3 Grenzen der Algorithmisierung

In Kapitel 2 haben wir einen Algorithmus zu einem gegebenen Problem entwickelt, immer unter der naiven Annahme, es werde schon einen geben. In diesem Kapitel wollen wir die Begriffe weiter präzisieren und darauf aufbauend einige theoretische Ergebnisse über die Grenzen der Algorithmisierbarkeit ableiten.
Wir interessieren uns dafür, zur Lösung welcher Probleme man Computer überhaupt einsetzen kann und welche Probleme *beweisbar* nicht mit Computern gelöst werden können. Hierzu stellen wir in Abschnitt 3.1 einige mathematische Grundbegriffe zusammen, um die angekündigten Präzisierungen vornehmen zu können. In den Abschnitten 3.2 und 3.3 beweisen wir Sätze über die Existenz und Nicht-Existenz von Algorithmen. Warum man von der algorithmischen Unlösbarkeit eines Problems auf seine maschinelle Unlösbarkeit schließen darf, erläutern wir in Abschnitt 3.4.

3.1 Mathematische Grundbegriffe

Die folgenden Grundbegriffe finden Sie auch in Vorlesungen über Mathematik. Grundlegend für präzise mathematische Formulierungen ist die Mengenlehre, von der wir hier nur den begrifflichen Teil benötigen.

Eine **Menge** ist eine Zusammenfassung von wohlunterscheidbaren Objekten zu einem ganzen. Die Objekte der Menge nennt man **Elemente**. Wenn ein Objekt x Element einer Menge M ist, so schreibt man: $x \in M$. Gehört x nicht zu M, schreibt man: $x \notin M$. Wenn eine Menge endlich viele Elemente $x_1,x_2,...,x_n$ besitzt, so kann man sie explizit durch Auflisten dieser Elemente innerhalb geschweifter Klammern definieren:

$$M=\{x_1,x_2,...,x_n\}.$$

Zum *Beispiel* lautet die Menge SF der Spielkartenfarben:

$$SF=\{Kreuz,Pik,Herz,Karo\}$$

und die Menge EQ der ersten 10 Quadratzahlen:

$$EQ=\{1,4,9,16,25,36,49,64,81,100\}.$$

Die Reihenfolge, in der die Elemente aufgelistet werden, ist bei Mengen unwichtig. Man beachte, daß die Elemente "wohlunterscheidbar" sind. Listet man ein Element mehrfach auf, so ist es trotzdem nur einmal in der Menge enthalten. Es gilt z. B.:

$$\{1,4,9\}=\{1,4,1,4,4,9,1\}$$

Eine spezielle Menge ist die Menge, die kein Element besitzt. Man nennt sie die **leere Menge** und bezeichnet sie mit $\varnothing$.

Meist werden Mengen nicht explizit durch Auflisten ihrer Elemente, sondern implizit durch Angabe von Eigenschaften ihrer Elemente definiert. Schreibweise:

$\{f(x) \mid \text{Eigenschaften von } x\}$.

Hierbei ist f eine Funktion oder Vorschrift; oft ist f die Identität.

Beispiele: $EQ=\{x \mid x \text{ ist Quadratzahl}, 1 \le x \le 100\}$ oder

$EQ=\{x^2 \mid 1 \le x \le 10\}$.

Eine Menge M ist in einer Menge M' enthalten, wenn jedes Element aus M auch in M' liegt. Man schreibt dann: $M \subseteq M'$ oder $M' \supseteq M$ und nennt M **Teilmenge** von M', bzw. M' **Obermenge** von M. Zwei Mengen sind *gleich* (M=M'), wenn sie wechselweise ineinander enthalten sind:

$M=M' \Leftrightarrow (M \subseteq M' \text{ und } M' \subseteq M)$.

M und M' sind verschieden, wenn nicht M=M' gilt. Im Zeichen: $M \neq M'$. M heißt *echte Teilmenge* von M', wenn $M \subseteq M'$ und $M \neq M'$ gilt.

Zum *Beispiel* ist {1,4,16} echte Teilmenge von EQ, und die beiden Mengen

{x | x ist Primzahl, und x ist durch 3 teilbar}

und {3} sind gleich.

Übliche Operationen auf Mengen sind die **Vereinigung** "$\cup$", der **Durchschnitt** "$\cap$" und die **Differenz** "\":

$M \cup M' = \{x \mid x \in M \text{ oder } x \in M'\}$,

$M \cap M' = \{x \mid x \in M \text{ und } x \in M'\}$,

$M \setminus M' = \{x \mid x \in M \text{ und } x \notin M'\}$.

Wenn M eine fest gewählte Menge ist, dann nennt man $M \setminus M'$ auch das **Komplement** von M' (bzgl. M).

Besonders häufig auftretende Mengen haben spezielle Zeichen erhalten, z.B. die Menge der natürlichen Zahlen IN, die Menge der ganzen Zahlen Z, die Menge der rationalen Zahlen Q und die Menge der reellen Zahlen IR. In der Mathematik beginnen die natürlichen Zahlen in der Regel mit der Zahl 1; in der Informatik ist es jedoch nützlich, die Zahl 0 ebenfalls als natürliche Zahl aufzufassen, d.h., in dieser Vorlesung gilt stets: $IN=\{0,1,2,3,...\}$.

Weitere Operationen auf Mengen sind das kartesische Produkt, die Bildung aller endlichen Folgen (*-Operator) und die Bildung der Potenzmenge.

Das **kartesische Produkt** $M_1 \times M_2$ ist die Menge aller geordneten Paare:

$M_1 \times M_2 = \{(x,y) \mid x \in M_1 \text{ und } y \in M_2\}$.

Die Menge der geordneten n-Tupel ergibt sich als n-faches kartesisches Produkt:

$$M_1 \times M_2 \times ... \times M_n = \{(x_1, x_2, ..., x_n) \mid x_i \in M_i \text{ für } i=1,...,n\}.$$

Sind alle Mengen M_i gleich einer Menge M, so schreibt man hierfür M^n.

