B.A. Trachtenbrot

Wieso können Automaten rechnen?

Übersetzung: Karl-Heinz Hoppe
1965 Deutscher Verlag der Wissenschaften
MSB: Nr. 60
Abschrift und LaTex-Satz: 2021 https://mathematikalpha.de

Einführung

Seit dem Ende des Krieges nahmen die elektronischen Rechenautomaten eine ungeheure Entwicklung. Sie werden daher heute schon zur Lösung der verschiedenartigsten mathematischen und logischen Probleme verwandt. Diese modernen Maschinen unterscheiden sich durch eine ganz charakteristische Besonderheit von den früheren Rechenmaschinen:
Es werden gewisse Anfangswerte und ein Programm eingegeben; dann arbeitet die Maschine an Hand des Programms ohne jedes menschliche Eingreifen, bis das endgültige Resultat herauskommt.

Die modernen elektronischen Rechenautomaten sind außerordentlich leistungsfähig. Sie führen in einer Sekunde bis zu 20000 arithmetische Operationen aus.
Das sind mehr als zehnmal so viel, wie ein Büro qualifizierter Rechner erreichen kann, das mit guten Tischrechenmaschinen arbeitet.

Rechenautomaten sind äußerst vielfältig anwendbar. Mit ihrer Hilfe können komplizierte Gleichungssysteme gelöst, Übersetzungen aus einer Sprache in eine andere ausgeführt, Schachwettkämpfe usw. durchgeführt werden. Sehr große Perspektiven bestehen noch bei der Überwachung und Steuerung technologischer Prozesse im Zuge der Automatisierung der Industrie. Außerdem werden dadurch, dass Versuchsergebnisse analysiert und schnell und sicher mathematisch bearbeitet werden können, die Voraussetzungen für die Entdeckung und Anwendung neuer, bisher nicht anwendbarer Untersuchungsmethoden in vielen Gebieten der Wissenschaft geschaffen.

Man weiß heute bereits, dass die elektronischen Rechenautomaten mächtige Hilfsmittel sind, die dem Menschen nicht nur die ganze Arbeit erleichtern können, sondern ihn sogar von gewissen Formen anstrengender geistiger Arbeit vollkommen zu befreien vermögen.

Diese ungeheuren Erfolge verleiten natürlich zu vielen unbegründeten Illusionen und erzeugen fälschlich die phantastischsten Prognosen bezüglich der Allmacht der Maschinen.
Insbesondere sei auf das Reklamegeschrei hingewiesen, das gelegentlich in der Presse über "gigantische Elektronenhirne" und über Automaten erhoben wird, die in der Lage sein sollen, beliebige Aufgaben zu lösen und die schöpferische Arbeit der Gelehrten zu ersetzen.

In diesem Zusammenhang taucht natürlich die Frage auf, welche Formen geistiger Arbeit wirklich von Rechenautomaten übernommen werden können. Diese Frage wird in der modernen Theorie der Algorithmen, einem wichtigen Teilgebiet der mathematischen Logik, von einem ganz bestimmten Standpunkt aus untersucht und beantwortet.

Zur mathematischen Logik gehört die Untersuchung des Wesens solcher Begriffe wie "Rechenprozess", "mathematischer Beweis", "Algorithmus" usw.
Schon einige Jahre vor der Konstruktion des ersten elektronischen Rechenautomaten wurden in der mathematischen Logik der exakte Begriff "Algorithmus" und das allgemeine Schema eines Rechenautomaten (Turing-Maschine)[1] erarbeitet sowie der enge Zusammenhang zwischen Algorithmen und Maschinen aufgedeckt.

Dadurch wurde es möglich, eine Reihe wichtiger Sätze aufzustellen, die das Wesen der in Automaten realisierbaren Prozesse enthüllten. Insbesondere konnte man exakt die Existenz solcher Probleme nachweisen, deren maschinelle Lösung unmöglich ist. Der Untersuchung des

[1] A. M. Turing, On computable numbers, with an application to the Entscheidungsproblem, Proc. London math. Soc. (23), 42, p. 230-265 (1936).

Zusammenhangs zwischen Algorithmen und Maschinen soll das vorliegende Bändchen gewidmet sein.

In den Paragraphen 1 bis 3 wird anhand von Beispielen erläutert, was ein Algorithmus ist. Es werden Algorithmen zur Lösung gewisser Klassen mathematischer und logischer Aufgaben entwickelt.
In den Paragraphen 4 und 5 werden die Prinzipien der Konstruktion elektronischer Rechenautomaten und des Aufstellens von Programmen, d.h. von Algorithmen, die zur Realisierung in Automaten geeignet sind, erörtert.

Die Paragraphen 6 bis 11 sind einigen wichtigen Tatsachen aus der Theorie der Algorithmen gewidmet, wobei als Grundbegriff der Theorie die Turing-Maschine verwendet wird.

In diesem kleinen Bändchen können natürlich viele Beweise schon ihrer Länge wegen nicht vollständig gebracht werden. Daher ist die Darstellung an einzelnen Stellen weder streng noch vollständig, was jedoch, wie uns scheint, kein Nachteil ist, sondern eher das Verständnis des Wesens der Sache erleichtert. Zur Abrundung des Bildes bringt § 6 einige allgemeine Bemerkungen.

Es sei noch auf folgendes hingewiesen:
Moderne Maschinen mit automatischer Steuerung nennt man elektronisch, da ihre wichtigsten Teile Elektronenröhren enthalten. Die Elektronik macht es möglich, die ökonomisch so wichtigen hohen Arbeitsgeschwindigkeiten zu erreichen.

Das Grundprinzip dieser Maschinen - die automatische Steuerung der Prozesse - ist jedoch nicht an die Verwendung der Elektronik gebunden. Prinzipiell könnte man die Elektronenröhren durch mechanisch arbeitende Teile ersetzen, d.h., man könnte eine mechanische Rechenmaschine mit vollautomatischer Steuerung bauen, die genau dieselben Aufgaben lösen kann wie eine elektronische (nur bedeutend langsamer).
Die Entwicklung der modernen Rechenmaschinen darf also keineswegs nur den Fortschritten der Elektronik zugeschrieben werden. Das erste 1936 im Rahmen der Theorie der Algorithmen in der Literatur angegebene allgemeine Schema eines Rechenautomaten (siehe § 7, Turing-Maschine) sah mechanische Hilfsmittel vor. Die erste tatsächlich gebaute Maschine (1940) war jedoch eine elektronische.

Bei der Beschreibung der Konstruktion der Maschinen werden wir jedoch nicht auf technische Einzelheiten eingehen. In dem vorliegenden Bändchen soll die Hauptaufmerksamkeit vielmehr auf das Zusammenwirken der einzelnen Teile der Maschinen gerichtet werden.
Das entspricht seiner Zielsetzung, welche in der Erörterung der mathematischen und logischen Möglichkeiten und nicht in der Beschreibung technischer Einzelheiten besteht.

Inhaltsverzeichnis

1 Numerische Algorithmen

Der Begriff "Algorithmus" gehört zu den Grundbegriffen der Mathematik. Unter einem Algorithmus versteht man eine genaue Vorschrift, nach der ein gewisses System von Operationen in einer bestimmten Reihenfolge auszuführen ist und nach der man alle Aufgaben eines gegebenen Typs lösen kann.

Das ist natürlich noch keine genaue Definition des Begriffs Algorithmus. Es wird nur kurz die inhaltliche Bedeutung des Wortes "Algorithmus" erklärt.
Dennoch ist diese Formulierung jedem Mathematiker geläufig und verständlich. Sie erfasst den Begriff des Algorithmus, wie er sich im Laufe der Zeit herausgebildet hat und schon im Altertum in der Mathematik verwendet wurde.

