1969-10-00

TITUTE FÜR ANGEWANDTE MATHEMATIK
IVERSITÄT · TECHNISCHE HOCHSCHULE
GRAZ

Steiermärkisches Mathematisches Symposium
Grottenhof-Hardt 6.-9. Okt. 1969

GRUNDBEGRIFFE DER ALGORITHMENTHEORIE

B. Buchberger
Inst. f. Rechentechnik, Univ. Innsbruck

(Vortragsauszug)

A-8010 G[illegible]EYRERGASSE 17

INHALT

Seite

1. Einführung in die Fragestellung 1
2. Definition der Grundbegriffe mit Hilfe des Begriffs der Turingmaschine 5
   2.1. Schreibweise und Bezeichnungen..................... 5
   2.2. Definition der Turingmaschinen 6
   2.3. Definition von "berechenbar", "aufzählbar", "entscheidbar" 11
   2.4. Zusammenhang zwischen den Begriffen "berechenbar", "aufzählbar", "entscheidbar" ... 13
   2.5. Skizze eines Unentscheidbarkeitsbeweises 14
   2.6. Turingmaschinen, von einem übergeordneten Standpunkt aus betrachtet 16
   2.7. Universelle Turingmaschinen 18
3. Definition der Grundbegriffe mit Hilfe anderer Konzepte 20
   3.1. Übersicht über die anderen Begriffsbildungen .. 20
   3.2. μ-rekursive Funktionen 22
   3.3. Äquivalenz der Begriffe "berechenbar im Sinne Turings" und μ-rekursiv" 23
4. Resultate der Algorithmentheorie in verschiedenen Anwendungsgebieten 25
   4.1. Beweismethoden 25
   4.2. Beispiel Gruppentheorie 25
   4.3. Beispiel Logik 27
   4.4. Beispiel Syntax formaler Sprachen..................... 28

Literatur 30,31

1. EINFÜHRUNG IN DIE FRAGESTELLUNG

Man kann - wenn man so will - zwei Grundtypen mathematischer Fragestellungen unterscheiden. Bei den Fragen des ersten Typs geht es darum, Aussagen, die innerhalb gewisser sprachlicher Systeme formuliert sind und einen Sachverhalt beschreiben, (in irgendeinem Sinne) zu beweisen. Beispiel: in einer geeigneten Mengenlehre kann der von G. Cantor vermutete Äquivalenzsatz "Wenn von zwei Mengen M und N jede einer Teilmenge der anderen äquivalent ist, so ist M äquivalent N" formuliert werden. Die (in diesem Falle von F. Bernstein gelöste) Aufgabe für den Mathematiker ist es, die Wahrheit oder Falschheit dieses Satzes zu beweisen (hier in dem Sinne: beweisen = ableiten aus den Axiomen einer Mengenlehre).

Ganz anders ist die Situation, wenn nach Verfahren gesucht wird, die natürlich auch wieder innerhalb eines gewissen sprachlichen Systems formuliert sein müssen und die, auf bestimmte Ausgangsgegenstände (Angaben, Datensätze, Argumente, Eingabewerte) angewendet, eine gewisse Klasse von Resultatgegenständen mit verlangten Eigenschaften liefern sollen. Aufgaben dieser Art haben eine der folgenden Gestalten:

a) Gib ein Verfahren an, das zu jedem Element aus dem Definitionsbereich einer Funktion (in einem sehr weiten Sinne) den zugehörigen Funktionswert berechnet.

b) Gib ein Verfahren an, das für jedes Element einer gegebenen Menge entscheidet, ob es eine gewisse Eigenschaft besitzt oder nicht.

c) Gib ein Verfahren an, das bei seiner Anwendung nach und nach sämtliche Elemente einer gewissen Menge liefert ("aufzählt").

Beispiele:

ad a) Gib ein Verfahren an, das zu jedem System linearer [illegible]braischer Gleichungen eine Lösung berechnet (die [illegible] kann als Funktion der spez[illegible]ffizienten [illegible]leichungssystems aufgefaßt [illegible] muß nicht für alle Argumente definiert sein).

ad b) Gib ein Verfahren an, das für jeden Ausdruck, der nur aus ALGOL-Symbolen zusammengesetzt ist, <u>entscheidet</u>, ob er ein syntaktisch richtiges ALGOL-Programm ist.

ad c) Gib ein Verfahren an, das nach und nach alle grammatikalisch richtigen Sätze der deutschen Sprache <u>aufzählt</u>.

Die beiden angegebenen Grundtypen mathematischer Fragestellungen entsprechen zwei Grundsituationen der menschlichen Sprachanwendung: die Sprache kann zur Mitteilung (Beschreibung) von Zuständen (Gegenständen, Situationen) dienen (<u>deskriptiver</u> Sprachgebrauch) oder zum Befehlen von Vorgängen, die zur Konstruktion neuer Zustände (Gegenstände, Situationen) führen (<u>imperativer</u> Sprachgebrauch). Die deskriptive Sprachanwendung hat ihre ausgeprägteste Form in den formalen Systemen der Mathematik gefunden (z.B. in den verschiedenen formalisierten Mengenlehren), die imperative neuerdings in den formalen Sprachen zur Programmierung elektronischer Rechenanlagen.

Die Umgangssprache wird in dauerndem Wechsel gemischt zu dem einen oder anderen Zweck verwendet und hat auch syntaktisch eigene Ausdrucksmöglichkeiten für die beiden Anwendungen gebildet.

Zwischen den beiden Sprachanwendungen in der Mathematik bestehen mannigfache Beziehungen, die selbst Gegenstand einer mathematischen Untersuchung sein können. So kann eine zu beweisende Aussage (oder eine Klasse solcher Aussagen) selbst als ein Gegenstand aufgefaßt werden, der mit Hilfe eines geeigneten allgemeinen Verfahrens konstruiert werden muß oder dessen Wahrheit mit Hilfe eines allgemeinen Verfahrens entschieden werden soll. Viele Bemühungen der modernen Logik ([illegible]tion und Untersuchung forma[illegible]) und der [illegible] Science (maschinelles Beweis[illegible] den Bestre[illegible] [illegible]lichst allgemeine derartige [illegible] erfinden.

Andererseits ist nach Angabe eines Verfahrens V zu beweisen, daß die Gegenstände, die durch Anwendung des Verfahrens auf die Eingangswerte x entstehen, tatsächlich die verlangte Eigenschaft P (z.B. Lösung eines Gleichungssystems zu sein) besitzen. Es müßte also in einem geeigneten mathematischen System (in stilisierter Schreibweise)

$$(\forall x)(P(V(x)))$$

gezeigt werden. In der Programmierpraxis wird diese Beweisaufgabe meist dadurch "gelöst", daß man für mehrere x das Resultat V(x) betrachtet und hofft, daß das Verfahren für alle x das richtige Resultat liefert, wenn das für einige "markante" x der Fall ist. Diese unbefriedigende Situation erhält heute meines Wissens noch wenig Unterstützung aus der Theorie. Das hängt vor allem damit zusammen, daß für die meisten Sprachen zur Angabe von Verfahren, keineswegs klar definiert ist, wie diese Sprachen interpretiert werden müssen. Man denke z.B. nur an ALGOL, dessen Syntax durch den Report von Naur u.a. 1960 (wenigstens zum Teil) klar definiert ist, dessen Interpretation aber nur in vagen umgangssprachlichen Sätzen formuliert wurde. Erst in jüngster Zeit (1968) ist hier durch die Arbeiten des Wiener IBM-Laboratoriums ein entscheidender Fortschritt gemacht worden.

Die Algorithmentheorie kann als die Mathematik der "imperativen" Sprachanwendung aufgefaßt werden, und zwar nicht in dem Sinne, daß Verfahren für die Lösung dieser oder jener konkreten Aufgabe angegeben werden, sondern vielmehr daß der Begri des "Verfahrens" selbst zum Gegenstand der Untersuchung gemacht wird, insbesondere klar definiert werden soll, was ein "effektives", (konstruktives, maschinelles, automatisierbares, ... u.ä.) Verfahren (Algorithmus) ist.

