$$

Die Funktion **exp**, die, falls x eine Potenz von b ist, den Exponenten von x zur Basis b berechnet, ist rekursiv, denn das Gleichheitsprädikat ist rekursiv, und

$$\exp(x,b) \simeq \mu t(b^t = x).$$

(c) Mit $p(y,x)$ ist auch das durch beschränkte Existenzquantifikation aus p entstehende Prädikat

$$q(t,x) \iff \text{ es gibt ein } y \leq t \text{ mit } p(y,x) \quad (\text{Schreibweise: } (\exists y \leq t) p(y,x))$$ [5]

rekursiv, da sich q induktiv durch $q(0,x) \iff p(0,x)$, $q(t+1,x) \iff q(t,x) \vee p(t+1,x)$ definieren läßt.
(d) Das Teilbarkeits- und das Primzahlprädikat sind rekursiv:

$$x | y \iff (\exists t \leq y)(x \cdot t = y)$$
$$\mathrm{prim}(x) \iff x > 1 \wedge \neg(\exists t \leq x)(1 < t \wedge t < x \wedge t | x)\,.$$

[4] $\neg$ „non”, $\wedge$ „und”, $\vee$ „oder”
[5] $\exists y\, p(y)$: „es gibt ein y mit $p(y)$”

Wir folgern, daß die durch

$$\mathrm{pr}(0) = 1,\ \mathrm{pr}(t) = \text{ die } t - \text{te Primzahl, für } t \geq 1$$

definierte Funktion **pr** rekursiv ist, denn

$$pr(t+1) = \mu y(\mathrm{prim}(y) \wedge \mathrm{pr}(t) < y)\,.$$

(e) Seien $p_1, \ldots, p_n$ rekursive Prädikate, so daß für jedes k-tupel $\overline{x}$ genau eins der p_i zutrifft; $f_1, \ldots, f_n$ seien rekursive Funktionen. Dann ist die durch Fallunterscheidung definierte Funktion $f : \mathbb{N}^k \hookrightarrow \mathbb{N}$ mit

$$f(\overline{x}) \simeq \begin{cases} f_1(\overline{x}) & \text{falls } p_1(\overline{x}) \\ \ldots & \\ f_n(\overline{x}) & \text{falls } p_n(\overline{x}) \end{cases}$$

rekursiv, da $f(\overline{x}) \simeq f_1(\overline{x}) \cdot p_1(\overline{x}) + \ldots + f_n(\overline{x}) \cdot p_n(\overline{x})$.

3.9 Primitiv rekursive Funktionen; die Ackermann-Funktion Für viele Betrachtungen der Berechenbarkeitstheorie kommt man mit einer Teilklasse der Klasse aller rekursiven Funktionen aus, den *primitiv rekursiven* Funktionen. Wir nennen eine Funktion primitiv rekursiv, wenn sie sich aus den Ausgangsfunktionen durch Einsetzung und Rekursion, jedoch ohne Anwendung des μ-Operators, gewinnen läßt. Alle in 3.6 und 3.8 definierten Funktionen sind primitiv rekursiv; wir übergehen den Beweis.
Die primitiv rekursiven Funktionen haben die angenehme Eigenschaft, total zu sein, wie die Bemerkung vor 3.5 zeigt. Jedoch ist nicht jede totale rekursive Funktion primitiv rekursiv: die *Ackermann-Funktion* $A : \mathbb{N}^2 \to \mathbb{N}$ wird definiert durch

$$A(0,x) = x+1\,, \qquad A(i+1,0) = A(i,1)\,, \qquad A(i+1,x+1) = A(i, A(i+1,x))\,.$$

Zur Bestimmung von $A(i+1,x)$ werden endlich viele der Werte $A(i,y)$ bzw. $A(i+1,z)$ mit $z < x$ benutzt. A ist intuitiv berechenbar; mit etwas mehr Kenntnis rekursiver Funktionen läßt sich auch die Rekursivität von A zeigen.
Das interessante Merkmal der Ackermann-Funktion ist ihr exorbitant schnelles Wachstum: man sieht leicht, teilweise durch Induktion über x,

$$\begin{array}{l} A(1,x) = x+2\,, \qquad A(2,x) = 2x+3 \\ A(3,0) = 5\,, \qquad A(3,x+1) = 2 \cdot A(3,x) + 3 \\ A(4,0) = 13, \quad A(4,1) = 65533, \quad A(4,2) > 10^{16000}\,. \end{array}$$

Weniger einfach ist folgende Eigenschaft der Ackermann-Funktion zu zeigen:
Zu jeder primitiv rekursiven Funktion $f : \mathbb{N} \to \mathbb{N}$ gibt es ein (von f abhängiges) $c \in \mathbb{N}$ mit $f(x) < A(c,x)$ für alle x.
Daher ist A nicht primitiv rekursiv: andernfalls wäre auch die durch $f(x) = A(x,x)$ definierte Funktion primitiv rekursiv, und es gäbe c mit $f(x) < A(c,x)$.

Für $x = c$ ergibt sich $A(c, c) < A(c, c)$, ein Widerspruch.
Die Klasse der primitiv rekursiven Funktionen ist also für eine adäquate Beschreibung des Berechenbarkeitsbegriffs zu klein. Ihre Bedeutung liegt darin, daß primitiv rekursive Funktionen total sind und viele in den folgenden Abschnitten auftretende Funktionen sich als primitiv rekursiv herausstellen - siehe 5.4.

4. Codierungen

In diesem Abschnitt führen wir die Technik der Codierung ein. Sie ist grundlegend für das Ergebnis von Abschnitt 5: jede RM- berechenbare Funktion ist rekursiv.
Von einer Codierung sprechen wir in folgender Situation. Es sei B ein effektiv gegebener Bereich (s. Abschnitt 1) und

$$c : B \to \mathbb{N} .$$

Wir nennen c eine *Codierungsfunktion*, wenn folgendes zutrifft:

(1) c ist injektiv, und der Wertebereich W von c ist, als Teilmenge von $\mathbb{N}$, entscheidbar

(2) c und die Umkehrfunktion $c^{-1} : W \to B$ sind (intuitiv) berechenbar.

Wir stellen uns $b \in B$ durch $y = c(b) \in \mathbb{N}$ "codiert" vor, da sich b aus y durch $b = c^{-1}(y)$ auf effektive Weise zurückgewinnen läßt. Die Entscheidbarkeit von $W \subseteq \mathbb{N}$ kann man, nach den Vorarbeiten aus Abschnitt 3, sinnvoll verschärfen zu:
(1 a) das Prädikat "$y \in W$" ist rekursiv.
In vielen Fällen ist $W = \mathbb{N}$; dann ist c bijektiv und (1 a) trivialerweise erfüllt.
Ist eine bijektive Codierung $c : B \to \mathbb{N}$ fest gewählt, so kann man Begriffe von B auf $\mathbb{N}$ übertragen und umgekehrt. Einer zweistelligen Operation σ auf B entspricht z.B. die durch

$$\tau(y, z) = c(\sigma(c^{-1}(y), c^{-1}(z)))$$

definierte Operation auf $\mathbb{N}$; ist τ rekursiv, so ist σ jedenfalls intuitiv berechenbar (wegen $\sigma(a, b) = c^{-1}(\tau(c(a), c(b)))$). Auf diese Weise liefert der Begriff der Rekursivität auf $\mathbb{N}$ einen sinnvollen Berechenbarkeits- bzw. Entscheidbarkeitsbegriff auf B. Codierungen werden manchmal auch als Gödelisierungen bezeichnet nach dem österreichischen Mathematiker Kurt Gödel, der als erster die Codierungstechnik für Untersuchungen im Rahmen von Entscheidbarkeitsproblemen systematisch einsetzte.
Im Rest des Abschnitts werden wir vier effektiv gegebene Bereiche codieren, die für das Rechnen auf Registermaschinen relevant sind. Es soll noch einmal betont werden, daß die wesentlichen Eigenschaften unserer Codierungen Injektivität

und Berechenbarkeit sind, nicht jedoch die genaue formelmäßige Definition, wie sie unten ausgeführt ist; diese kann der Leser, der bereit ist, an die Existenz der gewünschten Codierungsfunktionen zu glauben, überschlagen. Ferner sind alle unten als rekursiv behaupteten Funktionen sogar primitiv rekursiv, was mit einiger Kenntnis primitiv rekursiver Funktionen leicht zu zeigen ist.