Die einfachsten Algorithmen sind die Regeln, nach denen im dezimalen Zahlensystem die vier Grundrechenarten ausgeführt werden. (Das Wort "Algorithmus" selbst geht auf den Namen des usbekischen Gelehrten Al Hwarizmi zurück, der schon im IX. Jahrhundert solche Regeln aufstellte.)
So lässt sich beispielsweise die Addition zweier mehrstelliger Zahlen auf eine Kette elementarer Operationen zurückführen, bei denen der Rechner jeweils nur zwei entsprechende Ziffern der Summanden zu beachten braucht (von denen eine mit einem Zeichen versehen sein kann, das auf den Zehner-Übertrag hinweist). Hierbei treten zwei Arten von Operationen auf:

1. Das Niederschreiben der entsprechenden Ziffer der Summe;
2. der Hinweis auf den Übertrag für die links benachbarte Ziffer;

ferner ist vorgeschrieben, in welcher Reihenfolge die Operationen auszuführen sind (von rechts nach links).
Der formale Charakter dieser elementaren Operationen besteht darin, dass sie sich mit Hilfe einer ein für allemal vorgegebenen Additionstabelle für die Ziffern völlig automatisch ausführen lassen, wobei von ihrer inhaltlichen Bedeutung völlig abgesehen werden kann.

Ähnlich sieht es mit den anderen drei Grundrechenarten sowie dem Quadratwurzelziehen und ähnlichen Operationen aus. Der formale Charakter der entsprechenden Vorschriften (Algorithmen!) lässt offenbar keinerlei Zweifel aufkommen (bei Schülern kann das besonders bei der Vorschrift für das Quadratwurzelziehen festgestellt werden).

Als weiteres Beispiel wollen wir den Euklidischen Algorithmus betrachten, mit dem man alle Aufgaben des folgenden Typs lösen kann:

Zu zwei natürlichen Zahlen a, b ist der größte gemeinsame Teiler zu bestimmen.

Verschiedenen Aufgaben dieses Typs entsprechen offensichtlich nur verschiedene Zahlenpaare a, b.
Bekanntlich lässt sich die Lösung einer solchen Aufgabe mittels einer fallenden Folge von Zahlen konstruieren, von denen die erste die größere der beiden vorgegebenen Zahlen, die zweite die kleinere, die dritte der Rest bei der Division der ersten durch die zweite, die vierte der Rest bei der Division der zweiten durch die dritte ist und so fort.

Der Prozess wird solange fortgesetzt, bis die Division ohne Rest aufgeht. Der Divisor der letzten Division ist dann der gesuchte größte gemeinsame Teiler der beiden Zahlen.
Da man die Division auf wiederholte Subtraktionen zurückführen kann, lässt sich die Vorschrift zur Lösung einer solchen Aufgabe auch in Form folgender, nacheinander auszuführender An-

weisungen angeben:

Anweisung 1. Notiere die beiden Zahlen a und b. Gehe über zur folgenden Anweisung.

Anweisung 2. Vergleiche die notierten Zahlen ($a = b$ oder $a < b$ oder $a > b$). Gehe über zur folgenden Anweisung.

Anweisung 3. Wenn die notierten Zahlen gleich sind, so leistet jede von ihnen das Gewünschte; dann ist die Rechnung abzubrechen. Ist das nicht der Fall, dann gehe zur folgenden Anweisung über.

Anweisung 4. Wenn die erste der notierten Zahlen kleiner als die zweite ist, so vertausche ihre Plätze und gehe zur folgenden Anweisung über.

Anweisung 5. Subtrahiere die zweite Zahl von der ersten und notiere folgende beiden Zahlen: den Subtrahenden und die Differenz. Gehe über zur Anweisung 2.

Wenn also alle fünf Anweisungen ausgeführt worden sind, soll man zur zweiten zurückkehren, danach zur dritten, vierten, fünften und wiederum zur zweiten, dritten usw. übergeben, bis zwei gleiche Zahlen auftreten; wenn die in der dritten Anweisung enthaltene Bedingung erfüllt ist, bricht der Prozess ab.

In der Mathematik werden natürlich Algorithmen nicht immer so pedantisch formal beschrieben. Dass jeder der bekannten Algorithmen in dieser Form angegeben werden kann, wird jedoch anscheinend nicht bezweifelt.

In der oben angeführten Beschreibung des Euklidischen Algorithmus ist der Lösungsprozess in mehrere elementare Operationen aufgegliedert werden: Subtraktion zweier Zahlen, Vergleich zweier Zahlen, Vertauschung der Plätze zweier Zahlen.
Man sieht aber auch leicht ein, dass diese Aufgliederung noch viel weiter getrieben werden kann. So lässt sich z.B. die Anweisung 5 bezüglich der Subtraktion der notierten Zahlen wiederum in ein System von Anweisungen zerlegen, das den Algorithmus der Subtraktion zweier Zahlen beschreibt; da aber die Regeln der Grundrechenarten hinreichend bekannt und einfach sind, wird man natürlich von einer weiteren Aufgliederung Abstand nehmen.

Algorithmen, mit deren Hilfe sich die Lösung eines Problems auf die vier arithmetischen Operationen zurückführen lässt, bezeichnet man als numerische Algorithmen. Sie spielen in verschiedenen Gebieten der elementaren, aber auch der höheren Mathematik eine wesentliche Rolle, ob sie nun in Worten, in Formeln oder in Schemata der mannigfachsten Art beschrieben sind.

So kann beispielsweise der Algorithmus zur Auflösung eines Systems von zwei linearen Gleichungen mit zwei Unbekannten,

$$a_1x + b_1y = c_1 \qquad , \qquad a_2x + b_2y = c_2$$

durch die Formeln

$$x = \frac{c_1b_2 - c_2b_1}{a_1b_2 - a_2b_1} \qquad , \qquad y = \frac{a_1c_2 - a_2c_1}{a_1b_2 - a_2b_1}$$

beschrieben werden. Durch diese Formeln wird die Art der auszuführenden Operationen sowie ihre Reihenfolge völlig festgelegt.[2]

Diese Formeln beschreiben ein und dieselbe Kette von Rechenoperationen für alle Aufgaben

[2] Bei diesen Überlegungen wird stillschweigend angenommen, dass es sich um eindeutig lösbare Gleichungssysteme handelt, d.h. solche, bei denen $a_1b_2 - a_2b_1 \neq 0$ ist.

des angegebenen Typs (d.h. für beliebige Koeffizienten $a_1, b_1, c_1, a_2, b_2, c_2$).

Es sei aber darauf hingewiesen, dass im allgemeinen die Anzahl der Operationen, die zur Ausführung eines Algorithmus notwendig sind, im voraus nicht bekannt zu sein braucht. Diese Anzahl kann von den konkreten Bedingungen jeder einzelnen Aufgabe abhängen und sich erst im Lösungsprozess selbst ergeben.
Insbesondere gilt das auch für den Euklidischen Algorithmus. Hier hängt die Anzahl der notwendigen Subtraktionen von der speziellen Auswahl des Zahlenpaares a, b ab.

Die weite Verbreitung der numerischen Algorithmen erklärt sich daraus, dass man viele andere Operationen auf die vier Grundrechenarten zurückführen kann. In vielen Fällen ist diese Zurückführung nicht völlig exakt, kann aber mit jeder beliebig vorgebbaren Genauigkeit ausgeführt werden.
Dies lässt sich sehr gut am Beispiel des Algorithmus zum Quadratwurzelziehen erläutern, der die Quadratwurzel angenähert, aber mit beliebig vorgegebener Genauigkeit mit Hilfe von aufeinander folgenden Divisionen, Multiplikationen und Subtraktionen zu ziehen gestattet.