[illegible] mathematischen Umgangssprach[illegible] mit dem Titel [illegible]thmus" solche Verfahren ausg[illegible] die

a) ein gewisses Maß an Allgemeinheit besitzen, d.h. bei einer ganzen Reihe ähnlicher Aufgaben anwendbar sind (z.B. zur Lösung aller linearen algebraischen Gleichungssysteme mit nichtverschwindender Determinante)
b) den Lösungsweg eindeutig beschreiben, d.h. insbesondere auch beim Ausführenden weder "Einsicht" noch "schöpferische Leistungen" voraussetzen, d.h. eine automatische Abarbeitung zulassen
c) sich als endlichen Text in irgendeiner Sprache formulieren lassen
d) den gesuchten Gegenstand, falls sich dieser durch ein endliches Gebilde darstellen läßt, nach endlich vielen Schritten liefert.

(Die Forderung c) und d) wird oft auch anders formuliert oder weggelassen).

Diese umgangssprachliche Definition genügt so lange, als man nur gewissen besonders gebräuchlichen und augenscheinlich allgemein anwendbaren und mechanischen Verfahren eine Etikette geben will (Euklidischer Algorithmus, Gauß'scher Algorithmus). Sobald man jedoch die Frage stellt, ob diese oder jene Aufgabenklasse prinzipiell mit Hilfe eines algorithmischen Verfahrens lösbar ist, oder man versuchen will, die Unmöglichkeit der Existenz eines Algorithmus zu beweisen (im Rahmen eines gewissen mathematischen Systems), muß (in diesem System) klar definiert sein, was als Algorithmus angesehen werden kann und was nicht. Dieser Präzisierung dient z.B. der Begriff der Turingmaschinen.

2. DEFINITION DER GRUNDBEGRIFFE MIT HILFE DES BEGRIFFS DER TURINGMASCHINE

2.1. Schreibweise und Bezeichnungen

N bezeichne die Menge der natürlichen Zahlen, die Null eingeschlossen.

$S_1 \times \ldots \times S_n$ bezeichne das Cartesische Produkt der Mengen $S_1, \ldots, S_n$.

S^n bezeichnen das n-fache Cartesische Produkt der Menge S.

Eine Untermenge $R \subset S_1 \times \ldots \times S_n$ heißt n-stellige Relation zwischen $S_1, \ldots, S_n$.

Eine Untermenge $R \subset S^n$ heißt n-stellige Relation über S.

Sei $f \subset S_1 \times \ldots \times S_{n+1}$ eine (n+1)-stellige Relation und $D \subset S_1 \times \ldots \times S_n$. Dann heißt f eine n-stellige Funktion mit Definitionsbereich D und Wertbereich S_{n+1} (Schreibweise: $f: D \to S_{n+1}$), wenn

(1) $(\forall (x_1, \ldots, x_n) \in D)(\exists\, y \in S_{n+1})((x_1, \ldots, x_n, y) \in f)$
("auf D überall definiert")

(2) $(\forall (x_1, \ldots, x_n) \in D)(\forall\, y_1, y_2 \in S_{n+1})$
$(x_1, \ldots, x_n, y_1) \in f \wedge (x_1, \ldots, x_n, y_2) \in f \Rightarrow y_1 = y_2)$
("eindeutig definiert").

Ein Alphabet ist eine beliebige (wenn nicht ausdrücklich anders vorausgesetzt: endliche) Menge, deren Elemente Zeichen (Buchstaben, Symbole) heißen.

A^* bezeichne die Menge der Wörter über dem Alphabet A, (das leere Wort nicht eingeschlossen).

Alle hier in Betracht gezogenen Alphabete mögen aus Elementen $a_1, \ldots, a_n$ bestehen, die zusammen ein endliches "Anfangsstück" einer geordneten, abzählbaren Menge $\{a_1, a_2, \ldots,\}$, die wir A_T nennen wollen, bilden. a_1 sei das "Strich"-Symbol I, a_0 (das "Leerzeichen"), r, [illegible] Symbole, die in A_T nicht vorkommen.

2.2. Definition der Turingmaschinen

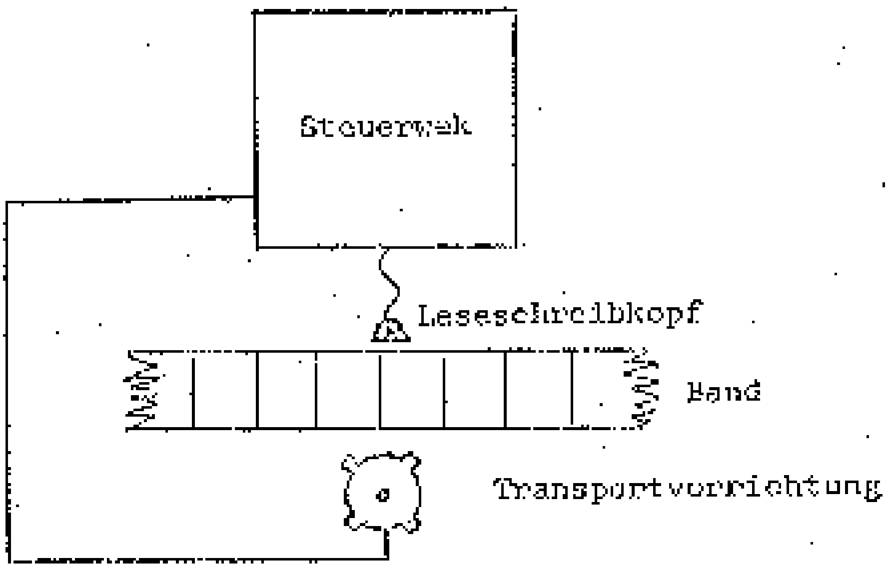
Definition 1: Eine Turingtafel (Turingmaschine) M über dem Alphabet $A_M = \{a_1,\dots,a_n\}$ mit der Zustandsmenge $C_M = \{c_1,\dots,c_m\}$ ($c_j \in \mathbb{N}$, alle c_j voneinander verschieden ($j=1,\dots,m$)) ist ein Gebilde folgender Art:

$$
\begin{array}{lllll}
 & a_0 & a_1 & \cdots\cdots\cdots & a_n \\
c_1 & (b_{1,o},c'_{1,o}) & (b_{1,1},c'_{1,1}) & \cdots\cdots\cdots & (b_{1,n},c'_{1,n}) \\
c_2 & (b_{2,o},c'_{2,o}) & (b_{2,1},c'_{2,1}) & \cdots\cdots\cdots & (b_{2,n},c'_{2,n}) \\
\vdots & & & & \\
c_m & (b_{m,o},c'_{m,o}) & (b_{m,1},c'_{m,1}) & \cdots\cdots\cdots & (b_{m,n},c'_{m,n})
\end{array}
$$

Zu jeder durch ein c_j gekennzeichneten Zeile und jeder durch ein a_i gekennzeichneten Spalte dieser Matrix gehört also ein Symbolpaar $(b_{j,i},c'_{j,i})$, wobei jedes $b_{j,i} \in A_M \cup \{a_o,r,l,h\}$ und jedes $c'_{j,i} \in C_M$. Die Elemente von C_m heißen Zustände von M. (Falls notwendig, werden wir auch die einzelnen Elemente von A_M und C_M mit dem Index M versehen, um ihre Zugehörigkeit zur speziellen Turingmaschine M zu verdeutlichen). $c_{1,M}$ heißt der Anfangszustand von M.

Eine Turingmaschine ist im Augenblick noch nichts anderes als ein sprachlicher Ausdruck, der aus endlich vielen Zeichen über dem Alphabet $A_M \cup C_M \cup \{a_o,r,l,h\}$ besteht. Er soll zur Beschreibung eines Verfahrens dienen und muß deshalb noch "interpretiert" werden, d.h. es muß gesagt werden, welchen Vorgang er beschreibt.