4.1 Codierung von Zahlenpaaren Die Menge $I\!N^2 = I\!N \times I\!N$ hat eine Codierungsfunktion, nämlich die Paarfunktion

$$\mathbf{pa} : I\!N^2 \to I\!N\,, \quad \mathrm{pa}(x,y) \;=\; 2^x \cdot (2y+1) - 1\,.$$

pa ist bijektiv, da jede natürliche Zahl $z \geq 1$ sich auf genau eine Weise als Produkt einer Zweierpotenz und einer ungeraden Zahl schreiben läßt; die Beispiele aus 3.6 zeigen, daß pa rekursiv ist. Man beachte, daß für alle x, y gilt:

$$x \leq \ \mathrm{pa}\ (x,y)\,, \quad y \leq \ \mathrm{pa}\ (x,y)\,.$$

Die beiden Funktionen

$\mathbf{l} : I\!N \to I\!N\,, \quad l(z) =$ die erste (linke) Komponente von $\mathrm{pa}^{-1}(z)$
$\mathbf{r} : I\!N \to I\!N\,, \quad r(z) =$ die zweite (rechte) Komponente von $\mathrm{pa}^{-1}(z)$

sind ebenfalls rekursiv nach 3.8 (c) und wegen

$$l(z) = \mu x(\exists y \leq z)\,(z = 2^x \cdot (2y+1)\ - 1)\,,$$
$$r(z) = \mu y(\exists x \leq z)\,(z = 2^x \cdot (2y+1)\ - 1)\,.$$

Insbesondere ist pa^{-1} intuitiv berechenbar, denn $\mathrm{pa}^{-1}(z)$ ist das Paar $(l(z), r(z))$.

4.2 Codierung von endlichen Folgen Wir kürzen die Menge aller endlichen Folgen ab mit

$$F = \{t : t = (a_1 \ldots a_n) \text{ ist endliche Folge mit } a_1, \ldots, a_n \in I\!N\}\,.$$

F wird codiert durch die Funktion $\mathbf{fz} : F \to I\!N$ mit

$$\mathrm{fz}(t) = \mathrm{pa}(n, \mathrm{pr}(1)^{a_n} \cdot \ \ldots \ \cdot \mathrm{pr}(n)^{a_n} - 1) \text{ für } t = (a_1 \ldots a_n)\,.$$

Dabei ist $\mathrm{pr}(i)$ die i-te Primzahl (siehe 3.8 (d)). $\mathrm{fz}(t)$ heißt die Folgenzahl von t. Sie codiert in der ersten Komponente die Länge n der Folge t, in der zweiten Komponente y die Folgenglieder a_i von t: a_i ist der Exponent von $\mathrm{pr}(i)$ in der Primzahlzerlegung von $y+1$.
Die Funktion fz ist bijektiv, da pa es ist und jede natürliche Zahl ≥ 1 sich auf genau eine Weise als Produkt von Primzahlpotenzen darstellen läßt. Für festes n ist $\mathrm{fz}(t)$, eingeschränkt auf die Menge aller Folgen der Länge n, rekursiv nach 3.8

(d). Auch die nächsten Funktionen von $I\!N$ bzw. $I\!N^2$ nach $I\!N$, die häufig benutzte Operationen bei der Handhabung von Folgen widerspiegeln, sind rekursiv:

$$\mathbf{le}(a) = l(a)\,, \quad \mathbf{co}(a,i) = \ \exp(r(a)+1, \mathrm{pr}(i))\,;$$

dabei ist l die linke , r die rechte Komponentenfunktion aus 4.1 und exp die Funktion aus 3.8 (b). Ihre Bedeutung ist folgende: Sei a die Folgenzahl von $t = (a_1 \ldots a_n) \in F$. Dann ist $le(a) = n$ die Länge n von t und, für $1 \leq i \leq n$, $\mathrm{co}(a,i) = a_i$ die i-te Komponente von t.

4.3 Codierung von Symbolen der Sprache RM Wir geben jetzt Funktionen **gnv** und **gnm** an , die die Symbole von RM (siehe 2.2) intuitiv berechenbar im Sinne von 1.2 nach $I\!N$ abbilden, genauer bildet gnv die Sonderzeichen und Variablen, gnm die Marken bijektiv ab. gnv(s) bzw. gnm(s) heißt die Gödelnummer des Symbols s. gnv bilde etwa die zehn Symbole “if”, “goto”, $\leftarrow, +, -, 0, 1, \neq, [,]$ auf die Zahlen von 0 bis 9 ab, die Variable Y auf 10, X_i auf $9+2i$ (für $i \geq 1$), Z_i auf 10+ $2i$. gnm bilde die Marke A_i (für $i \geq 1$, siehe 2.2) ab auf $i-1$.

4.4 Codierung von Programmen Die folgenden Funktionen, die wir der Einfachheit halber alle mit dem Symbol **gn** (für Gödelnummer) bezeichnen, codieren auf bijektive und intuitiv berechenbare Weise Befehle, Programmzeilen und Programme der Sprache RM.

b ist der Befehl $V \leftarrow V+1$:	gn(b) sei pa(0, gnv(V)–10)
b ist der Befehl $V \leftarrow V-1$:	gn(b) sei pa(1, gnv(V)–10)
b ist der Befehl if $V \neq 0$ goto M :	gn(b) sei pa(gnm(M)+2, gnv(V)–10).

Die Gödelnummer einer Programmzeile z, die aus einem Befehl b ohne vorangehende Marke besteht, sei pa(0, gn(b)). Hat z die Form “$[M]\,b$” (b Befehl, M Marke), so sei gn(z) = pa(gn(M)+1, gn(b)).
Die Gödelnummer eines Programms $P = (z_1, \ldots, z_n)$ sei die Folgenzahl fz (gn(z_1) ... gn(z_n)).
Diese Funktionen sind Bijektionen auf $I\!N$ und, samt ihren Umkehrfunktionen, intuitiv berechenbar. Daß z.B. die Menge aller Programmzeilen bijektiv auf $I\!N$ abgebildet wird, liegt u.a. daran, daß die Gödelnummern gnv von Variablen die Zahlen $10, 11, \ldots$, die Gödelnummern gnm von Marken die Zahlen $0, 1, \ldots$ sind.

5. Registermaschinen-berechenbare Funktionen sind rekursiv. Ein universelles Registermaschinen-Programm.

Die wesentliche Erkenntnis dieses Abschnitts und ein zentrales Ergebnis der Berechenbarkeitstheorie überhaupt ist die Aussage von Satz 5.2: die rekursiven Funktionen sind genau die RM-berechenbaren. 5.2 folgt leicht aus Satz 5.1, der seinerseits die für die Theorie der Berechenbarkeit fundamentalen Konsequenzen 5.3 und 5.4 hat. Wir wollen dies für k-stellige Funktionen aus $I\!N^k$ nach $I\!N$ zeigen und halten k im Rest der Betrachtungen fest. Zum Beweis betrachten wir die folgendermaßen definierte $k+1$-stellige partielle Funktion Φ: Für $e \in I\!N$ betrachte man das Programm P mit Gödelnummer e (siehe 4.4) und setze für jedes $x \in I\!N^k$

$$\Phi(e,\overline{x}) \simeq \varphi_P(\overline{x})\,;$$

φ_P wurde in 2.8 definiert als die durch P berechnete k-stellige Funktion. Wir nennen Φ die universelle $k+1$-stellige rekursive Funktion; Satz 5.1 und die Betrachtungen aus 5.3 rechtfertigen diese Benennung. Zur Bedeutung von Φ macht man sich folgendes klar.
Eine beliebige Funktion $f : I\!N^{k+1} \hookrightarrow I\!N$ liefert durch Festhalten von $e \in I\!N$ die k-stelligen Funktionen f_e mit $f_e(\overline{x}) \simeq f(e,\overline{x})$. Mit f sind auch alle f_e rekursiv, denn $f_e(\overline{x})$ entsteht aus $f(y,\overline{x})$ durch Einsetzen der konstanten Funktion k_e (siehe 3.6 (a)) für die Variable y. Ist nun $f = \Phi$ und P das Programm mit Gödelnummer e, so ist $f_e = \varphi_P$. Satz 5.1 besagt, daß nicht nur jedes φ_P, sondern darüber hinaus sogar Φ rekursiv ist. Der Beweis ist in 5.5 skizziert.

5.1 Satz *Es gibt ein $k+2$-stelliges rekursives Prädikat T und eine einstellige totale rekursive Funktion U, so daß für alle $\overline{x} \in I\!N^k$*

$$\Phi(e,\overline{x}) \simeq U(\mu t T(t,e,\overline{x}))\,.$$

Daher ist Φ rekursiv.