In einem speziellen Zweig der modernen Mathematik (der praktischen Analysis) werden gerade solche Verfahren erarbeitet, welche komplizierte Operationen wie Integration, Differentiation usw. auf die vier Grundrechenarten zurückführen.

In der Mathematik sieht man eine Schar von Aufgaben eines bestimmten Typs als gelöst an, wenn man einen Algorithmus zu ihrer Lösung aufgestellt hat. Das Auffinden solcher Algorithmen ist eine natürliche Aufgabe des Mathematikers. So hat man beispielsweise in der Algebra Algorithmen aufgestellt, mit deren Hilfe man aus den vorgegebenen Koeffizienten einer algebraischen Gleichung ermitteln kann, wieviele verschiedene Wurzeln sie hat, in welcher Vielfachheit diese Wurzeln auftreten und - mit beliebig vorgegebener Genauigkeit - welche numerischen Werte diese Wurzeln haben.

Auch wenn man keinen Algorithmus zur Lösung aller Aufgaben eines Typs besitzt, kann es möglich sein, einzelne Aufgaben dieses Typs zu lösen. Ein solches für den Einzelfall geschaffenes Verfahren kann aber in anderen Fällen durchaus versagen.

Wir wollen uns einen Aufgabentyp etwas näher ansehen, für den es in der Mathematik heute noch keinen Algorithmus gibt.
Wir betrachten alle möglichen diophantischen Gleichungen, d.h. Gleichungen der Form

$$P = 0$$

wobei P ein Polynom mit ganzzahligen Koeffizienten ist.[3] Beispiele solcher Gleichungen sind

$$x^2 + y^2 - z^2 = 0 \qquad , \qquad 6x^{18} - x + 3 = 0$$

Die erste dieser Gleichungen enthält drei, die zweite eine Unbestimmte (im allgemeinen werden Gleichungen mit beliebig vielen Unbekannten betrachtet). Eine solche Gleichung kann ganzzahlige Lösungen besitzen oder auch nicht. Die erste Gleichung besitzt z.B. die ganzzahlige Lösung

$$x = 3, \quad y = 4, \quad z = 5$$

während die zweite Gleichung keine ganzzahlige Lösung besitzen kann; man verifiziert nämlich für jedes ganzzahlige x leicht die Ungleichung

$$6x^{18} > x - 3$$

[3] Näheres über solche Gleichungen findet man z.B. in dem Bändchen "Die Auflösung von Gleichungen in ganzen Zahlen" von A.O. Gelfond, VEB Deutscher Verlag der Wissenschaften, 2. Aufl., Berlin 1960. (Red.)

Auf dem Internationalen Mathematikerkongress in Paris im Jahre 1900 legte der berühmte Göttinger Mathematiker David Hilbert Hilbert eine Liste von 23 schwierigen Problemen vor und lenkte die Aufmerksamkeit der mathematischen Öffentlichkeit auf die Wichtigkeit ihrer Lösung. Unter ihnen befand sich auch das folgende Problem (das zehnte Hilbertsche Problem):

Es ist ein Algorithmus aufzustellen, mit dessen Hilfe man von einer beliebigen diophantischen Gleichung aussagen kann, ob sie eine ganzzahlige Lösung besitzt oder nicht.

Für den Spezialfall der diophantischen Gleichungen mit einer einzigen Unbekannten ist ein solcher Algorithmus längst bekannt.
Wenn nämlich eine Gleichung mit ganzzahligen Koeffizienten

$$a_n x^n + a_{n-1} x^{n-1} + ... + a_1 x + a_0 = 0$$

eine ganzzahlige Wurzel x_0 haben soll, so muss a_0 durch x_0 teilbar sein. Dementsprechend lässt sich nun folgender Algorithmus angeben:

1. Bestimme alle Teiler der Zahl a_0 (es sind endlich viele).
2. Setze jeden Teiler in die linke Seite der Gleichung ein und berechne deren numerischen Wert.
3. Nimmt die linke Seite für einen gewissen Teiler den Wert Null an, so ist dieser Teiler eine Wurzel der Gleichung; wenn aber die linke Seite der Gleichung für jeden Teiler von Null verschieden ist, so besitzt die Gleichung keine ganzzahlige Lösung.

Das zehnte Hilbertsche Problem beschäftigte und beschäftigt auch weiterhin viele hervorragende Mathematiker, hat aber bis auf den heutigen Tag allen Angriffen standgehalten. Für Gleichungen mit zwei oder mehr Unbekannten konnte ein Algorithmus noch nicht ermittelt werden. Es ist sogar sehr wahrscheinlich, dass man einen solchen Algorithmus auch in Zukunft nicht finden wird. Der genaue Sinn dieser auf den ersten Blick pessimistischen Prognose wird dem Leser erst aus den späteren Darlegungen klar werden.

Die folgenden Eigenschaften der numerischen Algorithmen, die in den bisher betrachteten Beispielen genügend klar erkennbar sein dürften, sind auch für jeden anderen Algorithmus charakteristisch:

Die Determiniertheit eines Algorithmus.
Es wird gefordert, dass man das Rechenverfahren einer anderen Person in Form endlich vieler Anweisungen mitteilen kann, die besagen, wie man in den einzelnen Stadien der Rechnung vorzugehen hat. Hierbei hängt die Rechnung gemäß diesen Anweisungen nicht von dem Rechner ab und ist ein wohldeterminierter Prozess, der zu beliebiger Zeit wiederholt und mit demselben Ergebnis auch von anderen Personen durchgeführt werden kann.

Die Allgemeinverwendbarkeit eines Algorithmus.
Ein Algorithmus ist eine einheitliche Vorschrift, die einen Rechenprozess bestimmt, der bei verschiedenen Ausgangswerten beginnen kann und in allen Fällen zum entsprechenden Resultat führt. Ein Algorithmus dient also nicht nur zur Lösung einer individuellen Aufgabe, sondern zur Lösung einer ganzen Schar von Aufgaben gleichen Typs.

2 Algorithmen zur Lösung logischer Aufgaben

Die Aufgaben des vorhergehenden Paragraphen waren der Arithmetik, der Algebra und der Zahlentheorie entnommen. Sie sind für die Problematik dieser mathematischen Gebiete geradezu typisch und besitzen sozusagen traditionellen mathematischen Charakter.

Wir werden jetzt zwei Scharen von Aufgaben etwas näher betrachten, die völlig anders geartet sind. Man könnte kurz sagen, dass sie logischer statt mathematischer Natur sind, obwohl es ziemlich schwierig sein dürfte, ein exaktes Kriterium anzugeben, nach welchem ähnliche logische Aufgaben von den üblichen mathematischen zu unterscheiden sind.
Ohne uns auf weitere terminologische Betrachtungen einzulassen, wollen wir nur bemerken, dass es sich in beiden Fällen darum handelt, Algorithmen zu finden, d.h. Vorschriften, nach denen jeweils wieder ganze Scharen von Aufgaben gleichen Typs gelöst werden können. Bei den in diesem Paragraphen zu betrachtenden Fällen geht es jedoch nicht um numerische Algorithmen.

2.1 Das Fünfzehnerspiel

Auf einem quadratischen Brett sind 16 gleichgroße Felder eingezeichnet. Fünfzehn mit den Zahlen 1 bis 15 nummerierte Täfelchen sind in beliebiger Weise auf die Felder verteilt (aber so, dass höchstens ein Täfelchen auf einem Feld liegt).
Eine solche Konstellation soll eine Stellung genannt werden. Zwei Felder wollen wir als benachbart bezeichnen, wenn sie längs einer Strecke aneinandergrenzen. Zwei Stellungen sollen benachbart heißen, wenn sie durch einen einzigen Zug ineinander übergeführt werden können, d.h., wenn sie durch Verschieben eines Täfelchen in ein benachbartes leeres Feld auseinander hervorgehen.