Um eine heuristische [illegible] zur exakten Definition des d[illegible] eine Turingtafel be[illegible]en Vorganges zu haben, [illegible] stellen wir uns stilisierte Maschinen (die "Turingmaschinen") von folgendem Typ vor:



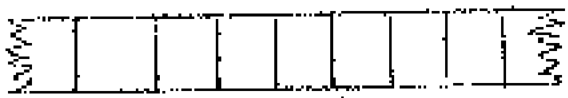
Ein nicht näher detailliertes Steuerwerk hat die in M enthaltene Information "verdrahtet" und kann die "Zustände" $c_1, \ldots, c_m$ annehmen. Ein nach beiden Seiten hin beliebig fortsetzbares Band, das in gleiche Felder eingeteilt ist, deren jedes eines der Symbole $a_o, a_1, \ldots, a_n$ tragen kann, kann durch eine Transportvorrichtung um jeweils ein Feld nach rechts oder links verschoben werden. Über einen Leseschreibkopf kann in das gerade unter dem Kopf liegende Feld (das "Arbeitsfeld") eines der Zeichen $a_o, \ldots, a_n$ eingetragen oder aus diesem ~~Feld abgelesen~~ gelesen werden.

Das Verhalten der ~~Maschine~~ stellt man sich in diskreten Arbeitsschritten ablaufend vor. Befinde sich die Maschine z.B. im Zustand c_p [illegible] im Arbeitsfeld gerade das Zeichen a_q eingetragen [illegible] ist ihr Verhalten im nächsten Arbeitsschritt [illegible] durch das (im Steuerwerk "ver[illegible] Element $(b_{pq}$ [illegible] bestimmt:

a) die Maschine bringt das Feld rechts (links) vom Arbeitsfeld unter den Leseschreibkopf, falls $b_{q,p} = r$ (l) und geht in den Zustand $c'_{p,q}$.
b) die Maschine trägt das Symbol a_r in das Arbeitsfeld ein, falls $b_{p,q} = a_r$ und geht in den Zustand $c'_{p,q}$.
c) die Maschine stoppt, falls $b_{p,q} = h$.

Das anschauliche Verhalten dieser Turingmaschinen wird im Folgenden mit den sprachlichen Mitteln des von uns verwendeten (nicht näher abgegrenzten) mathematischen Systems wie folgt beschrieben:

Definition 2: Eine Funktion $t: G \to \{a_0\} \cup A_M$ heißt M-Bandausdruck. (G...Menge der ganzen Zahlen).
(Anschaulich: ein M-Bandausdruck gibt die Buchstaben, einschließlich Leerzeichen, die im Band einer Turingmaschine eingetragen sind, wenn man sich die Felder des Bandes in irgendeiner Weise numeriert denkt. Im n-ten Feld des Bandes einer Turingmaschine steht also t(n). Eine denkbare Numerierung der Felder wäre z.B.:



-2 -1 0 1 2 3 4 ....).

Definition 3: Ein Tripel (s,t,u), wo $s \in G$ die Nummer eines Feldes, t ein M-Bandausdruck und $u \in C_M$ ist, heißt M-Konfiguration. Konfigurationen mit $u = c_{1,M}$ heißen M-Anfangskonfigurationen.

Jede M-Konfiguration (s,t,u) legt eindeutig durch den Zustand u und das Symbol t(s) (anschaulich: die "Eintragung im Arbeitsfeld") das Element $(b_{u,t(s)}, c'_{u,t(s)})$ der Matrix M fest.

Definition 4: (s,t,u) heißt M-Endkonfiguration, wenn
$b_{u,t(s)} = h$

<u>Definition 5</u>: Sei (s,t,u) eine M-Konfiguration, jedoch keine M-Endkonfiguration. Die Konfiguration Nachf(M,(s,t,u)) $=(s^M,t^M,u^M)$ mit

$$s^M = \begin{cases} s & \text{wenn} \quad b \in A_M \\ s+1 & \text{wenn} \quad b = r \\ s-1 & \text{wenn} \quad b = l \end{cases}$$

$$t^M(x) = \begin{cases} t(x) & \text{wenn } x \neq s \\ t(x) & \text{wenn } x = s \text{ und } (b = r \text{ oder } b = l) \\ b & \text{wenn } x = s \text{ und } b \in A_M \end{cases}$$

$$u^M = c'_{u,t(s)}$$

(wobei b als Abkürzung für $b_{u,t(s)}$ steht) heißt <u>Nachfolgekonfiguration</u> von (s,t,u) bei M.

Ausgehend von einer beliebigen M-Konfiguration $K_o=(s_o,t_o,u_o)$ wird durch K_{i+1} = Nachf(M,K_i) $(i=o,1,2,\dots)$ eine Folge von weiteren M-Konfigurationen definiert, die endlich ist, wenn für ein gewisses $\bar{i}$ $K_{\bar{i}}$ M-Endkonfiguration ist, und unendlich sonst. (Anschaulich: die Folge der M-Konfiguration gibt ein getreues Bild eines "Rechenablaufs" der Turingmaschine M).

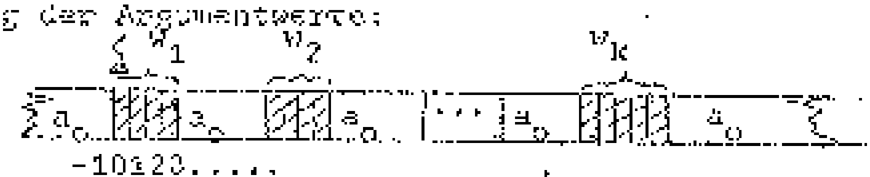
Die rekurrierende Funktion

$$J_T^M(M,K) = \begin{cases} J_T^M(M,\text{Nachf}(M,K)) & \text{wenn K nicht M-Endkonfiguration ist} \\ K \text{ sonst} \end{cases}$$

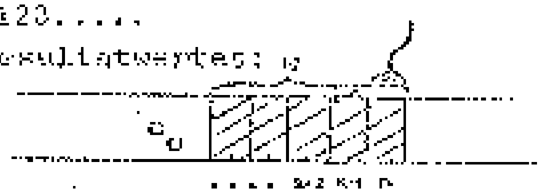
gibt die einer M-Konfiguration K durch M zugeordnete M-Endkonfiguration an, falls eine solche nach endlich vielen Schritten erreicht wird, und ist sonst nicht definiert. (Sprechweise: Falls $J_T^M(M,(s_o,t_o,u_o))=(s_e,t_e,u_e)$ definiert ist, sagt man "M, im Zustand u_o über dem Feld s_o auf den Bandausdruck t_o angesetzt, stoppt nach endlich vielen Schritten über dem Feld s_e auf dem Bandausdruck t_e im Zustand u_e").

Z.B. könnte [illegible] eine durch eine Turingmaschine [illegible] geführte Berech[illegible]folgt ablaufend vorstellen [illegible] die vorgese[illegible]entwerte $w_1,\dots,w_k$, die [illegible] über A_M sein müssen, jeweils durch ein a_o getrennt, auf das ansonsten leere (d.h. mit lauter a_o beschriebene) Band, und zwar ab dem Feld o nach rechts, stellt den Leseschreibkopf auf das Feld o, bringt die Maschine in den Zustand $c_{1,M}$ und läßt sie arbeiten bis sie eventuell stoppt. Sei n die Nummer des Feldes, über dem die Maschine anhält, dann nehmen wir als Resultat w der "Rechnung" jenes Wort über A_M (a_o kommt in einem solchen Wort nicht vor!) das von links her bis zum Feld n reicht.

Anordnung der Argumentwerte:



Anordnung des Resultatwertes:



Wir definieren exakt: Für beliebige Wörter $w_1, \ldots, w_k \in A_M^*$ bilden wir $b_o b_1 \ldots b_p = w_1 a_o w_2 a_o \ldots a_o w_k$ ($b_j \in \{a_o\} \cup A_M$ für $j = o, \ldots, p$).

Mit

$$beg_{M,k}(w_1, \ldots, w_k) = (o, t_{w_1 \ldots w_k}, c_{1,M}), \text{ wobei}$$

$$t_{w_1 \ldots w_k}(x) = \begin{cases} a_o & \text{für } x < o \text{ und } x > p \\ b_x & \text{für } o \leq x \leq p \end{cases}$$

ist für jedes solche k-Tupel von Wörtern eine M-Anfangskonfiguration festgelegt. Andererseits wird durch

$$res_M(s,t,u) = d_1 d_2 \ldots d_v \text{ mit}$$

$$d_v = t(s),\ d_{v-1} = t(s-1), \ldots, d_1 = t(s-v+1), \text{ und}$$

$$d_j \neq a_o \text{ für } j=1, \ldots, v) \text{ und } t(s-v) = a_o$$

für jede M-Konfiguration (und speziell auch für jede M-Endkonfiguration) ein aus ihr zu entnehmendes "Resultat" bestimmt.