5.2 Satz *Für jede Funktion $f : I\!N^k \hookrightarrow I\!N$ sind äquivalent:*

(a) f ist rekursiv

(b) f ist RM-berechenbar

(c) es gibt ein $e \in I\!N$, so daß für alle $\overline{x}$
(1) $\quad f(\overline{x}) \simeq \Phi(e,\overline{x}) \simeq U(\mu t T(t,e,\overline{x}))\,.$

Die Äquivalenz von (a) und (c) heißt der *Kleenesche Normalformensatz* nach dem amerikanischen Mathematiker Stephen C. Kleene; die Formel (1) liefert die sog. Kleenesche Normalform für f.
Beweis. (b) folgt aus (a) nach Satz 3.7. (a) folgt aus (c): nach 5.1 ist Φ rekursiv, und f entsteht aus Φ durch Festhalten der ersten Stelle. (c) folgt aus (b): f werde durch ein Programm P berechnet; man setze $e = \mathrm{gn}(P)$. Die Definition von Φ und Satz 5.1 zeigen,

daß für jedes $\overline{x}$

$$f(\overline{x}) \simeq \varphi_P(\overline{x}) \simeq \Phi(e, \overline{x}) \simeq U(\mu t\, T(t, e, \overline{x}))\,.$$

$\square$

5.3 Kleenes Aufzählungssatz Sei $f : I\!N^{k+1} \hookrightarrow I\!N$ eine gegebene Funktion; für jedes $e \in I\!N$ erhält man aus f die k-stellige Funktion f_e mit $f_e(\overline{x}) \simeq f(e, \overline{x})$. Sei $\mathbf{K}$ eine Menge von k-stelligen Funktionen. Wir sagen, f zähle $\mathbf{K}$ auf, wenn $\mathbf{K} = \{f_e : e \in I\!N\}$.
Die Äquivalenz von (a) und (c) in 5.2 liefert *Kleenes Aufzählungssatz.* Er besagt, daß Φ (eine $k+1$-stellige rekursive partielle Funktion) die Menge aller k-stelligen rekursiven (partiellen) Funktionen aufzählt.

5.4 Ein universelles Programm Die universelle Funktion Φ ist $k+1$-stellig, rekursiv und liefert bei Festhalten der ersten Stelle e alle rekursiven k-stelligen Funktionen. Nach Satz 5.2 für $k+1$ statt k sind die rekursiven Funktionen gerade die RM-berechenbaren. Wir erhalten so die bemerkenswerte Tatsache, daß Φ selbst RM-berechenbar ist und bei Festhalten der ersten Stelle alle RM-berechenbaren k-stelligen Funktionen liefert.
Sei etwa P_U ein RM-Programm, das Φ berechnet. P_U *simuliert* die Berechnung aller k-stelligen RM-berechenbaren Funktionen in folgendem Sinne: ist f eine solche Funktion, so gibt es ein Programm P, das f berechnet , d.h. mit $f = \varphi_P$. Sei e die Gödelnummer von P. Nach Eingabe von e und $\overline{x}$ berechnet P_U den Wert $\Phi(e, \overline{x}) \simeq \varphi_P(\overline{x}) \simeq f(\overline{x})$. Wir nennen P_U ein *universelles Programm* für die Klasse der k-stelligen RM-berechenbaren Funktionen.

5.5 Beweisskizze zu Satz 5.1 Wir definieren eine Hilfsfunktion

$$\mathrm{cont} : I\!N^{k+2} \to I\!N$$

folgendermaßen. Seien $t, e \in I\!N$ und $\overline{x} = (x_1 \ldots x_k) \in I\!N^k$ gegeben. P sei das Programm mit Gödelnummer e. Für jede Variable V (in der Sprache RM) sei

$$\begin{aligned} \mathrm{cont}_V(t, e, \overline{x}) &= \text{der Wert von } V \text{ nach } t \text{ Rechenschritten bei der Berechnung} \\ &\quad\ \text{von } \varphi_P(\overline{x}) \\ \mathrm{count}(t, e, \overline{x}) &= \text{der Wert des Programmzählers nach } t \text{ Rechenschritten bei} \\ &\quad\ \text{der Berechnung von } \varphi_P(\overline{x}), \end{aligned}$$

d.h. bei der Berechnung gemäß P mit dem Eingabetupel $(x_1 \ldots x_k)$. Dann setzen wir

$$\mathrm{cont}(t, e, \overline{x}) = 2^{\mathrm{count}(t,e,\overline{x})} \cdot \prod_V \mathrm{pr}(\mathrm{gn}(V) + 1)^{\mathrm{cont}_V(t,e,\overline{x})}\,,$$

das Produkt läuft über alle Variablen von RM. Dies formal unendliche Produkt ist wohldefiniert, da bei einer Berechnung gemäß P außer $X_1, \ldots, X_k$ nur solche

Variablen von 0 verschiedene Werte annehmen, die in P wirklich vorkommen; fast alle $\mathrm{cont}_V(t,e,\overline{x})$ sind also 0. $\mathrm{cont}(t,e,\overline{x})$ codiert den Inhalt sämtlicher Speicherplätze nach t Rechenschritten bei der Berechnung gemäß P mit dem Eingabetupel $\overline{x}$.

Die entscheidende Tatsache des Beweises ist nun, daß die Funktion cont rekursiv ist. Der Nachweis ist mit den Hilfsmitteln aus Abschnitt 3 und 4 nicht schwierig, aber umfangreich bzw. schreibtechnisch aufwendig. Er kann beim ersten Lesen evtl. überschlagen werden, zumal die intuitive Berechenbarkeit von cont klar sein dürfte. Wir geben im folgenden eine Skizze.

$\mathrm{cont}(t,e,\overline{x})$ ist durch Rekursion über t definiert. Und zwar ist

$$\mathrm{cont}(0,e,x) = 2^1 \cdot \mathrm{pr}((gn(X_1)+1)^{x_1} \cdot \ldots \cdot \mathrm{pr}(gn(X_k)+1)^{x_k}$$

eine rekursive Funktion, die von e und $\overline{x}$ abhängt. $\mathrm{cont}(t+1,e,\overline{x})$ geht aus $\mathrm{cont}(t,e,\overline{x})$ hervor durch Anwenden einer Funktion h:

$$\mathrm{cont}(t+1,e,\overline{x}) \;=\; h(e,\mathrm{cont}(t,e,\overline{x}))\,;$$

wir deuten an, wie die Funktion h definiert ist und warum sie rekursiv ist. Zur Berechnung von $h(e,\mathrm{cont}(t,e,\overline{x}))$ ermittelt man durch Decodieren von $\mathrm{cont}(t,e,\overline{x})$ den Wert $i = \mathrm{count}(t,e,\overline{x}) = \exp(2,\mathrm{cont}(t,e,\overline{x}))$ des Programmzählers zur Zeit t; durch Decodieren von e die i-te Programmzeile z_i von e: Es ist einfach $\mathrm{gn}(z_i) = \exp(e,\mathrm{pr}((i))$, durch Decodieren von $\mathrm{gn}(z_i)$ den als nächsten auszuführenden Befehl b und die für b relevante Variable V sowie evtl. die in z_i auftretende Marke M. Dann bildet man den neuen Wert von V (ersetzt also in $\mathrm{cont}(t,e,\overline{x})$ z.B., falls b die Form $V \leftarrow V+1$ und die Gödelnummer j hat, den Faktor $\mathrm{pr}(j)^{\mathrm{cont}_V(t,e,\overline{x})}$ durch $\mathrm{pr}(j)^{\mathrm{cont}_V(t,e,\overline{x})+1}$) und den neuen Wert des Programmzählers (ersetzt also in $\mathrm{cont}(t,e,\overline{x})$ 2^i durch 2^r, falls z_r die als nächste zu bearbeitende Programmzeile ist). Ein genauerer Nachweis der Rekursivität von h benutzt i.w. die Codierungen aus Abschnitt 4 und, wegen der Fallunterscheidungen in 2.3, die Möglichkeit der Definition rekursiver Funktionen durch rekursive Fallunterscheidung (siehe 3.8 (e)).

Das Prädikat T sei nun definiert durch

$$T(t,e,\overline{x}) \iff \text{zum Zeitpunkt } t \text{ ist die Berechnung gemäß } P \text{ mit dem Eingabetupel } \overline{x} \text{ beendet.}$$

T ist rekursiv wegen $T(t,e,\overline{x}) \iff \exp(\mathrm{cont}(t,e,\overline{x}),2) > \mathrm{le}(e)$, und da cont und exp rekursiv sind. Man beachte dabei, daß $\mathrm{le}(e)$ die Länge n von P ist und $\exp(\mathrm{cont}(t,e,\overline{x}),2)$ der Wert des Programmzählers zum Zeitpunkt t. $U : \mathbb{N} \to \mathbb{N}$ sei die rekursive Funktion mit

$$U(z) \;=\; \exp(z,\mathrm{pr}(11))\,;$$

ist $z = \mathrm{cont}(t,e,\overline{x})$, so ist $U(z)$ der Wert der Ausgabevariablen Y zum Zeitpunkt t, da Y die Gödelnummer 10 hat. Nach dieser Definition von T und U gilt, wie gewünscht,

$$\Phi(e, \overline{x}) \simeq U(\mu t T(t, e, \overline{x}))\,.$$

Genauere Betrachtungen zeigen übrigens , daß die Funktionen h und cont, somit auch T und U primitiv rekursiv sind. Dies liefert, zusammen mit (1) in 5.2, die interessante Tatsache, daß jede rekursive Funktion eine Darstellung hat, die nur einmal den μ-Operator benutzt. □

6. Die Churchsche These

Die Ergebnisse des vorangehenden Abschnitts liefern Argumente für eine fundiertere Diskussion der Churchschen These, die am Anfang des Kapitels ausgesprochen wurde.