Die Anzahl aller möglichen Stellungen ist endlich, nämlich gleich

$$16! = 20922789888000$$

wobei in jeder Stellung nur endlich viele Züge (zwei, drei oder vier) möglich sind.
Damit ergibt sich folgende Schar von Aufgaben gleichen Typs:

Von zwei beliebigen Stellungen A und B ist festzustellen, ob sie mittels endlich vieler Züge ineinander übergeführt werden können.

Wenn die Antwort auf die gestellte Frage bejahend ausfällt, so erzeugen die endlich vielen Züge eine Kette von Stellungen

$$A \leftrightarrow A_1 \leftrightarrow A_2 \leftrightarrow ... \leftrightarrow A_n \leftrightarrow B$$

die von A nach B führt. Der Pfeil bezeichnet jeweils einen Zug, der eine Stellung in eine ihr benachbarte überführt. Man kann annehmen, dass sich in dieser Kette keine Stellung wiederholt, weil sonst die ganze Kette zwischen den sich wiederholenden Stellungen weggelassen werden könnte und man eine kürzere Kette erhielte, die ebenfalls von A nach B führt.

Wenn also die Antwort bejahend ist, wird die Anzahl der erforderlichen Züge nicht größer als 16! - 1 sein.
Von diesen Überlegungen ausgehend kann man jetzt für das gestellte Problem folgenden Algorithmus formulieren, der auf einer ganz einfachen Idee beruht.

Man stellt alle möglichen Kombinationen von 1, 2, 3, ..., 16! - 1 Zügen auf und fällt dann die Entscheidung. Es wird also eine Liste von Stellungen angefertigt, die folgendes Aussehen hat:

Die Stellung A, die zu A benachbarten Stellungen, die Stellungen, die zu diesen benachbarten Stellungen benachbart sind und so weiter, höchstens $(16! - 1)$-mal. Wenn in dieser Liste die Position B auftaucht, so ist die Antwort bejahend, tritt sie nicht auf, so fällt die Antwort

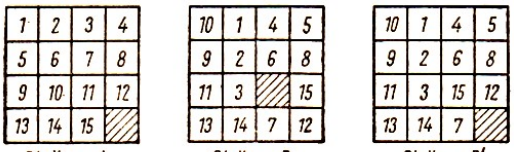
verneinend aus.

Obwohl der vorgeschlagene Lösungsweg sehr mühevoll ist, müssen wir die Frage, ob wir eine einzige Vorschrift besitzen, mit der wir eine beliebige Aufgabe des betrachteten Typs nach endlich vielen Schritten lösen können, mit ja beantworten. Es handelt sich hier offenbar um die potentielle Durchführbarkeit (die praktische Durchführbarkeit hängt von den Mitteln ab, die dem Rechner zur Verfügung stehen) des beschriebenen Prozesses, der zwar endlich ist, aber sehr lang sein kann.

Obwohl dieser Algorithmus nicht gerade sehr reizvoll ist, ist wichtig, dass er angegeben werden konnte. Für das zehnte Hilbertsche Problem konnte ja bisher überhaupt noch kein Algorithmus gefunden werden!
Wenn man erst einen Algorithmus hat, sei er noch so kompliziert, so kann man hoffen, dass man ihn verbessern oder dass man einen einfacheren finden kann. In unserem Beispiel ist dies auch der Fall. Wir wollen jetzt einen anderen Algorithmus beschreiben, der wesentlich einfacher zu handhaben ist. Zu seiner Begründung benötigen wir jedoch einen Satz, den wir ohne Beweis anführen müssen.

Die Vertauschung der Plätze zweier Täfelchen auf dem Brett wollen wir als Transposition bezeichnen. Wir überzeugen uns zunächst davon, dass man durch Verschiebungen und Transpositionen zwei beliebige Stellungen A und B ineinander überführen kann. Wenn nämlich in diesen Stellungen die leeren Felder nicht übereinstimmen, kann man B durch (höchstens 15) Verschiebungen[4] in eine Stellung B' mit dem gleichen Leerfeld wie A überführen.
Der Übergang von B' zu A ist mittels (höchstens 15) Transpositionen zu erreichen. Die Richtigkeit unserer Behauptung ergibt sich offenbar aus der Analyse irgendeines Beispiels. Es mögen die in Abb. 1 dargestellten Stellungen vorliegen.

Abb. 1



Wir verschieben in B das 15-te und das 12-te Täfelchen und gelangen so zu der Stellung B'. Dann bringen wir das Täfelchen mit der Nummer 1 mittels der Transposition (1, 10) in das der Stellung A entsprechende Feld; danach werden die Transpositionen (2, 10), (3, 4), (4, 5), (5, 9), (6, 10), (7, 10), (9, 11), (10, 11) und (11, 15) ausgeführt. Da sich die Täfelchen 8, 12, 13, 14, 15 bereits auf den entsprechenden Plätzen befinden, sind keine weiteren Operationen mehr notwendig.

Wir formulieren jetzt den vorhin erwähnten Satz, ohne ihn zu beweisen.

Satz. Kann man die Stellung B mittels Verschiebungen und einer geraden Anzahl von Transpositionen in die Stellung A überführen, so ist B auch durch Verschiebungen allein in A überführbar.
Kann man B mittels Verschiebungen und einer ungeraden Anzahl von Transpositionen in A überführen, dann ist B auf keine Weise durch bloße Verschiebungen in A zu überführen.

Damit ist aber auch schon klar, wie der Algorithmus für unser Problem aussieht. Entsprechend dem oben angegebenen Verfahren überführen wir mit Hilfe von Verschiebungen und Transpo-

[4] Man kommt sogar mit höchstens 6 Verschiebungen aus.

sitionen (insgesamt nicht mehr als $15 + 15 = 30$ Operationen) die Position A in die Position B.
Wenn dazu eine gerade Anzahl von Transpositionen notwendig war, so ist die gestellte Frage bejahend zu beantworten, im anderen Falle verneinend. In unserem Beispiel ist die Anzahl der Transpositionen gerade und somit die Stellung B tatsächlich in die Stellung A überführbar.

2.2 Das Irrgartenproblem

Die griechische Mythologie berichtet von dem legendären Helden Theseus, dass er sich in ein Labyrinth begab, um dort das Ungeheuer Minotaurus aufzuspüren und zu erschlagen. Bei diesem Vorhaben half ihm Ariadne, indem sie ihm ein Fadenknäuel mitgab, dessen Ende sie in der Hand behielt, damit Theseus den Rückweg finden konnte. Beim Eindringen in das Labyrinth wickelte Theseus das Knäuel ab; indem er den Faden wieder aufwickelte, gelangte er wohlbehalten zum Ausgang zurück.

An diese alte Legende erinnert das kürzlich entwickelte automatische Spielzeug "Maus im Labyrinth" des amerikanischen Mathematikers und Ingenieurs Claude Shannon. An einer Stelle eines speziellen Labyrinths befindet sich ein Stück "Speck" und an einer anderen die "Maus". Die Maus irrt planlos im Labyrinth umher, bis sie den "Speck" gefunden hat.

Hierbei kommt sie an einigen Orten mehrmals vorbei, d.h., sie läuft in Schlaufen zum "Speck". Lässt man die "Maus" aber zum zweitenmal von derselben Stelle aus starten, so läuft sie direkt, ohne eine Schlaufe zu machen, zum "Futter".

Wir wollen hier ein verwandtes Problem (genauer eine Schar von Aufgaben gleichen Typs) untersuchen:
Wie findet man den Weg in einem Labyrinth? Wir werden einen Algorithmus angeben, der beschreibt, wie man zur Erreichung des in der Aufgabe gestellten Ziels vorgehen kann.