Die Menge M^* der endlichen Folgen über einer Menge M erhält man als Menge aller geordneten n-Tupel über M, für alle $n \in \mathbb{N}$, d.h.:

$$M^* = M^0 \cup M^1 \cup M^2 \cup M^3 \cup ...$$

$$= \bigcup_{n \geq 0} M^n$$

Dabei bedeutet

$$\bigcup_{n \geq 0}$$

daß man die unendliche Vereinigung für alle $n \geq 0$ durchführen soll. Die unendliche Vereinigung von Mengen ist wieder eine Menge. Unklar ist hier noch, was M^0 und M^1 bedeuten sollen. Es ist $M^1 = M$ und $M^0 = \{()\}$. Die Menge M^0 ist also nicht die leere Menge, sondern sie enthält ein Element, und zwar das "leere Tupel", im Zeichen: ().

Die Elemente von M^* schreibt man nicht mit Klammern und Kommata als Tupel, sondern man fügt die Elemente aus M einfach zu Folgen aneinander:

$$M^* = \{x_1 x_2 ... x_n \mid n \in \mathbb{N}, x_i \in M \text{ für } i=1,...,n\}$$

Für n=0 ist hierin auch die leere Folge enthalten, die wir stets mit ε bezeichnen. Die Elemente von M^* nennt man **Wörter** über der Menge M, ε heißt daher auch das **leere Wort**. Wenn $w = x_1 x_2 ... x_n$ (mit $n \in \mathbb{N}$ und $x_i \in M$ für i=1,...,n) ein Wort ist, dann heißt $|w| = n$ die **Länge** von w. Speziell ist $|\varepsilon| = 0$. Auf M^* ist eine Operation "$\cdot$" (**Konkatenation**) definiert, die aus zwei Wörtern $u, v \in M^*$ ein neues Wort $u \cdot v = uv$ durch Aneinanderreihung bildet. Für diese Operation gilt:

$(u \cdot v) \cdot w = u \cdot (v \cdot w)$ für alle $u, v, w \in M^*$,

$u \cdot \varepsilon = \varepsilon \cdot u = u$ für alle $u \in M^*$.

Die Menge M^* zusammen mit dem leeren Wort ε und der Operation "$\cdot$" bildet somit ein Monoid (= Halbgruppe mit Einselement); näheres siehe Vorlesungen in Mathematik oder Theoretische Informatik. Die Menge M^* heißt oft auch **freies Monoid über M**. Man beachte, daß für die Funktion "Länge" gilt: $|u \cdot v| = |u| + |v|$.

Die **Potenzmenge** 2^M ist die Menge aller Teilmengen von M:

$$2^M = \{K \mid K \subseteq M\}.$$

Die Potenzmenge ist niemals leer, da stets $M \in 2^M$ gilt. Speziell ist:

$2^{\emptyset}=\{\emptyset\}$,

$2^{2^{\emptyset}}=2^{\{\emptyset\}}=\{\emptyset,\{\emptyset\}\}$.

Man beachte, daß für alle Mengen M (also auch für die leere Menge $\emptyset$) die Mengen M und {M} verschieden sind. Insbesondere ist {M} eine einelementige Menge, die nur das Element M besitzt.

Mit Hilfe der Begriffe der Mengenlehre kann man auch Funktionen und Relationen leicht einführen. Eine zweistellige **Relation** R über einer Menge M ist eine Teilmenge von $M\times M$. Zum *Beispiel* ist die Gleichheit auf einer Menge M folgende Relation $G\subseteq M\times M$ mit

$G=\{(x,x) \mid x\in M\}$,

und man schreibt x=y, falls $(x,y)\in G$ ist.

Eine (partielle) **Funktion** f von der Menge M in die Menge K ist eine Teilmenge von $M\times K$ mit der Eigenschaft, daß es zu jedem $x\in M$ höchstens ein $y\in K$ mit $(x,y)\in f$ gibt (diese Eigenschaft nennt man auch "rechtseindeutig"). Statt $f\subseteq M\times K$ schreibt man bei Funktionen $f: M\to K$, womit ausgedrückt wird, daß jedem Element x aus M höchstens ein Element $y\in K$ zugeordnet ist; für dieses Element y schreibt man f(x) und nennt f(x) den Funktionswert für das Argument x.

Eine Funktion $f: M\to K$ heißt **total**, wenn es zu jedem $x\in M$ ein $y\in K$ mit y=f(x) gibt. Anderenfalls heißt f **partiell**. Eine totale Funktion $f: M\to K$ heißt

- **injektiv**, wenn aus $f(x_1)=f(x_2)$ stets $x_1=x_2$ folgt,
- **surjektiv**, wenn es zu jedem $y\in K$ mindestens ein $x\in M$ mit f(x)=y gibt
- **bijektiv**, wenn f injektiv und surjektiv ist.

Zu einer bijektiven Funktion $f: M\to K$ gibt es stets eine **Umkehrabbildung** $f^{-1}: K\to M$, die ebenfalls bijektiv ist. Setze:

$f^{-1}(y)=x \Leftrightarrow f(x)=y$.

Da f injektiv gibt, kann es zu jedem y höchstens ein x mit f(x)=y geben, und da f auch surjektiv ist, muß es zu jedem y mindestens ein x mit f(x)=y geben; folglich führt die Definition von f^{-1} zu einer bijektiven Funktion.

Wenn $f: M\to K$ bijektiv ist, dann besitzen M und K im intuitiven Sinne gleich viele Elemente. Man sagt: M und K sind *gleichmächtig*. Unter der **Mächtigkeit** |M| von M versteht man die Anzahl der Elemente von M. Für endliche Mengen M bedeutet dies:

$|M|=n \Leftrightarrow$ Es gibt eine bijektive Funktion $f: M\to\{0,1,2,...,n-1\}$.

Zwei Mengen M und K heißen **isomorph**, wenn es eine bijektive Abbildung zwischen M und K gibt.

Eine Menge M heißt **abzählbar** unendlich, wenn sie gleichmächtig zur Menge der natürlichen Zahlen ist. Wir werden in Abschnitt 3.2. sehen, daß Z und Q abzählbar sind; IR dagegen nicht.