6.1 Churchsche These für Funktionen Eine Funktion ist genau dann (intuitiv) berechenbar, wenn sie rekursiv ist.

Eine Relation R ist genau dann intuitiv entscheidbar, wenn ihre charakteristische Funktion χ_R, ein Prädikat im Sinne von 2.8, intuitiv berechenbar ist. Als Folgerung von 6.1 ergibt sich:

6.2 Churchsche These für Relationen Eine Relation R ist genau dann (intuitiv) entscheidbar, wenn das zugehörige Prädikat χ_R rekursiv ist.

Die Churchsche These ist nicht als ein mathematischer Satz anzusprechen, da sie das Zusammenfallen eines anschaulichen Begriffs (intuitive Berechenbarkeit) mit einem mathematischen (Rekursivität) behauptet; aus diesem Grund kann sie auch nicht in einem strengen mathematischen Sinne beweisbar sein. Es handelt sich hier um ein erkenntnistheoretisches Prinzip, das man grundsätzlich akzeptieren oder verwerfen kann. Seit ihrer Formulierung 1934 durch den amerikanischen Mathematiker Alonzo Church und genaueren Begründung 1936 durch Turing (siehe 9.1) gilt sie als allgemein akzeptiert. Folgende Tatsachen sprechen für ihre Richtigkeit.
(1) Jede rekursive Funktion ist intuitiv berechenbar - z.B. nach Satz 3.7 RM-berechenbar.
(2) Bisher wurde keine intuitiv berechenbare, aber nicht-rekursive Funktion gefunden. Techniken aus den vorangehenden Abschnitten, insbesondere die der Codierung, machen es meist zu einer Routineangelegenheit, die Rekursivität konkret gegebener berechenbarer Funktionen nachzuweisen.
(3) Verschiedene Versuche, den Begriff der Berechenbarkeit zu präzisieren, erwiesen sich als äquivalent zu dem der Rekursivität. Solche Begriffe waren u.a., außer dem hier bereits behandelten der RM-Berechenbarkeit: Herbrand-Gödel-Berechenbarkeit (Gödel, Kleene 1934 - 1936), Darstellbarkeit in formalen Systemen der Arithmetik (Gödel, Church 1936), Turing-Berechenbarkeit (Turing

1936; siehe Abschnitt 9), λ-Definierbarkeit (Church, Kleene, Rosser 1933 - 1936), Flußdiagramm-Berechenbarkeit (Goldstine, von Neumann, Wang, Peter 1947-1959).
Das überzeugendste Argument für Churchs These scheint jedoch nicht so sehr das bloße Bestehen dieser Äquivalenzen, als vielmehr die Uniformität der Äquivalenzbeweise zu sein. Alle verlaufen nach folgendem Schema, das sich in Abschnitt 3 bis 5 oben abzeichnet. Man hat eine Klasse **K** von partiellen Funktionen und vermutet, daß sie mit der Klasse aller rekursiven Funktionen übereinstimmt. Ist z.B. **K** definiert als der Abschluß einiger Ausgangsfunktionen unter gewissen Erzeugungsprozessen, so reicht es zu zeigen:
(i) die Ausgangsfunktionen aus 3.1 sind in **K**, und die Erzeugungsprozesse aus 3.2 bis 3.4 führen nicht aus **K** heraus - damit gehören alle rekursiven Funktionen zu **K**;
(ii) die Ausgangsfunktionen von **K** sind rekursiv, und die Erzeugungsprozesse von **K** führen nicht aus der Klasse der rekursiven Funktionen heraus - damit sind alle Funktionen aus **K** rekursiv.
K könnte auch die Klasse aller in einem gegebenen System **A** von Algorithmen berechenbaren Funktionen sein; dann zeigt man:
(iii) die Ausgangsfunktionen aus 3.1 sind in **A** berechenbar, und die Erzeugungsprozesse aus 3.2 bis 3.4 führen nicht aus der Klasse der **A**-berechenbaren Funktionen heraus - damit sind alle rekursiven Funktionen **A**-berechenbar;
(iv) jede **A**-berechenbare Funktion ist rekursiv - hierzu benutzt man, in Nachahmung der Beweisskizze 5.4 , die Codierungsmethoden aus Abschnitt 4.

Die Churchsche These wird gern als Arbeitshypothese angewandt, wenn man sich ohne großen technischen Aufwand die Rekursivität einer gegebenen Funktion klarmachen will; es ist i.a. eine reine Fleißaufgabe, in solchen Anwendungen Churchs These zu eliminieren, d.h. einen strengen Beweis für die Rekursivität zu liefern. Unentbehrlich ist sie jedoch beim Nachweis der Nicht-Berechenbarkeit einer gegebenen Funktion f: in diesem Fall beweist man die Nicht-Rekursivität von f und braucht Churchs These, um auf die Nicht- Berechenbarkeit zu schließen. Entsprechend muß auf Churchs These zurückgegriffen werden, um die Unentscheidbarkeit von Relationen zu zeigen.

7. Das Halteproblem

Churchs These erlaubt es, die Unentscheidbarkeit von Problemen zu zeigen. Wir tun dies für das sogenannte Halteproblem, das in Betrachtungen der Berechenbarkeitstheorie eine zentrale Rolle spielt. Z.B. wird sich in Abschnitt 8 herausstellen, daß viele weitere unentscheidbare Probleme eng mit dem Halteproblem zusammenhängen. Das Halteproblem beruht auf der Tatsache, daß die universelle rekursive Funktion Φ aus Abschnitt 5 nicht total ist.

Nach Kleenes Satz 5.3. zählt die $k+1$-stellige, rekursive, partielle Funktion Φ die Menge aller k-stelligen rekursiven partiellen Funktionen auf. Gibt es eine $k+1$-stellige, rekursive, *totale* Funktion, die die Menge aller k-stelligen rekursiven totalen Funktionen aufzählt?
Ein typisches Diagonalargument, die konstruktive Fassung von Cantors Diagonalverfahren aus Abschnitt 1, gibt eine negative Antwort. Der Einfachheit halber sei dabei $k=1$. Wäre f eine solche aufzählende Funktion, so definiert man eine einstellige Funktion durch

$$d(x) = f_x(x) + 1\,.$$

Mit f ist auch d rekursiv und total; da f gerade diese Funktionen aufzählt , ist $d = f_e$ für ein $e \in \mathbb{N}$. Für $x = e$ ergibt sich der Widerspruch $f_e(e) = d(e) = f_e(e) + 1$.
Die Tatsache, daß rekursive Funktionen und insbesondere die universelle rekursive Funktion Φ i.a. nur partiell definiert sind, ist also der Preis für solch starke Ergebnisse wie Kleenes Aufzählungssatz und Satz 5.2. Und die in 7.2 bewiesene Unlösbarkeit des Halteproblems besagt, daß das nur partielle Definiertsein von Φ durchaus problematische Aspekte hat.

Im Rest des Abschnitts benutzen wir die Notation aus Abschnitt 5. Die Stellenzahl k sei dabei 1.

7.1 Die Halte-Relation Wir definieren Mengen $H \subseteq \mathbb{N}^2$ und $K \subseteq \mathbb{N}$ durch

$$H = \{(y,x) \in \mathbb{N}^2 : \Phi\,(y,x) \text{ ist definiert}\}, \quad K = \{x \in \mathbb{N} : (x,x) \in H\}\,.$$

Wir nennen H die *Halte-Relation*, denn:

$(y,x) \in H \iff \Phi(y,x)$ ist definiert
$\iff$ das universelle Programm P_U aus 5.4, das Φ berechnet, hält bei Eingabe von (y,x) nach endlich vielen Schritten
$\iff$ die Berechnung gemäß P (P das Programm mit $gn(P) = y$) hält bei Eingabe von x nach endlich vielen Schritten.

Das Problem, zu entscheiden, ob ein gegebenes Paar (y,x) zu H gehört, heißt das *Halteproblem*.