Wir stellen uns das Labyrinth als endliches System von Plätzen vor, von denen Gänge ausgehen, wobei jeder Gang zwei Plätze verbindet (solche Plätze sollen benachbart heißen); es kann aber solche Plätze geben, zu denen man nur längs eines einzigen Ganges gelangen kann; dann sprechen wir von Sackgassen.
Geometrisch lässt sich ein solches Labyrinth als ein System von Punkten (den Bildern der Plätze) A, B, C, ... und einer Gesamtheit von Strecken (den Bildern der Gänge) AB, BC, ..., die gewisse Paare von Punkten miteinander verbinden, darstellen (Abb. 2). Eine solche Figur nennt man auch einen Graphen.

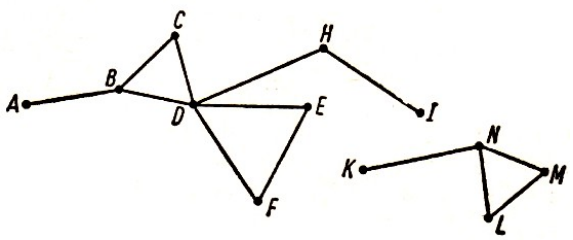


Abb. 2

Wir wollen sagen, dass ein Platz Y von einem Platz X aus erreichbar ist, wenn es einen Weg gibt, der längs gewisser Gänge und über gewisse Plätze von X nach Y führt. Genauer gesagt heißt das, dass X und Y entweder benachbarte Plätze sind oder dass eine Folge von Plätzen X_1, X_2, X_3, ..., X_n existiert derart, dass X und X_1, ferner X_1 und X_2, ferner X_2 und X_3, ..., und schließlich X_n und Y benachbart sind.

So ist in Abb. 2 beispielsweise der Platz H von der Sackgasse A aus längs des Weges AB, BC, CD, DE, EF, FD, DH erreichbar, während K von A aus nicht erreichbar ist.

Wenn Y von X aus überhaupt erreichbar ist, so auch längs eines einfachen Weges, d.h. längs eines solchen Weges, in dem jeder Platz (und erst recht auch jeder Gang) höchstens einmal vorkommt. Unser Beispiel war kein einfacher Weg. Schneidet man jedoch die Schlaufe DE, EF, FD ab, so ergibt sich der einfache Weg AB, BC, CD, DH.

Wir nehmen an, der Minotaurus halte sich auf einem Platz M des Labyrinths auf. Theseus, der sich zu Beginn auf dem Platz A befindet, auf dem Ariadne auf ihn wartet, hat die folgende Aufgabe zu lösen:
Er hat festzustellen, ob M von A aus erreichbar ist oder nicht.[5] Wenn M erreichbar ist, so muss Theseus auf irgendeinem Wege dorthin gelangen, aber auf einem einfachen Wege zu Ariadne zurückkehren. Ist aber M nicht erreichbar, so kehrt Theseus zu Ariadne zurück.

Es gibt unendlich viele verschiedene Labyrinthe, wobei auch die gegenseitige Lage der Plätze A und M noch variieren kann. Da Theseus im voraus nichts über den Aufbau des vorliegenden Labyrinths und über den Aufenthaltsort des Minotaurus weiß, ist die Aufgabe folgendermaßen zu verstehen:
Es soll ein allgemeines Verfahren gefunden werden, welches für beliebige Labyrinthe und beliebige Anordnungen der Plätze A und M brauchbar ist. Mit anderen Worten: Es wird ein Algorithmus gesucht, der auf beliebige Aufgaben des gegebenen Typs anwendbar ist.

Um diesen allgemeinen Algorithmus zu konstruieren, betrachten wir zunächst ein spezielles Suchverfahren. In jedem Stadium des Suchprozesses soll bei diesem Verfahren festgestellt werden können, ob Theseus einen Gang überhaupt noch nicht durchlaufen (solche Gänge seien mit grün gekennzeichnet), einmal durchlaufen (gelb) oder zweimal durchlaufen (rot) hat. Befindet sich Theseus auf irgendeinem Platz, so kann er zu einem benachbarten Platz auf eine der beiden folgenden Arten gelangen:

1. Abwickeln des Fadens. Theseus geht von dem vorgegebenen Platz durch einen grünen Gang zu dem benachbarten Platz. (Hierbei wird der Faden der Ariadne längs dieses Ganges, der nach dem Durchlaufen mit "gelb" gekennzeichnet wird, abgewickelt.)

2. Aufwickeln des Fadens. Theseus kehrt von dem erreichten Platz auf dem zuletzt durchlaufenen gelben Gang zu dem benachbarten Platz zurück. Hierbei wird der Faden der Ariadne, der vorher längs dieses Ganges abgewickelt worden war, wieder aufgewickelt. Dieser Gang ist dann rot gekennzeichnet.

Es soll weiterhin angenommen werden, dass Theseus keinerlei Kennzeichen hinterlässt, die es ihm gestatten könnten, später einen roten von einem grün markierten Gang zu unterscheiden. Ein gelber Gang ist ja dadurch ausgezeichnet, dass durch ihn der Faden der Ariadne läuft.
Wie Theseus sich in jedem einzelnen Stadium des Suchens zu verhalten hat, hängt von den Umständen ab, die er auf dem Platz vorfindet, auf dem er sich gerade aufhält. Diese Umstände lassen sich jeweils durch eines oder mehrere der folgenden Merkmale charakterisieren:

1. Minotaurus. Auf dem erreichten Platz befindet sich der Minotaurus.

2. Schlaufe. Über den erreichten Platz läuft schon der Faden der Ariadne; mit anderen Worten, von dem erreichten Platz gehen wenigstens zwei gelbe Gänge ab.

[5] Natürlich ist M von A verschieden.

3. Grüner Gang. Der erste in Uhrzeigerrichtung anzutreffende Ausgang von dem erreichten Platz ist grün markiert.

4. Ariadne. Auf dem erreichten Platz befindet sich Ariadne.

5. Fünfter Fall. Es fehlen alle erwähnten Merkmale.

Unser Suchverfahren kann man jetzt in folgendem Schema angeben:

Kennzeichen	Operation
1. Minotaurus	Halt
2. Schlaufe	Faden aufwickeln
3. Grüner Gang	Faden abwickeln
4. Ariadne	Halt
5. Fünfter Fall	Faden aufwickeln.

Ist Theseus auf einem Platz angekommen, so wird er wie folgt verfahren:
Er geht nacheinander die einzelnen Kennzeichen in der vorgeschriebenen Reihenfolge der linken Seite des Schemas durch und kontrolliert dabei, welches zutrifft. Nachdem er das erste zutreffende Kennzeichen ermittelt hat (die anderen prüft er dann nicht mehr), führt er die in der rechten Spalte stehende Operation aus. Dieses Verfahren setzt er solange fort, bis er zur Operation Halt kommt.
Die Brauchbarkeit der vorgeschlagenen Methode ergibt sich unmittelbar aus den folgenden drei Behauptungen:

1. Bei beliebiger gegenseitiger Lage von A und M im Labyrinth kommt Theseus nach endlich vielen Operationen entweder zum Minotaurus oder zu Ariadne, was ja die Operation Halt zur Folge hat.

2. Tritt das Halt auf dem Platz des Minotaurus ein, dann ist dieser natürlich erreichbar. Ja, mehr noch, der Faden der Ariadne ist längs eines von A nach M führenden einfachen Weges gezogen, und Theseus kann auf diesem Weg, indem er den Faden aufwickelt, zu Ariadne zurückkehren.