Funktionen kann man zu einer Menge zusammenfassen:

$$K^M=\{f \mid f: M\to K, f \text{ total}\}.$$

Eine Menge K kann man dann auch als eine solche Menge K^M mit $M=\{\emptyset\}$ auffassen. Hierzu betrachte man die Funktion $g: K\to K^{\{\emptyset\}}$, die definiert ist durch

$$g(k)=f_k, \text{ wobei für } f_k: \{\emptyset\}\to K \text{ gilt: } f_k(\emptyset)=k.$$

Dann sieht man leicht: g ist bijektiv. Hier erkennt man schon den immanenten Zusammenhang zwischen Mengen, speziell Datenmengen, und Funktionen. Diese Beziehung wird später von großer Bedeutung sein.

Für eine (partielle) Funktion $f: M\to K$ bezeichnen

$$D(f)=\{x\in M \mid f(x) \text{ existiert}\} \text{ und}$$
$$R(f)=\{y\in K \mid \text{Es existiert ein } x\in M \text{ mit } f(x)=y\}$$

den *Definitions*-, bzw. *Wertebereich* von f. Diese Mengen haben in der Literatur sehr verschiedene Namen, z.B. heißt der Definitionsbereich auch *Domain* oder *Quellbereich* von f, und der Wertebereich wird auch auch *Range* oder *Zielbereich* von f genannt. $M\to K$ heißt *Funktionalität* von f oder – falls M und K (Daten-)Typen sind – *Typ* von f. Eine Funktion $f: M\to K$ ist also genau dann total, wenn D(f)=M ist, und f ist surjektiv, wenn D(f)=M und R(f)=K gelten.

In der Informatik vermeidet man die Verwendung partieller Funktionen meist, um eine einheitlichere Darstellung zu erhalten. Hierzu erweitert man Quell- und Zielbereich um das sog. *bottom-Element* $\perp$ und definiert

$$A^\perp:=A\cup\{\perp\}, B^\perp:=B\cup\{\perp\}.$$

$A^\perp$ und $B^\perp$ heißen *Vervollständigungen* von A bzw. B. f erweitert man nun zu

$$f^\perp: A^\perp\to B^\perp \text{ mit}$$
$$f^\perp(a)=\begin{cases} f(a), & \text{falls } f(a) \text{ definiert,} \\ \perp, & \text{sonst.}\end{cases}$$

Bei Funktionen, die durch Algorithmen dargestellt werden, symbolisiert das Zeichen $\perp$ die Pseudoausgabe eines Algorithmus, der für die gewählte Eingabe nicht terminiert.

Wahlfreiheiten besitzt man, wenn A das kartesische Produkt von Mengen ist:

$$A=A_1\times A_2\times\ldots\times A_n.$$

Erweitert man nun die A_i um das bottom-Element, so ist die Frage, welchen Wert ein Ausdruck

$f(a_1,...,a_n)$

besitzt, in dem einige $a_i=\perp$ und einige $a_i\neq\perp$ sind. Meist entscheidet man sich hier für eine *strikte* Fortsetzung von f zu $f^{\perp}$, d.h. man setzt

$f^{\perp}: A_1\times A_2\times...\times A_n\rightarrow B$ mit

$$f^{\perp}(a_1,...,a_n)=\begin{cases} f(a_1,...,a_n), & \text{falls } a_i\neq\perp \text{ für } 1\leq i\leq n,\\ \perp, & \text{falls } a_i=\perp \text{ für ein } i\in\{1,...,n\}.\end{cases}$$

$f^{\perp}$ ist also undefiniert, wenn auch nur eines seiner Argumente undefiniert ist. Wir werden später Abschwächungen dieser Striktheit kennenlernen und Funktionen auch dann noch sinnvoll auswerten können, wenn einige ihrer Argumente undefiniert, also gleich $\perp$ sind. Für den Rest der Vorlesung (wenn nicht anders erwähnt) seien alle Wertemengen IN, IR, Z, ... stets vervollständigt, ohne daß wir dies durch das hochgestellte Zeichen $\perp$ besonders kenntlich machen.

3.2 Die Existenz nicht-berechenbarer Funktionen

Wir zeigen hier, daß es Funktionen gibt, die nicht mit einem Algorithmus berechnet werden können. Die Grundidee beruht auf der Tatsache, daß Algorithmen auf Texten arbeiten; da aber Algorithmen selbst als Texte formuliert werden, gibt es also Algorithmen, die auf ihrer eigenen Formulierung arbeiten. Liegt eine solche Situation vor, dann kann man aus der Annahme, mit Algorithmen könnte man alle denkbaren Funktionen beschreiben, Widersprüche konstruieren.

Ein Algorithmus hat eine endliche Beschreibung und wird in einer bestimmten Sprache (bei uns zunächst in Deutsch) formuliert. Die Wörter und Sätze der Sprache sind Zeichenfolgen eines *endlichen Alphabets* X. Das Alphabet, in dem wir unsere Algorithmen formulieren, ist durch die Zeichen einer Tastatur gegeben und lautet:

X={'a',...,'z','A',...,'Z','0',...,'9',',',':',...,'-','+','◊'}.

◊ symbolisiert das Leerzeichen; dies ist der Zwischenraum, der eines der wichtigsten Zeichen in der Sprache ist, auch wenn man ihn nicht hinschreibt. Der Zwischenraum ist streng von der leeren Folge ε zu unterscheiden. Jeder Algorithmus α ist dann eine Zeichenfolge, also $\alpha\in X^*$.

Um im folgenden die Zeichen von den Namen von Funktionen, Mengen usw. zu unterscheiden, werden wir Zeichen stets in Hochkomma einschließen.

Wir können annehmen, daß als Ein- und Ausgabewerte von Algorithmen ebenfalls beliebige Zeichenfolgen über diesem Alphabet X vorkommen können. Jeder Algorithmus $\alpha\in X^*$ berechnet dann eine Funktion (Bez.: f_α)

$f_\alpha: X^* \to X^*$.