Schließlich liefert

$$J_T(M,(w_1,\dots,w_k)) = res_M(J_T^{*}(M,beg_{M,k}(w_1,\dots,w_k)))$$

für jedes n-Tupel $(w_1,\dots,w_k)$ von Wörtern über A_M, für das $J_T^{*}(M,beg_{M,k}(w_1,\dots,w_k))$ definiert ist, ein Wort über A_M als "Resultat".

<u>Definition 6</u>: Die durch

$$f_{M,k}(w_1,\dots,w_k) = J_T(M,(w_1,\dots,w_k))$$

für jede Turingmaschine M definierten Funktionen $f_{M,1}$, $f_{M,2}\dots$ (deren Definitionsbereich nicht immer die Menge aller 1-Tupel, 2-Tupel,... ist) heißen die "<u>zu M gehörigen Funktionen</u>".

2.3. <u>Definition von "berechenbar", "aufzählbar", "entscheidbar"</u>.

<u>Definition 7</u>: A_o sei ein Alphabet. Eine k-stellige Funktion $f:(A_o^{*})^k \to A_o^{*}$ heißt <u>berechenbar</u> (im Sinne Turings, auch: effektiv berechenbar), wenn es eine Turingmaschine M über A (mit $A_o \subset A$) gibt, so daß für alle k-Tupel $(w_1,\dots,w_k)$ von Wörtern über A_o gilt:

$$f(w_1,\dots,w_k) = f_{M,k}(w_1,\dots,w_k).$$

<u>Definition 8</u>: Eine Menge $S \subset A^{*}$ heißt <u>aufzählbar</u> (im Sinne Turings, auch: rekursiv aufzählbar, formal repräsentierbar), wenn es eine berechenbare Funktion $f_S:\{I\}^{*} \to A^{*}$ gibt, so daß

$$S = \{w \mid w = f_S(n) \text{ für ein } n \in \{I\}^{*}\}.$$

(Die Menge [illegible] kann als Darstellung der natür[illegible] verwende[illegible] man z.B. festse[illegible]

$n \in N$ werde durch $\underbrace{III\ldots I}_{n+1 \text{ Striche}}$ repräsentiert.)

(Anschaulich lautet Definition 8: Eine Menge S der obigen Art heißt aufzählbar, wenn es eine Turingmaschine gibt, in die man nacheinander die Zahlen 0,1,2,... eingibt und die als Resultate nacheinander die zu S gehörigen Wörter liefert).

<u>Definition 9</u>: Eine Menge $S \subset A^*$ heißt <u>entscheidbar</u> (im Sinne Turings, rekursiv lösbar, rekursiv, lösbar; man sagt auch: das zu S gehörige Entscheidungsproblem ist lösbar) bezüglich der Symbole e_1 und e_2, wenn es eine berechenbare Funktion $f_S: A^* \to \{e_1, e_2\}$ gibt, so daß

$$f_S(w) = e_1, \text{ wenn } w \in S$$
$$f_S(w) = e_2, \text{ wenn } w \notin S.$$

(Definition 9 heißt anschaulich: Eine Menge S der obigen Art heißt entscheidbar, wenn es eine Turingmaschine gibt, die über dem Buchstaben e_1 stehen bleibt, wenn man ihr ein Wort aus S eingibt, und über e_2 sonst.)

Die Begriffe "aufzählbar" und "entscheidbar" können auch für Relationen definiert werden.

<u>Definition 10</u>: Eine k-stellige Relation R über A^* heißt aufzählbar (im Sinne Turings), wenn es k berechenbare Funktionen $f_{1,R}, \ldots, f_{k,R}$ gibt ($f_{j,R}: (I)^* \to A^*$ für $j=1,\ldots k$), so daß

$$R = \{(w_1, \ldots, w_k) \mid w_j = f_{j,R}(n) \text{ für ein } n \in (I)^* (j=1,\ldots,k)$$

<u>Definition 11</u>: Eine k-stellige Relation R über A^* heißt entscheidbar (im Sinne Turings) bezüglich der Symbole e_1 und e_2, wenn es eine berechenbare Funktion $f_R: (A^*)^k \to \{e_1, e_2\}$ gibt, so daß

$$f_R(w_1, \ldots, w_k) \text{ [illegible] enn } (w_1, \ldots, w_k) \in R$$
$$f_R(w_1, \ldots, w_k) \text{ [illegible] enn } (w_1, \ldots, w_k) \notin R.$$

Man beachte, daß der Begriff der Aufzählbarkeit nicht mit dem der Abzählbarkeit übereinstimmt. Selbstverständlich ist jede aufzählbare Menge abzählbar. Umgekehrt gibt es jedoch nichtaufzählbare, abzählbare Mengen, da die Menge der abzählbaren Wortmengen über einem festen Alphabet überabzählbar, die Menge der aufzählbaren Wortmengen über einem festen Alphabet jedoch abzählbar ist. Das Letztere ergibt sich aus dem Umstand, daß jede berechenbare Funktion durch eine Turingtafel charakterisiert ist und über einem fixen Alphabet nur abzählbar viele solche Tafeln gebildet werden können.

2.4. Zusammenhang zwischen den Begriffen "berechenbar", "aufzählbar", "entscheidbar".

Beziehungen zwischen den Begriffen "berechenbar", "aufzählbar", "entscheidbar" kommen in den folgenden Sätzen zum Ausdruck. Die Beweise dafür geschehen alle konstruktiv in dem Sinne, daß ein in endlich vielen Schritten durchführbares Verfahren angegeben wird, wie die Turingmaschinen, deren Existenz durch die Sätze behauptet wird, konstruiert werden müssen. Heißt ein Satz z.B. "Falls die Menge S entscheidbar ist, so ist sie auch aufzählbar", so muß ein Beweis für diesen Satz ein Verfahren angeben, wie aus einer Turingmaschine M_S, mit deren Hilfe man das Entscheidungsproblem für S lösen kann, eine Turingmaschine M_S' konstruiert werden kann, die die "Aufzählung" von S leistet. Die an und für sich leichten und anschaulichen, jedoch umständlichen Beweise lassen wir hier aus.

Satz 1: (Eine Relation (spezielle Menge) $R \subset (A^*)^k$ ist entscheidbar) $\Longleftrightarrow$

(Die charakteristische Funktion f_R ist berechenbar)

(Die charakteristische Funktion f_R einer Relation $R \subset (A^*)^k$ ist für alle k-Tupel $(w_1,\dots,w_k) \in (A^*)^k$ definiert durch:

$$f_R(w_1,\dots,w_k) = \begin{cases} 1 \text{ falls } (w_1,\dots,w_k) \in R \\ o \text{ falls } (w_1,\dots,w_k) \notin R. \end{cases}$$

<u>Satz 2</u>: (Eine <u>Relation</u> $R \subset (A^*)^k$ ist <u>entscheidbar</u>) $\Longleftrightarrow$

(R <u>und</u> <u>das</u> <u>Komplement</u> $\bar{R}$ sind <u>aufzählbar</u>)

($\bar{R}$ ist definiert durch:

$$\bar{R} = \{(w_1,\dots,w_k) \mid (w_1,\dots,w_k) \in (A^*)^k,\ (w_1,\dots,w_k) \notin R\})$$

<u>Satz 3</u>: (Eine <u>Funktion</u> $f:(A^*)^k \to A^*$ ist <u>berechenbar</u>) $\Longleftrightarrow$

(Der <u>Graph</u> G_f der Funktion ist <u>aufzählbar</u>).

Wenn man den Begriff "Funktion" als undefinierten Grundbegriff nimmt, ist der Graph G einer Funktion f wie folgt definiert: $G_f = \{(w_1,\dots,w_k,w) \mid w=f(w_1,\dots,w_k)\}$, der Graph einer Funktion ist also genau das, was wir in 1. als "Funktion" selbst eingeführt haben.