3. Tritt das Halt auf dem Platz der Ariadne ein, so ist der Minotaurus unerreichbar.

Bevor wir diese Behauptungen beweisen, zeigen wir an zwei Beispielen, wie die vorgeschlagene Methode gehandhabt wird.

Beispiel 1. Auf dem Platz A des Labyrinths (Abb. 2) beginne das Suchen nach dem Minotaurus, der sich in F aufhalte. Den Suchprozess kann man nach unserem Verfahren bequem in einer Tabelle beschreiben (siehe Tabelle 1, siehe nächste Seite). Es handelt sich jedoch nur um eines der möglichen Schemata, weil bei der Auswahl eines grünen Ganges verschiedene Möglichkeiten vorliegen können.

Wir sehen, dass der Minotaurus im vorliegenden Fall erreichbar ist. Wir suchen uns nun aus der vorletzten Spalte (unter Beachtung der Angaben in der letzten Spalte) diejenigen Gänge heraus, die gelb geblieben sind, und erhalten so einen einfachen Weg, der von A nach F führt:

$$AB, BC, CD, DF.$$

Beispiel 2. Das Suchen beginne auf dem Platz K. Das Schema des Suchprozesses ist in Tabelle 2 (nächstes Seite) dargestellt.

In diesem Fall ist also der Minotaurus unerreichbar.

Tabelle 1

Nr. der Operation	Vorgefundenes Kennzeichen	Operation	Zu durchlaufender Gang	Farbe des Ganges nach dem Durchlaufen
1	Grüner Gang	Abwickeln	AB	gelb
2	Grüner Gang	Abwickeln	BC	gelb
3	Grüner Gang	Abwickeln	CD	gelb
4	Grüner Gang	Abwickeln	DH	gelb
5	Grüner Gang	Abwickeln	HI	gelb
6	Fünfter Fall	Aufwickeln	IH	rot
7	Fünfter Fall	Aufwickeln	HD	rot
8	Grüner Gang	Abwickeln	DB	gelb
9	Schlaufe	Aufwickeln	BD	rot
10	Grüner Gang	Abwickeln	DF	gelb
11	Minotaurus	Halt	-	-

Tabelle 2

Nr. der Operation	Vorgefundenes Kennzeichen	Operation	Zu durchlaufender Gang	Farbe des Ganges nach dem Durchlaufen
1	Grüner Gang	Abwickeln	KN	gelb
2	Grüner Gang	Abwickeln	NL	gelb
3	Grüner Gang	Abwickeln	LM	gelb
4	Grüner Gang	Abwickeln	MN	gelb
5	Schlaufe	Aufwickeln	NM	rot
6	Fünfter Fall	Aufwickeln	ML	rot
7	Fünfter Fall	Aufwickeln	LN	rot
8	Fünfter Fall	Aufwickeln	NK	rot
9	Ariadne	Halt	-	-

Wir wollen uns jetzt dem Beweis der Behauptungen 1 bis 3 zuwenden.

Beweis der Behauptung 1. Zuerst zeigen wir mittels vollständiger Induktion[6] nach der Anzahl der Operationen des Theseus, dass in jedem Stadium des Suchprozesses die folgende Alternative gilt (d.h., es gilt einer der beiden folgenden, sich gegenseitig ausschließenden Fälle):

a) Im Labyrinth gibt es keinen gelben Gang; dann befindet sich Theseus auf dem Platz A (Ariadne).

b) Im Labyrinth gibt es gelbe Gänge, die, in derselben Reihenfolge betrachtet, wie sie von Theseus durchlaufen worden waren, einen Weg bilden, der von A zu dem Platz führt, auf dem sich Theseus befindet.

Außerdem wird sich zeigen, dass Theseus niemals durch einen roten Gang geht.
Die Behauptung stimmt zu Beginn des Prozesses, wenn sich Theseus auf dem Platz A befindet und noch keinen Gang durchlaufen hat, offenbar (alle Gänge sind grün). Wir wollen nun annehmen, dass die angegebene Alternative bis zur $(n-1)$-ten Operation richtig ist, und beweisen, dass die Alternative auch nach der n-ten Operation gilt (selbstverständlich soll die $(n-1)$-te Operation nicht gerade Halt gewesen sein, denn sonst gäbe es keine n-te Operation mehr).

[6] Über Induktionsschlüsse vgl. etwa I.S. Sominski, Die Methode der vollständigen Induktion, VEB Deutscher Verlag der Wissenschaften, 5. Aufl., Berlin 1964. (Red.)

Es liege nach der $(n-1)$-ten Operation der Fall a) vor. Dann wird die nächste Operation entweder das Durchlaufen eines grünen Ganges von A zu einem benachbarten Platz K sein (nach der n-ten Operation gilt der Fall b) mit dem gelben Gang AK) oder Halt auf dem Platz A (nach der n-ten Operation bleibt der Fall a) bestehen).

Wir wollen jetzt annehmen, dass nach der $(n-1)$-ten Operation der Fall b) mit s gelben Gängen vorliegt. Die gelben Gänge beschreiben dann den Weg AA_1, A_1A_2, ..., $A_{s-1}K$. Je nachdem, welches Kennzeichen Theseus nun am Platz K vorfindet, ergeben sich für die n-te Operation folgende Möglichkeiten:

1. Minotaurus. Theseus hat die Operation Halt auf dem Platz K auszuführen, und die früheren gelben Gänge bleiben erhalten (es liegt der Fall b) nach der n-ten Operation vor).

2. Schlaufe. Theseus wickelt den Faden auf, d.h., er läuft durch den Gang KA_{s-1} der jetzt rot wird, zurück. Der gelbe Weg verkürzt sich also um einen Gang. Wenn die Anzahl s der Gänge des früheren Weges größer als 1 war, liegt nach der n-ten Operation der Fall b) mit $s-1$ gelben Gängen vor. War jedoch $s=1$, so gilt der Fall a).

3. Grün. Theseus wickelt den Faden ab, d.h., er geht durch einen grünen Gang, der jetzt gelb wird. Es trifft also sicher der Fall b) mit $s+1$ gelben Gängen zu.

4. Ariadne. Nach diesem Kennzeichen wird Theseus sich nicht richten, weil er sich sonst längs des gelben Weges

$$AA_1, A_1A_2, ..., A_{s-1}K$$

(d.h., wenn $K = A$ ist) zu Ariadne zurückbewegt hätte. Dann hätte er sich gemäß dem vorgeschlagenen Suchverfahren nach dem Kennzeichen Schlaufe gerichtet.

5. Fehlen die ersten vier Kennzeichen, so wird der Faden aufgewickelt. Hierbei ergibt sich die gleiche Konstellation wie im Fall des Kennzeichens Schlaufe. Wir kommen auf den Fall a) (für $s=1$) oder den Fall b) (für $s>1$).

Damit ist die oben angegebene Alternative gerechtfertigt.
Gleichzeitig ist auch klar, dass Theseus keinen Gang mehr als zweimal durchläuft (d.h., er kommt nie durch einen roten Gang). Da aber die Anzahl aller Gänge des Labyrinths endlich ist, muss auch die Folge der Operationen endlich sein. Diese Folge kann jedoch nur bei Halt abbrechen, d.h., wenn Theseus sich entweder auf dem Platz des Minotaurus oder auf dem der Ariadne befindet.

Beweis der Behauptung 2. Tritt die Operation Halt auf dem Platz des Minotaurus auf, so ist dieser offensichtlich erreichbar.
Hierbei ist der Faden der Ariadne längs eines gelben Weges gezogen, dessen Existenz gerade oben nachgewiesen wurde. Der Faden kann auch keine Schlaufen haben, denn jedesmal, wenn Theseus beim Umherirren das Kennzeichen Schlaufe findet, läuft er wieder zurück und beseitigt sie.