Nun bestimmen wir die "Anzahl" aller möglichen Algorithmen über dem Alphabet X und anschließend die "Anzahl" aller Funktionen $f: X^* \to X^*$. Ein Vergleich der beiden Ergebnisse wird zeigen, daß es mehr Funktionen $f: X^* \to X^*$ als mögliche Algorithmen gibt. Daraus können wir folgern, daß es dann mindestens eine Funktion $f: X^* \to X^*$ geben muß, zu der es *keinen* Algorithmus $\alpha \in X^*$ gibt, der die Funktion f berechnet.

Der Begriff "Anzahl" ist nicht ganz zweckmäßig, denn offenbar gibt es sowohl unendlich viele verschiedene Algorithmen als auch unendlich viele verschiedene Funktionen $f: X^* \to X^*$ gibt. Wie vergleicht man zwei unendliche Mengen ("Welches Unendlich ist mehr?")?
Die Lösung liefern hier die Begriffe der *Abzählbarkeit* und der *Überabzählbarkeit*. Anschaulich heißt eine Menge M abzählbar unendlich (kurz: abzählbar), wenn sie gleichmächtig zur Menge der natürlichen Zahlen IN ist. Man kann die Elemente von M dann nacheinander durchnumerieren ("abzählen"), also

$$M = \{m_1, m_2, m_3, ...\}.$$

Definition A:
Eine unendliche Menge M heißt **abzählbar unendlich** (kurz: **abzählbar**), wenn es eine bijektive Abbildung $f: IN \to M$ gibt (oder – was gleichwertig ist – wenn es eine bijektive Abbildung $g: M \to IN$ gibt). f nennt man **Abzählung**. Ist M unendlich, aber nicht abzählbar, so heißt M **überabzählbar**.

Offensichtlich sind unendliche Teilmengen abzählbarer Mengen wieder abzählbar, denn hat man eine Abzählung der Grundmenge gegeben, so streicht man alle Elemente, die nicht zur Teilmenge gehören und numeriert die verbleibenden Elemente beginnend mit 1,2,3,... neu durch.

Satz B:
1) Jede unendliche Teilmenge einer abzählbaren Menge ist abzählbar.
2) Jede Obermenge einer überabzählbaren Menge ist überabzählbar.

Beweis:
1) Sei M eine abzählbare Menge und $M' \subseteq M$ eine unendliche Teilmenge. Sei

$$f: IN \to M$$

eine Abzählung von M. Seien $k_0,k_1,k_2,k_3,... \in$ IN diejenigen natürlichen Zahlen, für die $f(k_i) \in M'$ gilt. Eine Abzählung f' von M' erhält man dann durch die Definition:

$f': IN \to M'$ mit $f'(i)=f(k_i)$ für alle $i \in IN$. ♦

Beispiele:

1) Die Menge der ganzen Zahlen ist abzählbar. Man definiert hierzu eine bijektive Abbildung

$f: IN \to Z$ durch

$$f(i) = \begin{cases} i/2, & \text{falls i gerade,} \\ 1/2(-1-i), & \text{falls i ungerade.} \end{cases}$$

Hierdurch erhält man eine Numerierung gemäß Tabelle 1, d.h.,

die Zahl 0 bekommt die Nummer 0,
die Zahl -1 die Nummer 1,
die Zahl 1 die Nummer 2,
die Zahl -2 die Nummer 3 usw.

Auf diese Weise sind alle ganzen Zahlen durchnumeriert, denn jede ganze Zahl kommt in der Numerierung genau einmal vor.

i	0	1	2	3	4	5	6	7	8	9	10	...
f(i)	0	-1	1	-2	2	-3	3	-4	4	-5	5	...

Tab. 1: Abzählung der ganzen Zahlen

2) Die Menge der rationalen Zahlen Q ist abzählbar. Wir zeigen dies zunächst für die nicht-negativen rationalen Zahlen $Q^+=\{r \in Q \mid r \geq 0\}$. Jede rationale Zahl läßt sich als Bruch darstellen. Die Brüche zählen wir nun ab. Hierzu ordnet man alle Brüche p/q mit $p,q \in$ IN, $q \neq 0$, gemäß Tabelle 2 an. Die Tabelle liefert zu jedem Bruch eine natürliche Zahl und umgekehrt:

$0/1$ erhält die Nummer 0,
$0/2$ erhält die Nummer 1,
$1/1$ erhält die Nummer 2,
$0/3$ erhält die Nummer 3,
$1/2$ erhält die Nummer 4 usw.

Allgemein realisiert die Tabelle folgende bijektive Abbildung

f: "Menge der nichtnegativen Brüche" $\to$ IN mit

$f(p/q)=n$, wobei

$n=((p+q)^2+p-q)/2$ für alle $p \geq 0$, $q \geq 1$ gilt.

q \ p	0	1	2	3	4	5	6	...
1	0	2	5	9	14	20	...	
2	1	4	8	13	19	...		
3	3	7	12	18	...			
4	6	11	17	...				
5	10	16	...					
6	15	...						
...								

Tab. 2: Abzählung der nichtnegativen Brüche

Die negativen Brüche kann man ebenso hinzunehmen, wie man die ganzen Zahlen oben zu den natürlichen Zahlen hinzugefügt hat. Es folgt: Die Brüche sind abzählbar. Durch diese Zählung sind zwar alle Brüche $^{p}/_{q}$ mit $p,q \in \mathbb{N}$, $q \neq 0$, numeriert worden; da aber verschiedene Brüche die gleiche rationale Zahl (z.B. die Zahl $2 = {}^{2}/_{1} = {}^{4}/_{2}$) darstellen können, liefert dies noch keine Abzählung der rationalen Zahlen. Offenbar ist aber die Menge der rationalen Zahlen eine unendliche Teilmenge der Menge aller Brüche. Daher folgt nach Satz B auch die Abzählbarkeit der rationalen Zahlen.

In den folgenden beiden Sätzen zeigen wir nun, daß die Menge aller Algorithmen $\alpha \in X^*$ abzählbar, die Menge aller Abbildungen $f: X^* \to X^*$ jedoch überabzählbar ist. Dann muß es also Funktionen geben, die sich nicht durch einen Algorithmus berechnen lassen.

Satz C:
Die Menge aller Algorithmen ist abzählbar.