7.2 Unlösbarkeit des Halteproblems Nach Churchs These in der Form 6.2 bedeutet die Unlösbarkeit (d.h. Unentscheidbarkeit) des Halteproblems, daß die charakteristische Funktion χ_H von H nicht rekursiv ist.
Wäre nämlich χ_H rekursiv, so auch χ_K für die in 7.1 definierte Menge K, denn es ist $\chi_K(x) = \chi_H(x,x)$. Wir nehmen an, χ_K sei rekursiv, und leiten einen Widerspruch ab. Mit χ_K und Φ und wegen 3.8(e) ist auch folgende Funktion

$g : I\!N \hookrightarrow I\!N$ rekursiv:

$$g(x) \simeq \begin{cases} \Phi(x,x)+1 & \text{falls } x \in K \text{ (d.h. falls } \chi_K(x)=1) \\ 0 & \text{falls } x \notin K \text{ (d.h. falls } 1-\chi_K(x)=1)\,. \end{cases}$$

g ist total, denn für $x \in K$ ist $\Phi(x,x)$ definiert. Da g rekursiv ist, gibt es (siehe 5.2) ein e mit

(2) $g(x) = \Phi(e,x)$ für alle x.

Man betrachte den Wert $x = e$. Ist $g(e) = 0$, so folgt $e \notin K$ nach Definition von g, $(e,e) \notin H$ und $\Phi(e,e)$ ist undefiniert; ein Widerspruch zu (2), da g total ist. Ist aber $g(e) \neq 0$, so tritt in der Definition von $g(e)$ der erste Fall ein, und es ist $\Phi(e,e) = g(e) = \Phi(e,e)+1$, ein Widerspruch.

Ausgehend von der unentscheidbaren Menge $K \subseteq I\!N$ kann man durch Folgenzahl-Codierung leicht für beliebiges $k \geq 1$ eine k-stellige unentscheidbare Relation R angeben: man setze z.B.

$$R = \{t = (x_1 \ldots x_k) \in I\!N^k : \text{fz}(t) \in K\}\,;$$

$\text{fz}(t)$ ist dabei die Folgenzahl von t nach 4.2.

8. Rekursiv aufzählbare Mengen

Die in diesem Abschnitt behandelten rekursiv aufzählbaren Mengen sind bei Entscheidbarkeitsproblemen von Interesse: sie sind i. a. unentscheidbar, und viele unentscheidbare Mengen stellen sich umgekehrt als rekursiv aufzählbar heraus. Die Definition rekursiv aufzählbarer Mengen in 8.1 beruht auf der Tatsache, daß rekursive Funktionen i.a. nicht total sind. Satz 8.3 und der ihm folgende Kommentar geben eine Erklärung für die Bezeichnung "rekursiv aufzählbar" und eine anschauliche Interpretation des Begriffs.

8.1 Rekursive und rekursiv aufzählbare Mengen Ist $f : I\!N \hookrightarrow I\!N$ eine partielle Funktion, so schreiben wir $D(f)$ für den Definitionsbereich, $W(f)$ für den Wertebereich von f. Wir nennen eine Menge $A \subseteq I\!N$ *rekursiv*, falls das zugehörige Prädikat χ_A rekursiv ist; nach Churchs These in der Form 6.2 sind die rekursiven Mengen genau die entscheidbaren.
A heißt *rekursiv aufzählbar*, wenn es ein rekursives $f : I\!N \hookrightarrow I\!N$ mit $A = D(f)$ gibt.

8.2 Rekursive Mengen sind rekursiv aufzählbar Wir zeigen, daß jede rekursive Menge rekursiv aufzählbar ist: sei $A \subseteq I\!N$ rekursiv und damit das Prädikat

$p = \chi_A : \mathbb{N} \to \mathbb{N}$ rekursiv, also RM-berechenbar. Mit einem Makro für p schreiben wir das Programm P:

$$[M] \quad \text{if} \quad \neg p(x) \text{ goto } M\,.$$

Die von P berechnete Funktion hat gerade den Definitionsbereich A. Die Umkehrung gilt jedoch nicht. Z.B. ist die Menge K aus 7.1 rekursiv aufzählbar, denn die Funktion $f : \mathbb{N} \hookrightarrow \mathbb{N}$ mit $f(x) \simeq \Phi(x,x)$ ist rekursiv und hat den Definitionsbereich K; der Beweis von 7.2 zeigt, daß K nicht rekursiv ist. Zum Zusammenhang zwischen Rekursivität und rekursiver Aufzählbarkeit siehe auch 8.4.

Die leere Menge ist offenbar rekursiv, also rekursiv aufzählbar. Daher reicht es im folgenden Satz zu charakterisieren, welche nichtleeren Mengen rekursiv aufzählbar sind. Wir lassen den Beweis aus, obwohl er grundsätzlich nicht schwer ist.

8.3 Satz (Äquivalenzen zur rekursiven Aufzählbarkeit) *Für nichtleeres $A \subseteq \mathbb{N}$ sind äquivalent:*

(a) A ist rekursiv aufzählbar, d.h. $A = D(f)$ für ein rekursives f: $\mathbb{N} \hookrightarrow \mathbb{N}$

(b) es gibt ein rekursives Prädikat $p : \mathbb{N}^2 \to \{0,1\}$, so daß für alle $x \in \mathbb{N}$

$$x \in A \Longleftrightarrow \exists t p(t,x)$$

(A entsteht durch Existenzquantifikation aus p)

(c) es gibt ein rekursives $g : \mathbb{N} \hookrightarrow \mathbb{N}$ mit $A = W(g)$ (siehe 8.1); g kann sogar total gewählt werden.

Äquivalenz (c) des Satzes gibt eine intuitive Vorstellung von rekursiv aufzählbaren Mengen und erklärt die Namensgebung: ist $g : \mathbb{N} \to \mathbb{N}$ total und rekursiv mit $A = W(g)$, so wird A von g in der Form $A = \{g(0), g(1), g(2), \ldots\}$ aufgezählt. Man könnte einen Rechner programmieren, der der Reihe nach die Werte $a_n = g(n)$ berechnet und in Form einer Liste ausdruckt; die Elemente von A sind genau diejenigen, die auf der Liste $a_0, a_1, a_2, \ldots$ aufgezählt werden.

$$\begin{array}{c} a_0 \\ a_1 \\ a_2 \\ \ldots \end{array}$$

8.4 Zusammenhang zwischen Rekursivität und rekursiver Aufzählbarkeit Die Definition der rekursiven Aufzählbarkeit in 8.1 und ihre Charakterisierung in 8.3 greifen auf den Begriff der Rekursivität zurück. Umgekehrt läßt sich aber auch Rekursivität durch rekursive Aufzählbarkeit ausdrücken:

Satz *(a) Eine Teilmenge A von $I\!N$ ist genau dann rekursiv, wenn A und das Komplement $I\!N \setminus A$ beide rekursiv aufzählbar sind.*
(b) Eine Funktion $f : I\!N \hookrightarrow I\!N$ ist genau dann rekursiv, wenn die Menge

$$G(f) = \{pa(x,y) \,:\, x \in D(f) \text{ und } y = f(x)\},$$

die (den Graphen von) f codiert, rekursiv aufzählbar ist.
Beweis. (a) Ist A (also das zu A gehörige Prädikat χ_A) rekursiv, so auch $I\!N \setminus A$ wegen $\chi_{I\!N\setminus A} = \neg\chi_A$; nach 8.2 sind A und $I\!N \setminus A$ rekursiv aufzählbar. Sind umgekehrt A und $I\!N \setminus A$ beide rekursiv aufzählbar, etwa nach 8.3 (b)

$$x \in A \iff \exists\, t\, p(t,x), \qquad x \in I\!N \setminus A \iff \exists t\, q(t,x)$$

mit rekursiven Prädikaten p und q, so ist A rekursiv wegen

$$x \in A \iff p(\mu t\,(p(t,x) \vee q(t,x)))\,.$$

Auf den Beweis von (b) soll hier verzichtet werden. □

8.5 Unentscheidbarkeit und Reduzierbarkeit Rekursiv aufzählbare Mengen treten häufig im Zusammenhang mit unentscheidbaren Problemen auf. Z.B. ist die in 1.3 definierte unentscheidbare Teilmenge L von Pol rekursiv aufzählbar (bei sinngemäßer Übertragung des Begriffs "rekursiv aufzählbar" auf den effektiv gegebenen Bereich Pol). Für ein Polynom $p(x_1,\ldots,x_k)$ gilt nämlich

$$p \in L \iff \exists \overline{a} \in Z\!\!Z^k(p(\overline{a}) = 0)\,;$$

d.h. L entsteht durch Existenzquantifikation aus dem Prädikat Q mit $Q(p,\overline{a}) \iff p(\overline{a}) = 0$. Q ist intuitiv berechenbar; nach 8.3 und Churchs These ist L rekursiv aufzählbar.
Entscheidbarkeit bzw. Unentscheidbarkeit von Mengen $A, B \subseteq I\!N$ lassen sich oft durch Betrachtungen folgender Art zeigen. Sei $s : I\!N \to I\!N$ rekursiv und total, und es gelte für alle x