2.5. <u>Skizze eines Unentscheidbarkeitsbeweises</u>

Der Nutzen einer Präzisierung des Algorithmenbegriffs z.B. durch Einführung des Begriffs "Turingmaschine" liegt unter anderem in der Möglichkeit, die Unentscheidbarkeit bzw. Nicht-Aufzählbarkeit gewisser Mengen zeigen zu können. (Ein "Nutzen" ist dies allerdings nur, wenn man mehr oder weniger "sicher" ist, daß der präzisierte Algorithmenbegriff in etwa mit dem "intuitiven" Algorithmenbegriff übereinstimmt. Zu diesen außerhalb der Mathematik stehenden Überlegungen siehe 3.1.).

Wir setzen zunächst vo[illegible] daß jeder Turingtafel M in ein[illegible]deutiger Weise eine na[illegible] Zahl G_M (in der Darstellung [illegible] als Wörter über dem Al[illegible] zugeordnet sei. Wie eine solche "Gödelnumerierung" der Turingtafel effektiv vorgenommen werden kann, findet man z.B. bei (Hermes), S.103. Da auf Grund unserer Vereinbarungen über die verwendeten Alphabete das Alphabet einer jeden Turingmaschine jedenfalls den Strich enthält, ist für alle Turingmaschinen M die Zahl w_M ein Wort über dem Alphabet von M. "P(M)" stehe für "die Turingmaschine M hält nach endlich vielen Schritten an, wenn man w_M als Eingabewert für M verwendet". Wir fragen nun nach der Entscheidbarkeit der Menge $S_P = \{w_M \mid P(M)\}$, die eine Untermenge der Menge $\{I\}^*$ ist, und zeigen:

<u>Satz 8</u>: Die Menge S_P ist nicht entscheidbar.
Die Unentscheidbarkeit von S_P beweist man indirekt. Nehmen wir an, S_P wäre entscheidbar, z.B. bezüglich der Symbole e_1 und e_2, d.h. es gäbe eine Turingmaschine E, die mindestens den Strich I in ihrem Alphabet enthält, so daß für alle Gödelnummern w_M gilt:

w_M gehört zu S_P $\Longleftrightarrow$ E über w_M angesetzt, stoppt nach endlich vielen Schritten über e_1

oder:

w_M gehört nicht zu S_P $\Longleftrightarrow$ E über w_M angesetzt, stoppt nach endlich vielen Schritten über e_2.

Der letzte Satz müßte speziell auch für M=E gelten und lautet dann:

w_E gehört nicht zu S_P $\Longleftrightarrow$ E über w_E angesetzt, stoppt nach endlich vielen Schritten über e_2.

$w_E \notin S_P$ heißt aber per definitionem: E über w_E angesetzt, stoppt nicht nach endlich vielen Schritten, so daß wir den Widerspruch haben:

E über w_E angesetzt, st[illegible] $\Longleftrightarrow$ E über w_E angesetzt, [illegible] nicht nach endlich vi[illegible] endlich [illegible] Schritten.

Es kann also keine Entscheidungsprozedur für die Menge S_p geben. Die Unentscheidbarkeit von S_p nennt man die "Unlösbarkeit des Selbstanwendungsproblems für Turingmaschinen".

2.6. Turingmaschinen, von einem übergeordneten Standpunkt aus betrachtet.

Zur Einführung des Begriffs "Turingmaschine" und "berechenbar"(im Sinne Turings etc.) haben wir folgende Schritte ausgeführt:

1. Es wurden mit Hilfe von Zeichen aus einem unendlichen Zeichenvorrat $B_T = A_T \cup N \cup \{a_0, r, l, h\}$ gewisse Ausdrücke gebildet, die wir "Turingtafeln" nannten. Mann kann die Menge der Turingtafeln als eine Sprache $L_T \subset B_T^*$ auffassen. Die Regeln, wie Turingtafeln zu konstruieren sind, bilden die Syntax dieser Sprache.
2. Als Daten wurden k-Tupeln von Wörtern über dem Alphabet A_T zugelassen. Die Menge aller dieser Daten wollen wir X_T nennen.
3. Es wurde ein "Interpreter" für L_T angegeben, das ist eine Funktion

   $$J_T: D_T \to X_T, \text{ wobei } D_T \subset L_T \times X_T,$$

   die für jedes sprachliche Gebilde $M \in L_T$ ("Turingtafeln") und gewisse Daten $d \in X_T$ einen Resultatwert angibt, die also die "Wirkung von M auf d" beschreibt.

   Die Syntax von L_T, die Datenmenge X_T und der Interpreter J_T wurde mit den sprachlichen Mitteln einer nicht näher spezifizierten Mengenlehre bzw. Arithmetik definiert. Diese formalen mathematischen Systeme treten hier also als "Metasprache" für die "Turingsprache" L_T auf.

4. Als berechenbar ([illegible] Turings) wurden nun jene Funktionen $f: (A^*)^k \to$ [illegible] irgend ein endliches Alphabet [illegible]

definiert, für die es eine Turingmaschine $M \in L_T$ gibt, so daß für alle k-Tupel $(w_1,\ldots,w_k) \in (A^*)^k$

$$f(w_1,\ldots,w_k) = J_T\ (M,(w_1,\ldots,w_k))$$

gilt.

5. Mit Hilfe des Begriffs "berechenbare Funktionen" konnten die Begriffe "entscheidbare Menge (Relation)" und "aufzählbare Menge (Relation)" eingeführt werden.

Man könnte nun in Verallgemeinerung dieser Vorgangsweise definieren:

Definition 12: Ein System $P=(B_P,L_P,X_P,J_P)$ heißt Programmiersprache, wenn

$L_P \subset B_P^N$ (B_P heißt Alphabet der Programmiersprache, L_P die Menge der in P formulierbaren Programme)

$X_P \subset B_P^N$ (X_P heißt Menge der in P formulierbaren Daten)

J_P: $D_P \to X_P$, wobei $D_P \subset L_P \times X_P$ (J_P heißt der Interpreter der Programmiersprache P)

Die Turingsche Darstellung der Algorithmen wäre also in dieser Terminologie eine spezielle Programmiersprache. Die hier angegebene Betrachtungsweise wird noch keineswegs allgemein vertreten, doch scheint sie sich durchzusetzen (in einigen Arbeiten von John McCarthy, in den jüngsten Arbeiten des Wiener IBM-Laboratoriums, in dem noch nicht im Handel erhältlichen BI-Taschenbuch von Maurer).

Freilich bleiben hier noch viele Fragen offen. Z.B. wäre es zu überlegen, ob das Attribut "Programmiersprache" nicht auf solche Systeme P eingeschränkt werden sollte, deren L_P, X_P und [illegible] Kriterien erfüllen, die den [illegible] Charak[illegible] ausdrückbaren Verfahren sich[illegible]

Wesentlich erscheint mir jedoch bei dieser Definition die Art der Festlegung des Interpreters, die den "imperativen" Charakter dieser Sprachen gut zum Ausdruck bringt und diese Sprachen klar scheidet von den "deskriptiven" Sprachen der formalen Logik und Mathematik, deren Interpreter von ganz anderem Charakter ist (siehe Modelltheorie!).

Für unsere Zwecke hat die obige Betrachtungsweise wenigstens den methodischen Vorteil, daß nachfolgende Betrachtungen über universelle Turingmaschinen und Äquivalenzbeweise zwischen verschiedenen Algorithmuskonzepten übersichtlich dargelegt werden können.

2.7. Universelle Turingmaschinen

Die Klasse der (im Sinne Turings) berechenbaren Funktionen ist so groß, daß sogar der Interpreter J_T selbst in einem gewissen Sinne berechenbar ist (eine Turingmaschine, die diese Berechnung leistet, nennt man "universelle Turingmaschine").

Eine berechenbare Funktion f ist per definitionem eine Funktion, für die es eine Turingmaschine M_f gibt, so daß für alle Argumenttupel

$$f(w_1,\dots,w_k) = J_T(M_f,(w_1,\dots,w_k)).$$

Es kann nur eine Turingmaschine U konstruiert werden, der man die Gödelnummer w_{M_f} der zu einer beliebigen berechenbaren Funktion f gehörigen Turingmaschine M_f und jedes Argumenttupel $(w_1,\dots,w_k)$ (ebenfalls in verschlüsselter Form als natürliche Zahl $w_{w_1,\dots,w_k}$ in der Darstellung als ein Wort über {I}) eingeben kann und die als Resultat genau $f(w_1,\dots,w_k)$ (in verschlüsselter Form als Zahl $w_{f(w_1,\dots,w_k)}$) [illegible] eine "universelle" Turing[illegible] gilt also:

$$w_{f(w_1,\ldots,w_k)} = J_T(U,(w_{M_f},w_{w_1}\ldots w_k))$$

für alle berechenbare Funktionen f und alle Argumente $(w_1,\ldots,w_k)$.