Beweis:
Die Menge aller Algorithmen ist eine Teilmenge von X^*. Wegen Satz B brauchen wir also nur zu zeigen, daß die Menge *aller* endlich langen Texte X^*, die man über dem Alphabet X bilden kann, abzählbar ist.
Sei $|X|$ die Anzahl der Elemente des Alphabets X. Die Elemente von X mögen irgendwie angeordnet sein, z.B. so:

X={'a',...,'z','A',...,'Z','0',...,'9',',',':',...,'-','+','◊'}.

Sei

$p: X \to \{1,2,...,|X|\}$

eine totale Funktion, die jedem Zeichen $x \in X$ die Stelle zuordnet, an der es im Alphabet X auftritt. Zum Beispiel gilt p('a')=1, p('z')=26, p('7')=60 usw.

Sei nun $w \in X^*$ ein beliebiges Wort. w hat eine endliche Länge $|w|=n \in \mathbb{N}$, d.h.

$$w=x_1x_2x_3...x_n,\ x_i \in X \text{ für alle } i \in \{1,...,n\}.$$

$x_1,...,x_n$ sind die Zeichen, aus denen w aufgebaut ist. Dann definieren wir folgende Funktion:

$\phi: X^* \to \mathbb{N}$ durch

$$\phi(w)=\phi(x_1x_2x_3...x_n)=\sum_{i=1}^{n} p(x_i)\cdot(|X| + 1)^i.$$

Speziell sei $\phi(\varepsilon)=0$. Offensichtlich ordnet ϕ jedem Wort aus X^* genau eine Zahl zu. ϕ ist also eine totale Funktion. Wir behaupten: ϕ ist injektiv. Hierzu beachte man, daß stets

$$\phi(x_1x_2x_3...x_n) = \sum_{i=1}^{n} p(x_i)\cdot(|X| + 1)^i < (|X| + 1)^{n+1}$$

gilt. Man betrachte nun zwei beliebige Wörter $v,w \in X^*$. Wenn diese zwei Wörter v und w verschiedene Länge besitzen, dann gilt wegen obiger Ungleichung sicher $\phi(w)\neq\phi(v)$. Sei daher $|w|=|v|$, also $w=x_1x_2x_3...x_n$ und $v=y_1y_2y_3...y_n$ für ein $n\geq 0$. Weiterhin gelte $\phi(w)=\phi(v)$, d.h.:

$$\phi(w) = \sum_{i=1}^{n} p(x_i)\cdot(|X| + 1)^i = \sum_{i=1}^{n} p(y_i)\cdot(|X| + 1)^i = \phi(v),$$

folglich:

$$\sum_{i=1}^{n} (p(x_i)-p(y_i))\cdot(|X| + 1)^i = 0$$

Dies kann aber nur zutreffen, wenn $p(x_i)-p(y_i)=0$ für alle i gilt. Da p bijektiv ist, müssen also alle x_i gleich den y_i sein. Wir haben somit gezeigt, daß für beliebige Wörter v und w die Gleichheit $\phi(w) = \phi(v)$ nur gelten kann, wenn v=w ist. Folglich ist ϕ injektiv.

Wir betrachten nun $\phi(X^*)$. Da X^* eine unendliche Menge und ϕ eine injektive Funktion ist, muß auch $\phi(X^*)$ eine unendliche Menge sein. Nach Satz B ist jede unendliche Teilmenge von IN abzählbar unendlich, also auch $\phi(X^*)$. Da $\phi(X^*)$ gleichmächtig zur Menge X^* ist, folgt hieraus die Abzählbarkeit von X^*. Somit ist auch die Menge aller Algorithmen als unendliche Teilmenge von X^* abzählbar. ♦

Zur Erläuterung: Die Zuordnung ϕ: $X^* \to IN$ basiert auf der Idee des *Stellenwertsystems*. Hier liegt ein Stellenwertsystem der Form S=(|X|+1,X,p) vor. Die Zahl |X|+1 ist die *Basis* des Systems. X ist die Menge der "Ziffernsymbole". p liefert das Gewicht jedes "Ziffernsymbols" aus X. Das Dezimalsystem ist ebenfalls ein Stellenwertsystem der Form (10,{'0','1',...,'9'},p) mit p('i')=i, d.h., das Ziffern*symbol* 'i' wird auf die Ziffer i abgebildet.

Satz D:
Die Menge aller Abbildungen f: $X^* \to X^*$ ist überabzählbar.

Beweis:
Wir zeigen die Behauptung für die Menge aller totalen Funktionen f: $X^* \to X^*$ und setzen zur Abkürzung F={f | f: $X^* \to X^*$ und f ist total}. Ist gezeigt, daß F nicht abzählbar ist, so folgt nach Satz B(2) auch die Überabzählbarkeit der Menge *aller* Funktionen f: $X^* \to X^*$.
Wir beweisen die Behauptung *indirekt*, d.h., wir nehmen an, daß die Menge F abzählbar ist und führen diese Annahme zu einem Widerspruch.
Annahme: F ist abzählbar. Dann gibt es nach Definition eine Abzählung h: $IN \to F$, d.h., man kann die Abbildungen der Reihe nach durchnumerieren:

$$f_0, f_1, f_2, ..., f_{1761}, ...$$

Nach Satz C ist die Menge aller Wörter $w \in X^*$ abzählbar. Sei

$$w_0, w_1, w_2, ... \text{ mit } w_i \in X^*$$

solch eine Abzählung aller Wörter über dem Alphabet X. Sei $a \in X$ ein beliebiges festgewähltes Zeichen. Wir betrachten nun die Funktion

$$g: X^* \to X^* \text{ mit}$$
$$g(w_i) = f_i(w_i) \cdot a \text{ für alle } w_i \in X^*.$$