$$x \in A \iff s(x) \in B\,.$$

Ist B entscheidbar, so auch A: um zu entscheiden, ob $x \in A$, berechnet man $s(x)$ und prüft, ob $s(x) \in B$. Die Funktion s *reduziert* also das Entscheidungsproblem für A auf das für B. Die Reduktionsmethode läßt sich auch in entgegengesetzter Richtung anwenden: ist A bereits als unentscheidbar bekannt (etwa $A = K$ aus 7.1), so folgt die Unentscheidbarkeit von B.
Die aus $H \subseteq I\!N^2$ (siehe 7.1) durch Paarcodierung entstehende Menge

$$H^* = \{pa(y,x) \,:\, (y,x) \in H\} \subseteq I\!N$$

ist rekursiv aufzählbar (denn: $z \in H^* \Longleftrightarrow \Phi$ ist an der Stelle $(l(z), r(z))$ definiert). Sie hat die interessante Eigenschaft, daß jede rekursiv aufzählbare Menge auf H^* reduzierbar ist. Sei nämlich A rekursiv aufzählbar, etwa $A = D(f)$ mit $f(x) \simeq \Phi(e, x)$ nach 5.2. Die Funktion s mit $s(x) = \mathrm{pa}(e, x)$ ist rekursiv, total und reduziert A auf H^* wegen

$$\begin{aligned} x \in A &\Longleftrightarrow \Phi(e,x) \text{ ist definiert} \\ &\Longleftrightarrow (e,x) \in H \\ &\Longleftrightarrow \mathrm{pa}(e,x) \in H^* \\ &\Longleftrightarrow s(x) \in H^* . \end{aligned}$$

In diesem Sinne ist H^* die "komplizierteste" rekursiv aufzählbare Menge. Es läßt sich zeigen, daß die Menge K aus 7.1 dieselbe Eigenschaft hat, d.h. K ist rekursiv aufzählbar, und jede rekursiv aufzählbare Menge ist auf K reduzierbar.

9. Turingmaschinen

9.1 Motivation 1936 gab der englische Mathematiker Alan Turing ein Modell der Berechenbarkeit an, das wir heute als Turingmaschine bezeichnen. Turingmaschinen sind äußerst primitive Rechenautomaten und die auf ihnen durchgeführten Berechnungen daher technisch sehr mühsam - siehe das Beispiel in 9.4. Für die theoretische Informatik sind sie von Interesse, weil sie das historisch früheste mechanische Berechenbarkeitsmodell darstellen und Turing ihre Definition motivierte durch Vergleich mit der Weise, wie ein Mathematiker traditionellerweise per Hand auf Papier rechnet. Turings Motivation war etwa die folgende.

1. Die Darstellung von Daten oder allgemeiner von mathematischem Text benutzt nur endlich viele Symbole, etwa die lateinischen (falls nötig, auch griechische, deutsche, ...) Buchstaben, die Ziffern (etwa $0, 1, \ldots, 9$ im Dezimalsystem oder $0, 1$ im Dualsystem) sowie endlich viele mathematische Sonderzeichen, insgesamt ein endliches Alphabet $A = \{s_1, \ldots, s_n\}$, wobei i.a. $n \geq 2$ ist; zusätzlich ist noch das "Leer"-Symbol $\star$ (auf englisch "blank") erlaubt. - Dies ist offensichtlich keine Beschränkung der Allgemeinheit, da man weitere evtl. benötigte Symbole durch endliche Zeichenreihen (Worte) des benutzten Alphabets codieren kann.

2. Die Symbole werden in Kästchen bzw. "Felder" geschrieben, wie beim Rechnen in der Grundschule auf Karopapier, in jedes Feld höchstens ein Symbol. - Auch in einem real existierenden Computer kann ein Speicherplatz nur Bit-Folgen fester endlicher Länge speichern, und diese können durch endlich viele Symbole codiert werden.

3. Die Felder sind in Form eines sogenannten Turingbandes nebeneinander angeordnet:

...	$\star$	1	0	1	$\star$	...

das Band ist in beiden Richtungen (potentiell) unendlich lang. - Ein Blatt Karopapier ist rechteckig in $n \cdot m$ Kästchen eingeteilt; diese könnte man in $n \cdot m$ nebeneinander liegende Felder übertragen.

4. Zu jedem Zeitpunkt der Rechnung wird genau ein Feld des Bandes gelesen bzw. beobachtet - etwa durch den "Lesekopf" einer Maschine, unter dem das Band hin- und herbewegt wird. Das jeweils beobachtete Feld ist das "Arbeitsfeld"; wir stellen es graphisch durch einen daruntergezeichneten Pfeil dar. - Ein auf Karopapier rechnender Mathematiker kann nur eine beschränkte Anzahl von Feldern gleichzeitig lesen und bearbeiten; diese kann man, bei entsprechend vergrößertem Alphabet, in ein einziges Feld speichern.

5. Die erlaubten "Rechen"operationen sind:
(a) mit dem Lesekopf auf dem Band ein Feld nach rechts gehen
(b) mit dem Lesekopf auf dem Band ein Feld nach links gehen
(c) in das Arbeitsfeld ein Symbol schreiben, das evtl. vorher vorhandene Symbol wird
überschrieben.
Verschieben des Arbeitsfeldes um mehrere Schritte nach rechts oder links bzw. Beschriften von mehreren Feldern lassen sich durch mehrmaliges Anwenden dieser Einzelschritte ausführen.

Diese Überlegungen brachten Turing zu der Auffassung, das geschilderte Modell der Berechenbarkeit sei universell, d.h. jede intuitiv berechenbare Funktion lasse sich in seinem Modell berechnen. Sie sind ein starkes Argument für die Churchsche These, denn tatsächlich sind die Turingmaschinen-berechenbaren Funktionen genau die rekursiven; siehe 9.5.

9.2 Definition Eine Turingmaschine ist ein Automat, der sich in endlich vielen wohldefinierten Zuständen befinden kann und seine Aktionen in Abhängigkeit vom Zustand und dem Inhalt des Arbeitsfeldes ausführt; häufig spielt einer der Zustände als Anfangszustand z_a und einer als Endzustand z_e eine Sonderrolle.
Die mathematisch genaue Definition ist folgende. Eine *Turingmaschine* über dem Alphabet $A = \{s_1, \ldots, s_n\}$ und der Zustandsmenge $Z = \{z_1, \ldots, z_k\}$ ist eine Menge M von (Befehls-) Quadrupeln (d.h. Folgen der Länge vier)

$$q = (z, s, f, z'),$$

mit: $z \in Z$ und $z' \in Z, s \in A \cup \{\star\}^6, f \in A \cup \{\star\} \cup \{l, r\}$ (wir nennen z den Anfangszustand, z' den Endzustand des Quadrupels, s das gelesene Symbol und

[6] $A \cup B$ ist die Vereinigungsmenge $\{x : x \in A \text{ oder } x \in B\}$

f die ausgeführte Aktion), so daß für jedes Paar (z, s) in M *höchstens* ein Quadrupel q in M existiert, das mit z und s beginnt.
Die Arbeitsweise einer Turingmaschine denken wir uns so: befindet sich die Maschine in einem gewissen Zeitpunkt im Zustand z und liest das Symbol s auf dem Arbeitsfeld und ist $q = (z, s, f, z')$ dasjenige Quadrupel aus M, das mit z und s beginnt, so geht sie in den Zustand z' über. Vorher geht sie einen Schritt nach rechts, falls $f = r$, einen Schritt nach links, falls $f = l$; falls $f \in A \cup \{\star\}$, schreibt sie das Symbol f auf das Arbeitsfeld. Beginnt kein Quadrupel in M mit z und s, so stoppt sie und hat damit die Rechnung beendet. – Setzt man die Maschine in einem der Zustände aus Z auf ein mit Symbolen aus $A \cup \{\star\}$ beschriebenes Band an, so ist also ihr Rechnen durch die Quadrupel aus M eindeutig festgelegt.

9.3 Turing-berechenbare Funktionen Sei $f : I\!N^k \hookrightarrow I\!N$ eine k-stellige partielle Funktion. Wir nennen f *Tuiing-berechenbar*, wenn es eine Turingmaschine M gibt, die folgendes leistet. M benutzt außer dem Leersymbol $\star$ das Alphabet $\{0, 1\}$ und hat die Zustandsmenge Z mit zwei ausgezeichneten Zuständen z_a und z_e; M bearbeitet natürliche Zahlen in Dualzahldarstellung. (Ebenso kann man natürlich Maschinen definieren, die natürliche Zahlen in Dezimalzahl- oder anderer Darstellung bearbeiten; wir legen uns hier der Deutlichkeit halber auf eine bestimmte Darstellung fest). Sind $x_1, \dots, x_k \in I\!N$ und wird die Maschine im Zustand z_a angesetzt auf die Situation

$$\begin{array}{cccccccccc} \hline \dots & \star & x_1 & \star & x_2 & \star & \dots & \star\, x_k & \star & \dots \\ \hline & & \uparrow & & & & & & & \end{array}$$

bei der $x_1, \dots, x_k$ in Dualschreibweise dargestellt sind, so stoppt sie nach endlich vielen Schritten genau dann, wenn $f(x_1, \dots, x_k)$ definiert ist, und zwar auf der Bandkonfiguration

$$\begin{array}{ccccc} \hline \dots & \star & f(x_1 \dots x_k) & \star & \dots \\ \hline & & \uparrow & & \end{array}$$

im Zustand z_e. Im Arbeitsfeld soll dabei das am weitesten links gelegene Symbol von x_1 bzw. $f(x_1, \dots, x_k)$ stehen.