Beispiele für solche universelle Turingmaschinen können konstruktiv angegeben werden, man vergleiche etwa [Hermes], S. 203. Das Prinzip der Konstruktion universeller Turingmaschinen ist einfach, die effektive Konstruktion jedoch mühsam: U muß einerseits in der Lage sein, aus der Gödelnummer einer Turingmaschine die Struktur der Maschine vollständig zu rekonstruieren (daß dies möglich ist, stellt bestimmte Forderungen an die gewählte Gödelnumerierung!) und andererseits alle Teilfunktionen des Interpreters J_T, wie sie in 2.2 angegeben sind "simulieren". Daß gerade dieses Letztere möglich ist, ist der eigentlich überraschende Punkt und scheint eine Eigenheit aller imperativen Sprachen zu sein, deren Ausdrucksfähigkeit nicht kleiner als die der "Turingsprache" ist. Für eine konkrete Programmiersprache wurde dieser Zug meines Wissens das erste Mal von John McCarthy explizit herausgearbeitet. Er gab den Interpreter seiner Programmiersprache LISP in einer Metasprache, die selbst nur Ausdrucksmittel von LISP enthält.

3. DEFINITION DER GRUNDBEGRIFFE MIT HILFE ANDERER KONZEPTE

3.1. Übersicht über die anderen Begriffsbildungen

Außer der Möglichkeit, die Turing zur Definition der Grundbegriffe der Algorithmentheorie 1937 vorgeschlagen hat, wurden seit ca. 1950 eine ganze Reihe anderer Möglichkeiten angegeben.

Die eine Gruppe von Begriffsbildungen kann man (mehr oder weniger gekünstelt) genau nach dem Schema, das wir in 2.6. herausgearbeitet haben (dem "Programmiersprachenschema") betrachten, nur daß eben die Mengen B_P, L_P, X_P und der Interpreter J_P jedesmal anders definiert werden. Zu diesen Begriffsbildungen gehört z.B. die "Sprache" der μ-rekursiven Funktionen (Kleene 1936) und die "Sprache" der normalen Algorithmen (Markow 1954). Man kann dazu auch die vielen speziellen Programmiersprachen für die Computeranwendungen rechnen (ALGOL, PL/1 usw.).

Eine andere Gruppe von Begriffsbildungen läuft nach dem folgenden Schema (dem "Kalkülschema"): man gibt wieder durch eine gewisse Syntax eine Menge sprachlicher Ausdrücke L_K über einem Alphabet B_K an, die als "Axiomensysteme" (entspricht dem Begriff "Programm") verwendet werden. Durch ein "Kalkül" J_K (der in etwa die Stelle des "Interpreters" einnimmt) wird definiert, wie jedes Axiomensystem $k \in L_K$ eine Menge S_k von Elementen einer "Datenmenge" X_K ausscheidet (so wie jedes Programm einer Programmiersprache L_P durch den Interpreter eine Funktion bestimmt). Die durch ein $k \in L_K$ festgelegten Mengen S_k nennt man dann "aufzählbar (formal repräsentierbar usw.) im Sinne der Sprache L_K". Die Begriffe "berechenbar" und "entscheidbar" lassen sich dann einführen, indem man die Sätze 2 und 3 von 2.5. als D[illegible] liest.

Bei den "Programmiersprachen" ist "berechenbar" Grundbegriff, "entscheidbar" und "aufzählbar" davon ableitbar, bei den "Kalkülsprachen" "aufzählbar" Grundbegriff und "entscheidbar" und "berechenbar" abgeleitet.

"Kalkülsprachen" sind z.B. der "Gleichungskalkül", der die rekursiven Funktionen bestimmt (Herbrand, Gödel, Kleene ab 1931), der Lambda-K-Kalkül (Church 1936), die "elementary formal systems"(Smullyan 1961, eine Abwandlung der "canonical systems" von Post, 1943), die "minimal logic" (Fitch 1953).

Die verschiedenen Ansätze hatten alle dasselbe Ziel: die Begriffe "Algorithmus", "berechenbar" etc. innerhalb eines mathematischen Systems streng zu definieren. Sie wurden z.T. gänzlich voneinander unabhängig entwickelt. Trotzdem haben sie sich - und das ist eins der interessantesten Resultate der Algorithmentheorie - als untereinander äquivalent erwiesen, in dem Sinne, daß die Klasse der "berechenbaren Funktionen" bei den verschiedenen Ansätzen jeweils dieselbe ist. Das läßt die Vermutung zu, daß der Begriff der "Berechenbarkeit" (und die verwandten Grundbegriffe "aufzählbar" etc.) tatsächlich ein sehr wesentliches Konzept trifft, das mit unserer intuitiven Vorstellung von "berechenbar" in etwa übereinstimmt.

Die berühmteste Fassung hat diese Vermutung durch die bekannte "CHURCH-sche These" im Jahre 1936 erhalten, in der Church die Ansicht vertritt, daß sowohl die Klasse der rekursiven Funktionen, als auch die Klasse der lambda-definierbaren Funktionen zusammenfällt mit der Klasse der im intuitiven Sinne "berechenbaren" Funktionen. Diese These müßte genauer lauten: Man wird auch in Zukunft keine (in irgendeinem System der formalen Logik beschreibbaren) Funktionen außer den rekursiven Funktionen erhalten, [illegible] auch im [illegible] Sinne berechenbar (mechanisch [illegible]bar, durch ei[illegible] [illegible]er geartete Rechenmaschine [illegible] oder so ähnlich) nennen würde.

Wir geben als Beispiel für einen Äquivalenzbeweis die Beweisskizze für die Begriffe "Berechenbar im Sinne Turings" und "μ-rekursiv".

3.2. μ-rekursive Funktionen

Definition 13: Eine Funktion $f: N^k \to N$ heißt μ-rekursiv, wenn sie ausgehend von den Basisfunktionen s, u_n^i und c_o^o durch endlich oftmalige Anwendung der folgenden Definitionsschemata gewonnen werden kann:

1. Substitution
2. Rekursion (Induktion)
3. Anwendung des μ-Operators auf reguläre Funktionen

Dabei sei:

$s: N \to N$, $s(x) = x+1$ (Nachfolgefunktion)

$u_n^i: N^n \to N$, $u_n^i(x_1, \ldots, x_n) = x_i$ (Identitätsfunktionen)

$c_o^o = 0$ (die "nullstellige" konstante Funktion o)

$\bar{x}^{(n)}$ bezeichne im Folgenden ein n-Tupel $(x_1, \ldots, x_n) \in N^n$.
Man definiert:

Substitution: seien $h_1, \ldots, h_r: N^n \to N$ und $g: N^r \to N$ Funktionen, dann heißt $f: N^n \to N$ durch Substitution von $h_1, \ldots, h_r$ in g gewonnen, wenn

$$f(\bar{x}^{(n)}) = g(h_1(\bar{x}^{(n)}), \ldots, h_r(\bar{x}^{(n)})) \text{ für alle } \bar{x}^{(n)}.$$

Rekursion: seien $g: N^n \to N$ und $h: N^{n+2} \to N$ Funktionen, dann heißt $f: N^{n+1} \to N$ durch Rekursion (induktiv) mit Hilfe der Funktionen g und h definiert, wenn:

$$f(\bar{x}^{(n)}, o) = g(\bar{x}^{(n)}) \quad \text{für alle } \bar{x}^{(n)}$$

$$f(\bar{x}^{(n)}, s(y)) = h(\bar{x}^{(n)}, y, f(\bar{x}^{(n)}, y)) \quad \text{für alle } y \in N \text{ und alle } \bar{x}^{(n)}$$

reguläre Funktionen: eine Funktion $g: N^{n+1} \rightarrow N$ heißt regulär, wenn für alle $\bar{x}^{(n)}$ ein y existiert, so daß $g(\bar{x}^{(n)}, y) = o$.