Das Bild von g zu einem Argument $w \in X^*$ erhält man so: Man ermittelt in der Abzählung $w_0, w_1, w_2, ...$ die natürliche Zahl i mit $w=w_i$, betrachtet in der Abzählung aller Funktionen aus F die entsprechende Abbildung f_i, wendet diese Abbildung auf w an und ergänzt das entstandene Wort am Ende um das Zeichen a. Somit ist die Funktion g total, also für jedes Wort $w \in X^*$ definiert, d.h.: $g \in F$. Dann muß g natürlich insbesondere in der obigen Abzählung der f_j vorkommen. Es muß also ein $k \in IN$ geben mit $g=f_k$ bzw. allgemein

$$g(v)=f_k(v) \text{ für alle } v \in X^*.$$

Für das spezielle Wort w_k folgt dann einerseits

$$g(w_k) = f_k(w_k),$$

andererseits folgt aus der Definition von g aber

$$g(w_k) = f_k(w_k) \cdot a \neq f_k(w_k).$$

g kann für das Argument w_k nicht zwei verschiedene Werte $f_k(w_k)$ und $f_k(w_k)\cdot a$ besitzen. Folglich liegt ein Widerspruch vor. Die Annahme war also falsch, d.h., die Menge F ist nicht abzählbar. Damit ist die Behauptung bewiesen. ♦

Dieses Beweisverfahren stammt von dem Mathematiker G. Cantor (19. Jahrhundert). Es wurde von ihm u.a. verwendet, um zu zeigen, daß die Menge der reellen Zahlen überabzählbar ist. Das Verfahren heißt allgemein **Diagonalisierung**, weil man versucht, eine Annahme zum Widerspruch zu führen, indem man nur die Elemente auf einer Diagonalen heranzieht.
Ordnet man nämlich die Funktionen $f_0,f_1,f_2,...$ und ihre Argumente $w_0,w_1,w_2,...$ gemäß Tabelle 3 an, so werden zur Definition von g die Diagonalelemente $f_k(w_k)$ herangezogen. g weicht dann von jeder beliebigen Funktion f_i, $i\in$ IN, an der Stelle w_i ab und kann daher nicht in der Abzählung $f_0,f_1,f_2,...$ vorkommen.

w_j f_i	f_0	f_1	f_2	f_3	f_4	f_5	f_6 ...
w_0	$\mathbf{f_0(w_0)}$	$f_1(w_0)$	$f_2(w_0)$	$f_3(w_0)$	$f_4(w_0)$	$f_5(w_0)$ ...	
w_1	$f_0(w_1)$	$\mathbf{f_1(w_1)}$	$f_2(w_1)$	$f_3(w_1)$	$f_4(w_1)$ ...		
w_2	$f_0(w_2)$	$f_1(w_2)$	$\mathbf{f_2(w_2)}$	$f_3(w_2)$ ...			
w_3	$f_0(w_3)$	$f_1(w_3)$	$f_2(w_3)$ ...				
w_4	$f_0(w_4)$	$f_1(w_4)$ ...					
w_5	$f_0(w_5)$ ...						
...							

Tab. 3: Prinzip der Diagonalisierung

Wir haben nun gezeigt, daß die Menge aller Funktionen f: $X^*\rightarrow X^*$ überabzählbar ist, während die Menge aller Algorithmen nur abzählbar ist. Folglich gibt es Funktionen, die man nicht durch einen Algorithmus berechnen kann.

Satz E:
Es gibt eine Funktion f: $X^*\rightarrow X^*$, die nicht durch einen Algorithmus berechnet werden kann.

Satz E beinhaltet eine reine Existenzaussage. Wir wissen deshalb noch nicht, wie solch eine nicht-berechenbare Funktion aussieht. Insbesondere wäre es interessant zu wissen, ob es auch für die Praxis relevante Funktionen gibt, die nicht algorithmisch berechnet werden können. Im folgenden Abschnitt wollen wir daher einige konkrete nicht-berechenbare Funktionen angeben.

Bemerkung: Es gibt sogar überabzählbar viele Funktionen, die nicht durch Algorithmen beschrieben werden. Denn wenn es nur abzählbar viele solcher Funktionen gäbe, dann könnte man ihre Abzählung abwechselnd mit der Abzählung der algorithmisch beschreibbaren Funktionen hintereinanderschreiben, woraus sich eine Abzählung aller Funktionen ergäbe, im Widerspruch zu Satz D. Anschaulich kann man diese Aussage so deuten: Würde man willkürlich aus der Menge aller Funktionen $f: X^* \to X^*$ eine beliebige Funktion g herausgreifen, so könnte man sicher sein, daß diese nicht durch einen Algorithmus beschrieben werden kann. Die Zahl der berechenbaren Funktionen ist also verschwindend gering.

3.3 Beispiele nicht-berechenbarer Funktionen

Wir geben nun einige konkrete nicht-berechenbare Funktionen an. Zu einem Algorithmus $\alpha \in X^*$ bezeichne

$$f_\alpha: X^* \to X^*$$

wieder die Funktion, die dieser Algorithmus berechnet.

Es ist klar, daß ein Algorithmus, der für ein spezielles Problem formuliert ist, eine ganz präzise vorgegebene Menge von Eingabewerten verarbeiten kann. So akzeptiert z.B. ein Algorithmus zum Multiplizieren nur Zahlen als Eingabe. Versucht man etwa mit diesem Algorithmus einen Text zu bearbeiten, so kommt man nicht weiter, weil dies im Multiplikationsalgorithmus nicht vorgesehen ist. Weitere besondere Situationen sind z.B.: Ein Algorithmus terminiert für manche Eingaben nicht, oder: Auf bestimmte Eingaben hin erhält man eine unzulässige Situation, z.B. Division durch Null. In allen diesen Fällen sagen wir: Der Algorithmus ist für die entsprechende Eingabe nicht definiert. Mathematisch ausgedrückt: Diese Eingabe liegt nicht im Definitionsbereich $D(f_\alpha)$ der Funktion f_α. $D(f_\alpha)$ ist somit die Menge aller Eingabewerte, für die der Algorithmus α ohne Fehler terminiert.

Wir betrachten folgenden Algorithmus $\delta \in X^*$, der beliebige Wörter $w \in X^*$ einliest, die Anzahl der Zeichen zählt und die Länge des Wortes w ausgibt. Offensichtlich gilt dann für f_δ:

$$f_\delta: X^* \to X^* \text{ mit } f_\delta(w)=|w|.$$

Der Definitionsbereich ist die Menge aller Wörter, also $D(f_\delta)=X^*$.