9.4 Beispiel Die folgende Maschine arbeitet mit dem Symbolvorrat $\{\star, 0, 1\}$ und der Zustandsmenge $\{z_a, z_1, z_2, z_3, z_e\}$. Sie berechnet die Nachfolgerfunktion $s : I\!N \to I\!N$ mit $s(x) = x + 1$. Im Anfangszustand z_a läuft die Maschine ans rechte Ende der Dualdarstellung von x, in den Zuständen z_1 und z_2 wird die 1 (mit Übertrag) addiert, im Zustand z_3 wird zurück ans linke Ende der Darstellung von $x + 1$ gelaufen. Schließlich bleibt die Maschine im Endzustand z_e stehen.

$z_a\, 0\, r\, z_a$	$z_2\, 0\, l\, z_1$
$z_a\, 1\, r\, z_a$	$z_3\, 0\, l\, z_3$
$z_a \star l\, z_1$	$z_3\, 1\, l\, z_3$
$z_1\, 0\, 1\, z_3$	$z_3 \star r\, z_e$
$z_1\, 1\, 0\, z_2$	
$z_1 \star 1\, z_e$	

9.5 Turing-Berechenbarkeit und Rekursivität Die Turing - berechenbaren Funktionen sind genau die rekursiven. Der Beweis verläuft grundsätzlich wie in 5.2: einerseits sieht man wie in 3.7, daß jede rekursive Funktion Turing-berechenbar ist, denn die Klasse der Turing-berechenbaren Funktionen umfaßt alle Ausgangsfunktionen und ist abgeschlossen unter Einsetzung, Rekursion und Anwendung des μ-Operators. Die Einzelheiten dieses Beweisteils sind allerdings sehr mühsam, da bereits Turing-Berechnungen für die Ausgangsfunktionen recht umfangreich sind - siehe Beispiel 9.4. Andererseits überlegt man wie in 5.5, daß es eine rekursive Funktion gibt, die (in Abhängigkeit von $e, x_1, \ldots, x_k$) das schrittweise Rechnen der Turingmaschine mit Gödelnummer e, angesetzt auf natürliche Zahlen $x_1, \ldots, x_k$, simuliert; daher ist jede Turing-berechenbare Funktion rekursiv.

10. Komplexität

Im Gegensatz zu den prinzipiellen Betrachtungen zum Begriff der Berechenbarkeit bzw. Entscheidbarkeit in den vorigen Abschnitten (siehe die Bemerkung am Ende von Abschnitt 1) diskutieren wir hier für entscheidbare Probleme die *Komplexität* eines Algorithmus, d.h. den Aufwand an Zeit, Platz oder anderen Ressourcen, den er benötigt. Der Einfachheit halber befassen wir uns nur mit der Zeitkomplexität, d.h. der Anzahl der Rechenschritte, verglichen mit der Länge der Eingabedaten. Wenngleich in 10.1 und 10.2 die Begriffe "Länge" und "Anzahl der Rechenschritte" für das Modell der Turingmaschine genauer definiert werden, ist es für die Anschauung zweckmäßig, sie weniger formal zu verstehen, wie es auch in einigen der angedeuteten Beispiele geschieht.

10.1 Alphabete, Worte und Sprachen Sei A eine endliche Menge mit mindestens zwei Elementen; wir nennen A ein Alphabet und die Elemente von A die Buchstaben. Ein Wort über A ist eine endliche Folge $w = s_1 \ldots s_n$ über A; die Zahl n ist die Länge von w. Beispiele wären etwa:

1. $A = \{a, \ldots, z\}$
2. $A = \{0, 1\}$ – die Worte über A sind die Dualdarstellungen natürlicher Zahlen
3. $A = \{0, \ldots, 9\}$ – die Worte über A sind die Dezimaldarstellungen natürlicher Zahlen.

Die Zahl Eintausendunddrei hat also die Dezimaldarstellung 1003 mit der Länge 4, ihre Dualdarstellung ist 11 1110 1011 mit der Länge 10.
Mit A^* bezeichnen wir die Menge aller Worte über A; sie ist ein effektiv gegebener Bereich im Sinne von Abschnitt 1. Jede Teilmenge L von A^* nennen wir eine *Sprache* oder auch ein *Problem* über dem Alphabet A; wir identifizieren L mit dem Entscheidungsproblem "$w \in L$?" für $w \in A^*$ und interessieren uns für die Komplexität von Entscheidungsalgorithmen für entscheidbare Probleme $L \subseteq A^*$.
Als Beispiel beschreiben wir informal eine für uns besonders interessante Sprache SAT, die wir schon in 1.2(c) angedeutet hatten. Ihr Alphabet sei $A = \{p, 0, 1, (,), \neg, \wedge, \vee\}$. Jede aussagenlogische Formel mit den Aussagenvariablen $p_0, p_1, p_2, \ldots$ können wir als Wort über A schreiben, wenn wir p_n durch die Folge darstellen, die aus dem Buchstaben p und der Dualdarstellung von n besteht – z.B. die Variable p_9 durch das Wort $p\,1001$. Nun sei SAT die Menge derjenigen Worte über A, die eine erfüllbare Formel darstellen (erfüllbar = *sat*isfiable auf englisch); diese Sprache ist mit dem bekannten Wahrheitstafelverfahren entscheidbar. Jedoch ist die Komplexität dieses Verfahrens sehr hoch: faßt man (grob vereinfachend) in einer Formel w die Anzahl n der in w vorkommenden Variablen als die Länge von w auf, so muß man eine Wahrheitstafel mit 2^n Zeilen ausrechnen, um zu entscheiden, ob $w \in$ SAT, d.h. für Worte mit der Länge n sind (wiederum vereinfachend) 2^n Rechenschritte erforderlich. Die Laufzeit des Algorithmus wächst exponentiell mit der Länge n der Worte, er ist damit für große Werte von n praktisch undurchführbar.

10.2 Polynomiell entscheidbare Probleme; die Klasse P Sei L eine entscheidbare Sprache über dem Alphabet A. L heißt *in polynomieller Zeit entscheidbar*, falls es eine Turingmaschine M gibt, die L entscheidet (d.h. die charakteristische Funktion $\chi_L : A^* \to \{0, 1\}$ berechnet) und ein Polynom $p(x)$ mit Koeffizienten aus $\mathbb{N}$, so daß folgendes gilt: für jedes Wort w in A^* mit der Länge n erfordert die Berechnung von $\chi_L(w)$ durch M höchstens $p(n)$ Schritte. Als einen Schritt verstehen wir dabei die Abarbeitung eines Befehlsquadrupels von M. Mit **P** bezeichnen wir die Klasse aller in polynomieller Zeit entscheidbaren Sprachen (über beliebigen endlichen Alphabeten).
Die obige Definition der Klasse **P** wird dem Leser zunächst recht verwickelt erscheinen; man ist aber überzeugt, daß **P** eine wichtige und natürliche Klasse von Entscheidungsproblemen beschreibt. Es ist z.B. nicht allzu schwierig einzusehen, daß die Definition von **P** weitgehend unabhängig ist vom gewählten Berechenbarkeitsmodell, der Definition der Begriffe "Rechenschritt" und "Länge eines Wortes". Kurz gesagt, liegt ein Problem genau dann in **P**, wenn es entscheidbar ist durch einen Algorithmus, dessen Rechenzeit für alle Eingaben der Länge n beschränkt ist durch $p(n)$, wobei $p(x)$ eine Polynomfunktion ist. Für viele entscheidbare Probleme zeigen bereits die gängigsten Entscheidungsalgorithmen die Zugehörigkeit zu **P**.