Beweis der Behauptung 3. Im Fall der Operation Halt auf dem Platz der Ariadne kann folgendes vorliegen:

1. Für jeden Gang des Labyrinths gilt: Er wurde entweder zweimal (roter Gang) oder gar nicht durchlaufen (grüner Gang). Mit anderen Worten, das Knäuel ist vollständig aufgewickelt, es gibt im Labyrinth keinen gelben Gang.[7]

[7] Der Beweis dieser Tatsache bleibe dem Leser überlassen.

2. Alle Gänge, die von A ausgehen, sind rot; denn das Kennzeichen Ariadne kann entsprechend dem Schema nur dann berücksichtigt werden, wenn alle drei im Schema früher auftretenden Kennzeichen, insbesondere auch grün, nicht zutreffen.

Wir wollen jetzt das Gegenteil der Behauptung 3 annehmen.
Der Minotaurus sei erreichbar, und AA_1, A_1A_2, ..., A_nM sei ein Weg, der von A nach M führt. Der erste Gang dieses Weges ist rot, weil er von A ausgeht; der letzte ist grün, weil Theseus beim Suchen nicht bis zum Minotaurus gekommen ist.

Es sei A_iA_{i+1} der erste grüne Gang in dieser Folge. Nach A_i führen also grüne und rote Gänge. Untersuchen wir einmal den letzten Durchgang des Theseus durch A_i. Offenbar kam er durch einen roten Gang, der an A_i angrenzt. Das könnte aber nur durch Aufwickeln des Fadens geschehen sein, weil er den Gang zweimal durchlief.
Folglich hätte Theseus entweder das Kennzeichen Schlaufe oder das Kennzeichen Fünfter Fall vorfinden müssen.

Das letztere ist aber unmöglich, weil von A_i der grüne Gang A_iA_{i+1} abgeht. Also muss man annehmen, dass Theseus in A_i eine Schlaufe vorfand.
Dies führt aber unmittelbar auf einen Widerspruch, womit dann die Behauptung 3 vollständig bewiesen ist.

Hätte Theseus in A_i eine Schlaufe vorgefunden, so würden von A_i zwei gelbe Gänge wegführen. Bei der nächsten Operation hätte er einen dieser gelben Gänge in einen roten verwandelt. Den anderen gelben Gang hätte er später noch einmal durchlaufen müssen, weil kein gelber Gang existiert.
Folglich müsste Theseus den Platz A_i noch einmal betreten haben. Das widerspricht aber der Annahme, dass wir den letzten Durchgang des Theseus durch A_i betrachtet haben.

Wir müssen noch darauf hinweisen, dass in der von uns vorgeschlagenen Vorschrift noch ein willkürliches Element steckt.
Beim Kennzeichen grün ist die nächste Operation nämlich nicht eindeutig bestimmt, wenn von dem gegebenen Platz mehrere grüne Gänge wegführen. Unsere Vorschrift lässt offen, welchen man zu beschreiten hat. Sie gestattet, genauer ausgedrückt, die willkürliche Auswahl eines Ganges. Dadurch geht aber die Eigenschaft der Determiniertheit verloren, von der gerade im vorhergehenden Paragraphen gesagt wurde, dass sie allen Algorithmen charakteristisch ist.

Dieses zufällige Element könnte man leicht durch eine Vereinbarung beseitigen (und dadurch die angegebene Vorschrift in einen Algorithmus verwandeln), nach der beim Vorhandensein mehrerer grüner Gänge der zu begehende Gang auf Grund irgendeiner Regel ausgesucht wird. Beispielsweise könnte man verlangen, dass Theseus den betretenen Platz im Uhrzeigersinn bis zum ersten grünen Gang umläuft und diesen Gang entlang geht.
Genauso gut könnte man natürlich irgendeine andere Vereinbarung treffen.

Die Untersuchung solcher Vorschriften, in denen man das Auftreten willkürlicher Elemente voraussehen kann, ist von großem theoretischen und praktischen Interesse, besonders in der modernen Theorie der Spiele. Wir werden uns jedoch mit diesen Fragen nicht weiter beschäftigen und nur streng determinierte Prozesse behandeln.

Zum Abschluss dieses Paragraphen wollen wir die beiden Aufgaben noch einmal vergleichen und einen bestimmten Zusammenhang zwischen ihnen feststellen.
Trotz ihrer äußeren Unterschiede sind diese Aufgaben völlig gleichartig; genauer gesagt, die erste ist ein Spezialfall der zweiten. In der Tat, wenn jede Stellung des "Fünfzehnerspiels" einen

Platz und jeder Übergang von einer Stellung zu einer benachbarten einen Gang darstellt, der die benachbarten Plätze (Stellungen) verbindet, so ist die erste Aufgabe dieses Paragraphen ein Irrgartenproblem in einem speziellen Labyrinth mit 16! Plätzen, von denen jeder durch Gänge mit zwei, drei oder vier benachbarten Plätzen verbunden ist.

Dann kann aber der von uns vorgeschlagene Algorithmus auch auf das "Fünfzehnerspiel" übertragen werden. Wie man leicht sieht, handelt es sich einfach um eine etwas vervollkommnete Variante des Algorithmus, der von uns zuerst für das "Fünfzehnerspiel" angegeben worden war. Die Vervollkommnung besteht darin, dass die überflüssigen Wiederholungen ausgeschaltet wurden.

Für speziellere Labyrinthe (wie das "Fünfzehnerspiel") lassen sich oft auch spezielle, einfachere Algorithmen finden.

Dagegen kann natürlich im allgemeinen Fall ein Algorithmus, der auf beliebige Labyrinthe anwendbar sein soll, kaum anders beschaffen sein, als dass in gewisser Weise alle Möglichkeiten durchgemustert werden. Man kann daher nur schwerlich auf die Schaffung eines Algorithmus hoffen, der einfacher ist als der, den wir vorgeschlagen haben.

3 Das Wortproblem

Das Wortproblem ist eine weitgehende Verallgemeinerung des "Fünfzehnerspiels" und des Suchproblems des Theseus. Während das Fünfzehnerspiel das Suchen in einem speziellen, vorher festgelegten endlichen Irrgarten ist und das Problem des Theseus einen beliebigen endlichen Irrgarten betrifft, kann das Wortproblem in gewissem Sinne als Suchen in einem unendlichen Irrgarten aufgefasst werden.

Das Wortproblem entstand in speziellen Teilen der modernen Algebra, der sogenannten Theorie der Halbgruppen (assoziativen Systeme) und der Gruppentheorie. Seine Bedeutung geht jedoch weit über den Rahmen dieser speziellen Theorien hinaus.
Verschiedene Varianten dieses Problems wurden mit Erfolg von den hervorragenden sowjetischen Mathematikern Andrej Andrejewitsch Markow und Pjotr Sergejewitsch Nowikow sowie ihren Schülern bearbeitet.

Zuerst wollen wir einige vorbereitende Begriffe einführen:

Als Alphabet bezeichnen wir ein beliebiges endliches System paarweise verschiedener Zeichen, die wir Buchstaben dieses Alphabets nennen. So kann man beispielsweise das Alphabet $\{\alpha, \aleph, z, ?\}$ betrachten, welches aus dem griechischen Buchstaben α, dem hebräischem $\aleph$, dem lateinischen z und dem Fragezeichen besteht.[8]
Ein Wort in einem gegebenen Alphabet ist eine beliebige Folge von Buchstaben dieses Alphabets. Zum Beispiel sind $abaa$ und $bbac$ Wörter im Alphabet $\{a, b, c\}$.