Der Algorithmus δ ist selbst ein Wort aus X^*, das wir ebenfalls als Eingabe verwenden können. Wenn δ seinen eigenen Text als Eingabe erhält, so gilt natürlich

$$f_\delta(\delta)=|\delta|,$$

d.h., der Algorithmus δ stellt dann fest, aus wievielen Zeichen er selbst aufgebaut ist.

Diese Methode, einem Algorithmus seinen eigenen Text als Eingabe zuzuführen, ist die zentrale Idee, auf der alle folgenden Aussagen über nicht-berechenbare Funktionen beruhen.

Fassen wir nun einen Algorithmus als Text auf, so kann man jedem beliebigen Algorithmus $\alpha \in X^*$ seinen eigenen Text als Eingabe anbieten. Wir können dann unterscheiden, ob α mit der Eingabe α etwas anfangen kann, also eine Ausgabe liefert und ohne Fehler terminiert oder undefiniert ist, ob also

$$\alpha \in D(f_\alpha)$$

gilt oder nicht. Frage: Können wir auch maschinell entscheiden, ob für beliebige $\alpha \in X^*$ gilt: $\alpha \in D(f_\alpha)$? Mit anderen Worten: Gibt es einen Algorithmus β, der beliebige Texte von Algorithmen $\alpha \in X^*$ als Eingabe erhält, und der 'ja' ausgibt, falls $\alpha \in D(f_\alpha)$ ist, und 'nein', falls $\alpha \notin D(f_\alpha)$ gilt? β soll also folgende Funktion $S: X^* \to X^*$ berechnen mit

$$S(\alpha) = \begin{cases} \text{'ja', falls } \alpha \text{ ein Algorithmus ist und } \alpha \in D(f_\alpha), \\ \text{'nein', sonst.} \end{cases}$$

S ist eine totale Funktion.

Der folgende Satz besagt, daß ein Algorithmus β für S nicht existiert. Wir haben damit ein erstes Problem gefunden, das nicht durch einen Algorithmus gelöst werden kann.

Satz F:
Es gibt keinen Algorithmus, der bei Eingabe beliebiger Algorithmen $\alpha \in X^*$ ausgibt, ob $\alpha \in D(f_\alpha)$ gilt oder nicht, d.h., die obige Funktion S ist nicht berechenbar.

Beweis:
Wir beweisen den Satz durch Widerspruch. Wir nehmen also an, daß ein Algorithmus existiert, der die Funktion S berechnet.
Sei $a \in X$ ein fest gewähltes Zeichen. Wenn S berechenbar ist, dann ist auch folgende totale Funktion $g: X^* \to X^*$ berechenbar mit

$$g(\alpha) = \begin{cases} f_\alpha(\alpha) \cdot a, \text{ falls } S(\alpha) = \text{'ja'}, \\ \text{'nein', falls } S(\alpha) = \text{'nein'}. \end{cases}$$

g läßt sich zum Beispiel durch folgenden Algorithmus γ berechnen:

γ: Lies ein beliebiges Wort $\alpha \in X^*$ ein;
berechne $S(\alpha)$ [dies ist nach Annahme möglich];
falls das Ergebnis gleich 'ja' ist, dann

berechne $f_\alpha(\alpha)$ [dieser Wert existiert dann!]
und gib den um das Zeichen 'a' verlängerten
Ergebnistext aus;
anderenfalls gib 'nein' aus.

Es gilt also $g=f_\gamma$ und insbesondere $g(\gamma)=f_\gamma(\gamma)$. Andererseits folgt aber aus der Definition von g die Gleichung

$$g(\gamma) = \begin{cases} f_\gamma(\gamma)\cdot a, & \text{falls } S(\gamma)=\text{'ja'}, \\ \text{'nein'}, & \text{falls } S(\gamma)=\text{'nein'}. \end{cases}$$

Die erste Alternative kann nicht zutreffen, da $g(\gamma)=f_\gamma(\gamma)\neq f_\gamma(\gamma)\cdot a$ ist. Es muß also $g(\gamma)=\text{'nein'}=f_\gamma(\gamma)$ und $S(\gamma)=\text{'nein'}$ gelten.
Die letzte Gleichung liefert nach Definition von S die Bedingung $\gamma \notin D(f_\gamma)$, d.h., der Algorithmus γ ist für seinen eigenen Text als Eingabe nicht definiert. Dies ist ein Widerspruch, denn $f_\gamma(\gamma)$ ist sehr wohl definiert ist, nämlich

$$f_\gamma(\gamma)=g(\gamma)=\text{'nein'}.$$

Also war die Annahme falsch, d.h., S kann nicht mit einem Algorithmus berechnet werden. ♦

Beachten Sie, daß wir den Widerspruch wiederum durch Diagonalisierung ähnlich zum Beweis von Satz D erzeugt haben.

Satz F löst das sog. **Selbstanwendungsproblem** im negativen Sinne.
Nun hat das Selbstanwendungsproblem noch keine besondere praktische Bedeutung. Welchen Zweck kann es haben, einen Algorithmus auf seine eigene Beschreibung anzusetzen? Die nächsten Sätze werden zeigen, daß das Selbstanwendungsproblem gewissermaßen das einfachste unter all den Problemen darstellt, die sich mit der maschinellen Untersuchung von Eigenschaften von Algorithmen befassen. Idee des weiteren Vorgehens: Wenn man schon nicht algorithmisch entscheiden kann, ob ein beliebiger Algorithmus für seinen eigenen Text als Eingabe definiert ist oder nicht, dann kann man erst recht nicht entscheiden, ob ein beliebiger Algorithmus überhaupt für eine Eingabe definiert ist. Dies führt auf das sog. *Halteproblem*. Man kann nicht maschinell feststellen, ob ein beliebiger Algorithmus für eine beliebige Eingabe terminiert oder nicht.
Diese Methode der *Reduktion* eines Problems auf ein anderes (bekanntes) ist eine fundamentale Idee der Informatik und wird Sie während des gesamten Studiums begleiten.