μ-Operator: die Anwendung des μ-Operators auf reguläre Funktionen $g: N^{n+1} \rightarrow N$ liefert eine Funktion $f: N^n \rightarrow N$ und ist wie folgt definiert:

$$f(\bar{x}^{(n)}) = \mu\, g(\bar{x}^{(n)}, y) = \text{das kleinste } y, \text{ so daß } g(\bar{x}^{(n)}, y) = o.$$

3.3. <u>Äquivalenz der Begriffe "berechenbar im Sinne Turings" und μ-rekursiv.</u>

Wir wollen zeigen:

<u>Satz 5</u>: (a) Jede μ-rekursive Funktion ist berechenbar im Sinne Turings.

(b) Jede im Sinne Turings berechenbare Funktion ist μ-rekursiv.

Beweisskizze:

ad (a): Der Beweis für (a) geht im Wesentlichen so vor sich daß zunächst für die Funktionen s, u_n^i und c_o^o konkrete Beispiele von Turingmaschinen angegeben werden, deren zugehörige Funktionen gerade s, u_n^i und c_o^o sind. Dann werden für die drei Vorgänge der Substitution, Induktion und Anwendung des μ-Operators "konstruktive" Verfahren angegeben, mit deren Hilfe man z.B. für die durch Substitution erhaltene Funktion f eine Turingmaschine bekommen kann, die diese Funktion berechnet, wenn man für die Ausgangsfunktionen $g, h_1, \ldots, h_n$ Turingmaschinen kennt, die sie berechnen. Damit hat man ein Verfahren in der Hand, wie man zu jeder μ-rekursiven Funktion f ausgehend von ihrer Darstellung durch die Basisfunktionen s, u_n^i und c_o^o eine Turingmaschine effektiv konstruieren kann, die f berechnet.

[illegible] Beweis von (b) geht ähnl[illegible] Beweis der [illegible] einer universellen Turingma[illegible] darin bestand, daß man die Berechenbarkeit des Interpreters J_T nachwies. Jetzt zeigt man, daß dieser Interpreter im Wesentlichen auch μ-rekursiv ist, m.a.W. man konstruiert eine μ-rekursive Funktion ψ, der man die Gödelnummer w_{M_f} der Turingmaschine, die zu einer beliebigen berechenbaren Funktion f gehört, und ein Argumenttupel $(w_1,\ldots,w_k)$ von f (ebenfalls in verschlüsselter Form) als Argumente vorlegen kann und die als Resultat genau $f(w_1,\ldots,w_k)$ (ebenfalls in verschlüsselter Form) liefert. Jede berechenbare Funktion läßt sich also im Wesentlichen (d.h. wenn man von der notwendigen Verschlüsselung der Argumente und Resultate absieht) durch die μ-rekursive Funktion ψ darstellen.

Die für (a) und (b) verwandten Beweismethoden, nämlich die Angabe eines Verfahrens ("Übersetzers", "Compilers"), der zu jedem Ausdrucksmittel der einen Sprache ("μ-rekursiven Funktion") ein Ausdrucksmittel der anderen Sprache (Turingmaschine) angibt, das "dasselbe" leistet, und die Konstruktion eines "Interpreters" für die eine Sprache mit den Ausdrucksmitteln der anderen, scheinen zwei wesentliche Beweismethoden für Äquivalenzbeweise von "imperativen" Sprachen zu sein.

4. RESULTATE DER ALGORITHMENTHEORIE IN VERSCHIEDENEN ANWENDUNGSGEBIETEN

4.1. Beweismethoden

Unentscheidbarkeitsbeweise wurden zunächst für Eigenschaften (d.h. für die durch diese Eigenschaften definierten Mengen) erbracht, die im Zusammenhang mit den Grundbegriffen der Algorithmentheorie definiert werden können. Ein Beispiel wurde in 2.5. angegeben. Für die Beweise ist das "Diagonalverfahren" typisch, das in seiner Grundstruktur aus der Beweisskizze in 2.5. abgelesen werden kann.

Andere Beweise geschehen oft durch Zurückführen des gegebenen Entscheidungsproblems auf ein anderes in der folgenden Art: Die Menge S sei als unentscheidbar zu beweisen. Man zeigt: aus einem Entscheidungsverfahren für S könnte man eines für S' konstruieren, wobei S' eine Menge sein muß, deren Unentscheidbarkeit bereits bewiesen wurde. So kann man z.B. zeigen:

Satz 6: Es ist nicht entscheidbar, ob eine beliebige Turingmaschine, die auf das leere Band angesetzt wird, nach endlich vielen Schritten stoppen wird (Halting Problem).
und:
Satz 7: Es ist nicht entscheidbar, ob eine universelle Turingmaschine, auf ein beliebiges Wort angesetzt, nach endlich vielen Schritten stehen bleibt.

4.2. Beispiel aus der Gruppentheorie

Definition 15: Sei $S = \{S_1, \ldots, S_n\}$ ein endliches Alphabet. In die Menge der Wörter über S sei auch ein Element □ aufgenommen (das "leere" Wort), das durch $w\square = \square w = w$ für alle Wörter w def[illegible] ist. Ein Gruppensystem G über [illegible] durch eine [illegible]s des Alphabets S auf S [illegible]

Abbildung $s: S \xrightarrow{\text{auf}} S$ mit $s(s(S_j)) = S_j$ für alle S_j) und eine endliche Menge geordneter Paare (D_i, D_i') (D_i, D_i' seien Wörter über S, $i=1,\dots,m$), die "definierenden Relationen" gegeben, wobei noch gelten muß:

(1) mit (D_i, D_i') ist auch (D_i, D_i') in der Menge der definierenden Relationen

(2) alle $(S_j, s(S_j))$ ($j=1,\dots,n$) sind in der Menge der definierenden Relationen.

<u>Definition 15</u>: W' heißt unmittelbares Folgewort von W in Bezug auf das Gruppensystem G ($W \text{ --]}\ W'$) wenn es ein i und zwei Wörter U,V gibt, so daß:

$$W = UD_iV \text{ und } W' = UD_i'V.$$

<u>Definition 16</u>: W' heißt Folgewort von W in Bezug auf das Gruppensystem G ($W \overset{*}{\text{--]}} W'$), wenn es eine endliche Kette von Wörtern $W_o, W_1, \dots, W_p$ gibt, so daß

$$W = W_o,\ W_o \text{ --]}\ W_1,\ W_1 \text{ --]}\ W_2, \dots,\ W_{p-1} \text{ --]}\ W_p,\ W_p = W'.$$

(--] ist eine Kongruenzrelation bezüglich der Verknüpfung "Aneinanderreihen von Wörtern" in der Menge der Wörter über S. Die Restklassenalgebra bezüglich dieser Relation ist eine Gruppe).

Das <u>spezielle</u> <u>Wortproblem</u> für ein Gruppensystem G ist die Frage, ob ein Algorithmus existiert (also z.B. eine Turingmaschine), der für zwei beliebige Wörter W und W' über S in endlich vielen Schritten entscheidet, ob $W \overset{*}{\text{--]}} W'$ gilt oder nicht. Das allgemeine Wortproblem für Gruppensysteme ist die Aufgabe, einen Algorithmus zu finden, der für ein beliebiges Gruppensystem G und zwei beliebige Wörter W und W' über dem Alphabet von G in endliche vielen Schritten entscheidet, ob $W \overset{*}{\text{--]}} W'$ gilt oder nicht.

<u>Satz 8</u>: [illegible] eine Wortproblem für Gruppensysteme ist unlösbar[illegible] Beispiele von Gruppensystemen [illegible] speziel[illegible] unlösbar ist.

4.3. Beispiele aus der formalen Logik

Prädikatenlogik erster Stufe:
Die Prädikatenlogik erster Stufe ist eine Sprache über dem Alphabet:

a,b,c,... (Individuenkonstante)
x,y,z,... (Individuenvariable)
A,B,C,... (Aussagenvariable)
$\wedge, \vee, \neg, \rightarrow$ (logische Konnektoren)
$\forall, \exists$ (Quantoren).

Über diesem Alphabet lassen sich alle möglichen Wörter bilden. Nur gewisse werden aber mittels eines speziellen Kalküls (Syntax) als "Formeln" ausgezeichnet, z.B. ist

$(\exists x)(P(x) \rightarrow (\forall z)Q(z,x))$ eine Formel

und $\exists\forall xx \rightarrow pq$ keine Formel.