10.3 Die These von Cook und Karp Die Cook-Karpsche These ist die folgende nach den amerikanischen Informatikern Stephen A. Cook und Richard M. Karp benannte Behauptung: eine Sprache $L \subseteq A^*$ ist genau dann praktisch entscheidbar, wenn sie zu **P** gehört. Anschaulich gesprochen: praktisch entscheidbar sind genau die in polynomieller Zeit entscheidbaren Probleme. "Praktisch entscheidbar" (auf englisch: tractable) bedeutet dabei, daß es ein Entscheidungsverfahren gibt, das sich nicht nur grundsätzlich, sondern mit realistisch vertretbarem Aufwand durchführen läßt.
Die Cook-Karpsche These ist philosophisch sicherlich weniger abgesichert als Churchs These, zumal der Begriff "praktisch entscheidbar" recht vage ist; sie wird aber in der theoretischen Informatik weitgehend akzeptiert. Zwingend scheint jedenfalls die schwächere ihrer Implikationen: ein Algorithmus, dessen Laufzeit nicht polynomiell beschränkt ist, ist nicht praktisch durchführbar.

10.4 Nichtdeterministische Algorithmen Interessanterweise lassen sich viele wichtige Entscheidungsprobleme in polynomieller Zeit lösen, wenn man einen wesentlich liberalisierten Algorithmenbegriff zugrunde legt, den des *nichtdeterministischen Algorithmus* oder *Zufallsalgorithmus*. Im Gegensatz zur Definition eines deterministischen Algorithmus in Abschnitt 1 darf bei einem Zufallsalgorithmus in manchen Schritten des Rechenverfahrens eine Wahl zwischen mehreren (jedoch endlich vielen) Möglichkeiten getroffen werden; wir stellen uns vor, daß der Rechner eine der erlaubten Möglichkeiten "rät" bzw. seine Wahl durch Würfeln oder Anwenden eines Zufallsgenerators trifft.
Für einen Zufallsalgorithmus α und ein gegebenes Wort w in A^* als Eingabe kann es mehrere Rechengänge gemäß α geben. Wir nennen einen solchen Rechengang eine *w akzeptierende Berechnung*, wenn der Rechner nach endlich vielen Schritten stehen bleibt und die Ausgabe 1 zeigt; w heißt dann von *α akzeptiert*. Die Menge aller von α akzeptierten Worte $w \in A^*$ nennen wir die von *α akzeptierte Sprache*.
Diese informalen Betrachtungen lassen sich durch eine Abänderung des Begriffs der Turingmaschine sehr leicht präzisieren: Wir nennen (mit einem etwas unglücklichen Sprachgebrauch) jede Menge M von Quadrupeln $q = (z, s, f, z')$ wie in 9.2 eine *nichtdeterministische Turingmaschine*, wobei aber jetzt die Forderung aus 9.2 entfällt, daß für jedes Paar (z, s) (z Zustand, s Symbol) höchstens ein Quadrupel in M mit z und s beginnt. Beginnen mehrere Befehlsquadrupel von M mit demselben Paar (z, s), so darf der Rechner einen dieser Befehle wählen und mit ihm weiterrechnen.
Das Erfüllbarkeitsproblem SAT aus 10.1 läßt sich mit einem nichtdeterministischen Algorithmus schnell, nämlich mit polynomiellem Zeitaufwand, lösen: für eine gegebene aussagenlogische Formel w, in der n Variable vorkommen, rät der Algorithmus für jede der Variablen einen Wahrheitswert (grob gesprochen, n Rechenschritte) und berechnet dann den Wahrheitswert von w unter der geratenen Belegung der Variablen mit Wahrheitswerten (insgesamt $n + 1$ Schritte). Die von diesem Algorithmus akzeptierten Formeln sind genau die erfüllbaren.

10.5 Die Klasse NP; das P=NP-Problem Eine Sprache $L \subseteq A^*$ gehört zur Klasse **NP** (sie ist *nichtdeterministisch in polynomieller Zeit entscheidbar*), falls es einen nichtdeterministischen Algorithmus α und ein Polynom p gibt mit:
1. L ist die von α akzeptierte Sprache
2. für jedes $w \in L$ von der Länge n gibt es (mindestens) einen Rechengang gemäß α, der w akzeptiert und höchstens die Länge $p(n)$ hat.

Z.B. liegt nach der Beweisskizze aus 10.4 das Erfüllbarkeitsproblem SAT in **NP**. Offensichtlich gilt $\mathbf{P} \subseteq \mathbf{NP}$: jedes deterministisch in polynomieller Zeit entscheidbare Problem ist insbesondere nichtdeterministisch in polynomieller Zeit entscheidbar. Offen ist jedoch das

P=NP-Problem Ist $\mathbf{P} = \mathbf{NP}$, d.h. ist jedes nichtdeterministisch in polynomieller Zeit entscheidbare Problem auch deterministisch in polynomieller Zeit entscheidbar?

Es gilt als das berühmteste ungelöste Problem der theoretischen Informatik. Eine positive Lösung (die allerdings nach allgemeiner Einschätzung sehr unwahrscheinlich ist) würde bedeuten, daß zahlreiche Probleme, für die bisher nur Algorithmen mit großem (i. a. exponentiellem) Zeitaufwand bekannt sind, schnelle Lösungsalgorithmen haben, d.h. praktisch entscheidbar sind.

10.6 NP-vollständige Probleme Wie in 8.5 für rekursiv aufzählbare Mengen kann man auch für entscheidbare Sprachen einen der Komplexitätstheorie angepaßten Begriff der Reduzierbarkeit betrachten. Wir nennen eine Sprache K *polynomiell reduzierbar* auf die Sprache L, falls es eine Funktion $s : A^* \to A^*$ gibt mit:
1. s reduziert K auf L, d.h. für alle $w \in A^*$ gilt

$$w \in K \iff s(w) \in L$$

2. s ist durch einen deterministischen Algorithmus in polynomieller Zeit berechenbar (d.h. es gibt ein Polynom p, so daß für $w \in A^*$ mit der Länge n der Wert $s(w)$ in höchstens $p(n)$ Schritten berechnet wird).

Eine Sprache L heißt **NP**-vollständig, falls sie selbst zu **NP** gehört und jede weitere Sprache $K \in \mathbf{NP}$ polynomiell auf L reduzierbar ist. Die **NP**-vollständigen Probleme sind die "schwierigsten" in **NP**, ebenso wie in 8.5 die Menge H^* (die Codierung des Halteproblems) die komplizierteste rekursiv aufzählbare Menge war.

Die **NP**-vollständigen Probleme bilden eine interessante Klasse, da zahlreiche Probleme in **NP**, darunter auch praktisch relevante aus der diskreten Mathematik, sich als **NP**-vollständig erwiesen. Eines dieser Probleme ist, wie Cook bewiesen hat, das Erfüllbarkeitsproblem SAT. Außerdem läßt sich das $\mathbf{P} = \mathbf{NP}$- Problem auf die Betrachtung **NP**-vollständiger Probleme zurückführen: es ist leicht zu sehen, daß $\mathbf{P} = \mathbf{NP}$ schon dann gilt, wenn wenigstens ein **NP**- vollständiges

Problem in **P** liegt. Nach dem Satz von Cook ist also genau dann $\mathbf{P} = \mathbf{NP}$, wenn SAT $\in \mathbf{P}$.

11. Literatur

Wissenschaftlich orientierte Darstellungen und Überblicke

M. Davis: Unsolvable Problems; in: Handbook of Mathematical Logic; 1977; Herausgeber J. Barwise; S. 567-594

M. Davis, E. Weyuker: Computability, Complexity and Languages; 1983; Academic Press; Kapitel 2-6, 15

H. Enderton: Elements of Recursion Theory; in: Handbook of Mathematical Logic; 1977; Herausgeber J. Barwise; S. 527-566

A. Oberschelp: Rekursionstheorie; 1993; BI Wissenschaftsverlag; Kapitel I-III

P. Odifreddi: Classical Recursion Theory; 1989; North Holland Publishing Company; Kapitel I

W. Pohlers: Mathematische Grundlagen der Informatik; 1993; R. Oldenbourg Verlag; Kapitel 1-3, 6

A. Salomaa: Computation and Automata; 1985; Cambridge University Press; Kapitel 1, 4.1-4.4, 5, 6

Schulnahe Darstellungen

F. Gasper, I. Leiß, M. Spengler, H. Stimm: Technische und theoretische Informatik; 1992; Bayerischer Schulbuch-Verlag München; Kapitel 12-16

K. Menzel, J. Schornstein, S. Stief: Grundkurs Mathematik, Materialien zur Lehrerfortbildung und Weiterbildung für Lehrer der Sekundarstufen, Band V, 1, Elektronische Datenverarbeitung; Deutsches Institut für Fernstudien im Medienverbund DIFF; Kapitel 2-3

E. Modrow: Automaten, Schaltwerke, Sprachen, Lehr- und Übungsbuch der technischen und theoretischen Informatik; 1986; Ferdinand Dümmler's Verlag, Bonn; Kapitel 8

E. Modrow: Zur Didaktik des Informatik-Unterrichts, Band 2; 1992; Ferdinand Dümmler's Verlag, Bonn; Kapitel 9, insbesondere 9.5-9.6