Satz G:

Es gibt keinen Algorithmus, der bei Eingabe beliebiger Algorithmen $\alpha \in X^*$ und beliebiger Wörter $w \in X^*$ ausgibt, ob $w \in D(f_\alpha)$ gilt oder nicht. Präziser: Die totale Funktion $h: X^* \times X^* \to X^*$ mit

$$h(\alpha,w) = \begin{cases} \text{'ja'}, & \text{falls } w \in D(f_\alpha), \\ \text{'nein'}, & \text{sonst,} \end{cases}$$

kann nicht durch einen Algorithmus berechnet werden.

Beweis:
Dieses Halteproblem kann man auf das Selbstanwendungsproblem zurückführen, indem man zeigt, daß aus der Lösbarkeit des Halteproblems die Lösbarkeit des Selbstanwendungsproblems folgt, was im Widerspruch zu Satz F steht.
Angenommen, es gibt einen Algorithmus, der das Halteproblem löst, der also h berechnet. Wir definieren eine neue totale Funktion

$g: X^* \to X^*$ mit

$g(\alpha) = h(\alpha,\alpha)$.

Mit h ist auch g berechenbar. Es gibt also einen Algorithmus γ, der g berechnet. g beschreibt aber genau das Selbstanwendungsproblem; denn es gilt $g(\alpha)$ = 'ja' genau dann, wenn $\alpha \in D(f_\alpha)$ ist. Das ist aber ein Widerspruch zu Satz F. Folglich ist Satz G bewiesen. ♦

Man beachte, daß beide Resultate nur besagen: Man kann die Probleme nicht algorithmisch für *alle* Algorithmen $\alpha \in X^*$ (und ggf. $w \in X^*$) lösen. Für *einzelne* gegebene Algorithmen α kann man **und muß man** sich meist einen speziellen Beweis einfallen lassen, um das Selbstanwendungs- oder das Halteproblem zu lösen, denn vor allem das Halteproblem steckt inhärent in jeder Softwareentwicklung. Vgl. das letzte Beispiel in Abschnitt 3.4.

Eine wesentlich allgemeinere Aussage über die algorithmische Unlösbarkeit von Problemen gibt der folgende Satz (von dem Mathematiker Rice) wieder, den wir ohne Beweis angeben, weil dazu größere Vorbereitungen notwendig sind.

Satz H: (umgangssprachliche, nicht exakte Formulierung)
Sei E eine beliebige Eigenschaft, die von mindestens einer, aber nicht von allen berechenbaren Funktionen $f: X^* \to X^*$ erfüllt wird. Dann gibt es keinen Algorithmus, der für beliebige Algorithmen $\alpha \in X^*$ ausgibt, ob f_α die Eigenschaft E erfüllt oder nicht.

Anschaulich besagt der Satz, daß die Aufgabe, einem Algorithmus "anzusehen", ob die durch ihn berechnete Funktion eine bestimmte Eigenschaft besitzt, nicht algorithmisch gelöst werden kann.

3.4 Die Churchsche These

Unsere Sätze und Beweise über nicht-berechenbare Funktionen bezogen sich nie auf konkrete Maschinen, sondern wir haben stets über Algorithmen argumentiert. Konkrete Maschinen (Computer) arbeiten Programme ab, und im intuitiven Sinne ist klar, daß diese Abarbeitungen von Programmen mechanisch durchführbare Verfahren, also Algorithmen sind. Die Sätze gelten trivialerweise auch für Maschinen. Kann aber umgekehrt auch jeder Algorithmus auf einem Computer ausgeführt werden? Es wäre vorstellbar, daß man ein algorithmisches Verfahren angeben kann, das im intuitiven Sinne effektiv durchführbar ist, das aber von keinem Computer abgearbeitet werden kann. Jahrzehntelange Untersuchungen über dieses Problem führten zu keinem allgemein anerkannten Gegenbeispiel. Man glaubt daher an folgende These, die der amerikanische Logiker A. Church bereits im Jahre 1936 formuliert hat:

Jede im intuitiven Sinne berechenbare Funktion ist
maschinell berechenbar und umgekehrt.

Anschaulich besagt die These, daß jedes Problem, zu dem man ein in irgendeiner Form aufgeschriebenes algorithmisches Lösungsverfahren angeben kann, auch von einem Computer gelöst werden kann und umgekehrt. "Im intuitiven Sinne" bedeutet dabei, daß das Verfahren von einer genügend großen Zahl von Fachleuten als algorithmisch anerkannt ist. Sie werden dieser These in Vorlesungen über Theoretische Informatik im Laufe Ihres Studiums nochmal begegnen.

Die Churchsche These ist kein mathematischer Satz, und sie kann auch nicht bewiesen werden, da der Begriff der im intuitiven Sinne berechenbaren Funktion nicht präzisiert ist und in der Umgangssprache auch nicht exakt präzisiert werden kann. Da aber seit über 50 Jahren alle Versuche gescheitert sind, die Churchsche These zu widerlegen, wird sie heute allgemein anerkannt. Man ist sogar so sehr von ihrer Gültigkeit überzeugt, daß mathematische Beweise, die sich auf sie berufen, akzeptiert werden.

Die Sätze in Abschnitt 3.2 und 3.3, bewiesen für Algorithmen, können trivialerweise auf Maschinen übertragen werden, da beide Modelle nach der Churchschen These äquivalent sind.

Beispiel: Ein Software-Haus entwickelt laufend Computerprogramme für die unterschiedlichsten Zwecke und möchte jedesmal überprüfen, ob das fertige Programm das leistet, was der Entwickler oder der Auftraggeber sich vorgestellt haben. Diese Überprüfung kann man nicht von einem Computer für alle Programme vornehmen lassen, da das Problem nach Satz H nicht maschinell lösbar ist.

Das Beispiel zeigt, daß wesentliche Elemente bei der Programmentwicklung (Erstellung, Überprüfung usw.) nicht automatisch durchgeführt werden können. Jede einzelne Entwicklung eines Programms erfordert also eigene Überlegungen, die vom Menschen durchzuführen sind. Jedoch hat man gewisse allgemeine Methoden erarbeitet, die in der Praxis die Programmerstellung wesentlich erleichtern.