Das charakteristische syntaktische Kennzeichen der Prädikatenlogik erster Stufe sind die beiden Quantoren $\exists, \forall$, die nur auf Individuenvariable angewendet werden.

Jede Formel läßt sich in einem gegebenen "Gegenstandsbereich" "interpretieren" (d.h. allen vorkommenden Individuenkonstanten werden Individuen des Gegenstandsbereiches, allen Individuenvariablen sämtliche Elemente des Gegenstandsbereiches als Laufbereich, allen Prädikatenkonstanten Relationen über dem Gegenstandsbereich zugeordnet; dies ist eine Art der Interpretation, wie sie typisch ist für "deskriptive" Sprachen!). Eine Formel kann dabei in einen "wahren" oder "falschen" Satz übergehen. Geht eine Formel bei jeder Interpretation in einen wahren Satz über, so heißt sie "allgemeingültig" (exakt definieren lassen sich all diese Begriffe wie "Interpretation", "wahr", "falsch" usw. nur in einer geeigneten, genügend starken Metasprache).

Die Menge P der allgemeingültigen Formeln der Prädikatenlogik erster Stufe ist eine Untermenge der Menge aller Wörter über dem angegebenen Alphabet. [illegible]

Ist P_1 aufzählbar und entscheidbar?

Es ist der Inhalt des berühmten Vollständigkeitssatzes von Gödel (1930), daß die Menge P_1 aufzählbar ist, d.h. daß alle "allgemeingültigen" Formeln durch einen einzigen Algorithmus (z.B. eine Turingmaschine) erzeugt werden können. Es gibt mehrere konkrete Beispiele von Algorithmen ("Kalkülen"), die diese Erzeugung leisten.

Church bewies im Jahre 1936, daß P_1 jedoch nicht entscheidbar ist, d.h. daß es keinen allgemeinen Algorithmus gibt, der von einer beliebigen vorgelegten Formel in endlich vielen Schritten entscheiden könnte, ob sie allgemeingültig ist.

<u>Prädikatenlogik höherer Stufen:</u>

Bei einer Erweiterung der Ausdrucksmöglichkeiten der Sprache, die man Prädikatenlogik erster Stufe nennt, kann jedoch die Aufzählbarkeit der Menge aller allgemeingültigen Formeln verloren gehen.

Die Prädikatenlogik zweiter Stufe z.B. ist nicht mehr "vollständig" (d.h. es gibt keinen Algorithmus, der alle allgemeingültigen Formeln dieser Sprache liefern könnte). Das syntaktische Hauptkennzeichen dieser Sprache ist, daß es auch Prädikatenvariable gibt und sich die Quantoren $\exists, \forall$ auch auf Prädikatenvariable beziehen können. Die Unvollständigkeit der Prädikatenlogik zweiter Stufe wurde zuerst von Gödel (1931) gezeigt.

4.4. <u>Beispiele aus der Syntax formaler Sprachen.</u>

Die Syntax einer Sprache L scheidet aus der Menge aller Wörter über einem Alphabet jene Wörter aus, die zu L gehören sollen. Es gibt viele Arten, eine Syntax anzugeben. Das Studium [illegible]tax von Sprachen und der zug[illegible]-mengen [illegible] im Zusammenhang mit den Pro[illegible] besonders vorangetrieben worden. Dabei stellen sich viele Aussagen über die Aufzählbarkeit und Entscheidbarkeit der betrachteten Wortmengen ein. Eine bekannte Klasse von Sprachen sind die sogenannten kontextfreien, die durch "kontextfreie" Grammatiken erzeugt werden, die im wesentlichen dasselbe leisten wie die "Backusnotation" bei der formalen Definition von ALGOL. Über diese Klasse von Sprachen hat man z.B. u.a. die folgenden Entscheidbarkeitsresultate:

<u>Satz 9:</u> Seien G_1 und G_2 kontextfreie Grammatiken, die die kontextfreien Sprachen $L(G_1)$ und $L(G_2)$ erzeugen, dann gilt:

1. $L(G_1)$ ist eine rekursive Menge
2. Es ist unentscheidbar, ob $L(G_1) = L(G_2)$
3. Es ist unentscheidbar, ob $L(G_1)$ = Menge aller Wörter über dem jeweiligen Alphabet
4. Es ist unentscheidbar, ob $L(G_1) \cap L(G_2) = \emptyset$.

LITERATUR

Lehrbücher:

KLEENE, S.C.: Introduction to Metamathematics, North-Holland 1952

DAVIS, M.: Computability and Unsolvability, McGraw Hill, 1958.

HERMES, H.: Aufzählbarkeit, Entscheidbarkeit, Berechenbarkeit, Springer 1960. ((Hermes) bezieht sich auf die engl. Übersetzung dieses Buches, Springer 1965).

BRAUER/INDERMARK: Algorithmen, rekursive Funktionen und formale Sprachen, BI-Hochschulskripten 817.

HOTZ / WALTER: Automatentheorie und formale Sprachen, BI-Hochschulskripten 821/821a.

Originalarbeiten zu den einzelnen Algorithmenbegriffen:

Turingmaschinen: A.M. Turing: On Computabel Numbers with an Application to the Entscheidungsproblem. Proc. London math. Society (2), 42, 230-265 (1937) und (2), 43, 544-546 (1937).

primitiv rekursive Funktionen:

K. Gödel: Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I, Mh. Math. Physik 38, 173-198 (1931).

Hilbert / Bernays: Grundlagen der Mathematik I, Springer, 1934 (Neuauflage 1968).

S.C. Kleene: General Recusive Functions of Natural Numbers, Math. Ann 112, 727-742 (1936).

R. Peter: Rekursive Funktionen, Verl. ungar. Akademie der Wissenschaften 1955

μ-rekursive Funktionen:

die Arbeit von Kleene bei "primitiv rek. Funktionen"

rekursive Funktionen:

J. Herbrand: Sur la non-contradiction de l'Arithmétique, J. reine angew. Math., 166 1-8 (1931).

K. Gödel: On Undecidable Propositions of Formal Mathematical Systems, Inst. Advanced Study, Princeton, 1934.

die Arbeit von Kleene bei "primitiv rek. Funktionen"

lambda-Definierbarkeit:

A. Church: The Calculi of Lambda-Conversions, Princeton Univ. Press, 1941.

canonical calculi:

E.Post: Formal Reductions of the General Combinatorial Decision Problem, Am.J.Math., 65, 197-215 (1943).

elementary fomal systems:

R.Smullyan: Theory of Formal Systems, Princeton Univ.Press, 1961.

minimal logic:

F.B.Fitch: A Simplification of Basic Logic, J.symb. Logic, 18,317-325 (1953).

normale Algorithmen:

A.A.Markow: Teorija algorifmow, Akad.Nauk.UdSSR, Mathem.Inst.Trudy 42, 1954 (engl.Übersetzung: Theory of Algorithms, Israel Program for Scient. Transl., 1961).

Zugang zu anderen in der russischen Literatur entwickelten Algorithmenbegriffen erhält man durch das Buch von Thiele:

H.Thiele: Wissenschaftstheoretische Untersuchungen in algorithmischen Sprachen I, Deutscher V.Wiss., 1966, das reichlich Literaturhinweise auf Originalarbeiten enthält.

Außerdem sei auf die Artikel in der Serie "Probleme der Kybernetik", Akademie Verlag Berlin, hingewiesen, in der laufend Artikel russischer Autoren erscheinen. (Diese Serie ist eine Übersetzung der Serie gleichen Titels in Russisch, erscheint bei FISMATGIS, Moskau. Die deutsche Ausgabe kommt mit ca. 3-4 Jahren Verspätung).

Darstellungen von Programmiersprachen, die als Grundlage für die Definition der Grundbegriffe der Algorithmentheorie genommen werden könnten, sind:

P.Lauer: Formal Definition of ALGOL 60, IBM Lab. Wien TR 25.088

K.Walk u.a.: Abstract Syntax and Interpretation of PL/1, IBM Lab. Wien, TR 25.082

J.McCarthy, LISP 1.5 Programmer's Manual, MIT Press, 1962