Wenn ein Wort L Teil eines Wortes M ist, so sagen wir, das Wort L sei im Wort M (als Buchstabengruppe) eingebettet. So ist z. B. im Wort $abcbcbab$ zweimal das Wort bcb eingebettet; das eine beginnt mit dem zweiten und das andere mit dem vierten Buchstaben.
Wir werden die Transformation eines Wortes in ein anderes mittels gewisser zulässiger Substitutionen (Ersetzungen) betrachten, die wir in der Form

$$P - Q \qquad \text{bzw.} \qquad P \to Q$$

[8] In dieser Abschrift wurde das kyrillische "Z" durch das hebräische $\aleph$ ersetzt.

schreiben, wobei P, Q zwei Wörter ein und desselben Alphabets sind.

Die Anwendung der orientierten Substitution $P \to Q$ auf ein Wort R ist dann möglich, wenn in diesem Wort die Buchstabengruppe P wenigstens einmal eingebettet ist; dabei darf jede einzelne Einbettung durch die entsprechende rechte Seite Q ersetzt werden. Bei dieser Substitution tritt dann an Stelle des P das Wort Q.

Die Anwendung einer nichtorientierten Substitution $P - Q$ lässt sowohl das Ersetzen der linken Seite durch die rechte als auch der rechten Seite durch die linke zu,[9] Zunächst werden wir vorzugsweise nichtorientierte Substitutionen betrachten und sie dort, wo keine Verwechslungen möglich sind, einfach als Substitutionen bezeichnen.

Beispiel 1. Die Substitution $ab - bcb$ ist in vierfacher Weise auf das Wort $abcbcbab$ anwendbar. Wenn wir eine der beiden Buchstabengruppen bcb durch ab ersetzen, erhalten wir die Wörter

$$aabcbab \qquad \text{bzw.} \qquad abcabab$$

die Substitution einer der beiden eingebetteten Buchstabengruppen ab durch bcb liefert die Wörter

$$bcbcbcbab \qquad \text{bzw.} \qquad abcbcbbcb$$

Auf das Wort $bacb$ ist diese Substitution nicht anwendbar.

Die Gesamtheit aller Wörter eines Alphabets mit einem gewissen endlichen System zulässiger Substitutionen wollen wir Halbgruppe[10] nennen.
Um eine Halbgruppe vorzugeben, genügt es, das entsprechende Alphabet und ein System von Substitutionen anzugeben.

Wenn man ein Wort B durch eine einmalige Anwendung einer zulässigen Substitution in ein Wort S überführen kann, so kann auch S auf dieselbe Weise in R übergeführt werden. Dann werden B und S als benachbarte Wörter bezeichnet. Eine Folge von Wörtern

$$R_1, R_2, ..., R_{n-1}, R_n$$

die derart beschaffen ist, dass R_1 und R_2, R_2 und R_3, ..., R_{n-1} und R_n benachbart sind, soll eine deduktive Kette genannt werden, die von R_1 nach R_n führt.
Existiert eine deduktive Kette, die von einem Wort B zu einem Wort S führt, so gibt es offenbar auch eine deduktive Kette von S nach B. In diesem Fall sollen R und S als äquivalent bezeichnet werden, in Zeichen $R \sim S$.

Weiterhin ist klar, dass aus $S \sim R$ und $R \sim T$ auch $S \sim T$ folgt. Für unsere weiteren Überlegungen benötigen wir noch den folgenden Satz.

Satz. Es sei $P \sim Q$. Ist in irgendeinem Wort R das Wort P eingebettet und wird P durch das Wort Q ersetzt, so ist das neu entstandene Wort zu R äquivalent.

Beweis. Es sei also P in R eingebettet; dann lässt sich R in der Form SPT darstellen, wobei S der Teil des Wortes R ist, der vor P steht, und T der Teil hinter P. Das transformierte Wort hat dann die Gestalt SQT. Ferner existiert wegen der Äquivalenz $P \sim Q$ eine deduktive Kette

$$P, P_1, P_2, ..., P_m, Q$$

[9] In der Gruppentheorie werden das Alphabet als Erzeugendensystem und die zulässigen Substitutionen als definierende Relationen bezeichnet.

[10] Im Original wird die Bezeichnung "assoziatives System" verwendet, die aber in der deutschen Literatur wenig gebräuchlich ist.

Dann stellt aber, wie man leicht sieht, die Folge der Wörter.

$$SPT, SP_1T, SP_2T, ..., SP_mT, SQT$$

eine deduktive Kette dar, die von SPT (d.h. von R) zu SQT (d.h. dem transformierten Wort) führt. Der Satz ist damit bewiesen.

Beispiel 2. Wir betrachten nun eine Halbgruppe, die zuerst von G.S. Zeitin untersucht worden ist. Das Alphabet sei

$$\{a, b, c, d, e\}$$

Das System der zulässigen Substitutionen laute

1. $ac - ca$
2. $ad - da$
3. $bc - cb$
4. $bd - db$
5. $abac - abacc$
6. $eca - ae$
7. $edb - be$

In dieser Halbgruppe ist auf das Wort $abcde$ nur die dritte Substitution anwendbar, und es gibt nur ein benachbartes Wort $acbde$. Es gilt ferner die Äquivalenz $abcde \sim cadedb$, wie aus der folgenden deduktiven Kette ersichtlich ist:

$$a\mathbf{bc}de, \mathbf{ac}bde, ca\mathbf{bd}e, cad\mathbf{be}, cadedb$$

Auf das Wort $aaabb$ kann keine der zulässigen Substitutionen angewandt werden, daher existieren keinerlei ihm benachbarte Wörter und mithin auch kein von $aaabb$ verschiedenes Wort, das ihm äquivalent ist.

Für jede Halbgruppe entsteht in natürlicher Weise ein spezielles Äquivalenzproblem. Es lässt sich folgendermaßen formulieren:

Von je zwei Wörtern einer Halbgruppe ist festzustellen, ob sie äquivalent sind oder nicht.

In jeder Halbgruppe gibt es unendlich viele verschiedene Wörter. Folglich handelt es sich hier faktisch um unendlich viele Aufgaben gleichen Typs, zu deren Lösung ein Algorithmus gesucht wird, der die Äquivalenz oder die Nichtäquivalenz beliebiger Wortpaare festzustellen gestattet.

Es kann hier der Eindruck entstehen, als ob das Wortproblem ein künstlich erdachtes Zusammensetzspiel wäre, dessen Lösung mittels eines Algorithmus keinerlei praktischen oder theoretischen Wert hätte. Das ist durchaus nicht der Fall.
Es handelt sich sogar um ein ganz natürliches Problem von großer theoretischer Bedeutung, welche die Anstrengungen völlig rechtfertigt, die zur Konstruktion eines entsprechenden Algorithmus aufgewandt werden. Im jetzigen Stadium unserer Ausführungen können wir diese Fragen noch nicht ausführlich genug erörtern und gehen daher zunächst zur Untersuchung konkreter Tatsachen über.

Erstens wollen wir das Äquivalenzproblem für Wörter mit dem Problem des Theseus in Verbindung bringen. Ordnet man jedem Wort einen "Platz" und jedem Paar benachbarter Wörter einen Gang zu, der die entsprechenden Plätze miteinander verbindet, so ergibt sich, dass die

Halbgruppe die Gestalt eines Irrgartens mit unendlich vielen Plätzen und Gängen annimmt, wobei von jedem Platz nur endlich viele Gänge abgehen.
Es gibt selbstverständlich auch solche Plätze, von denen kein einziger Gang ausgeht (siehe das Wort $aaabb$ im Beispiel 2). Bei dieser Betrachtungsweise wird eine deduktive Kette, die von irgendeinem Wort R zu irgendeinem Wort Q führt, ein Weg in dem Irrgarten sein, der von einem Platz zu einem anderen geht; der Äquivalenz von Wörtern entspricht dann die gegenseitige Erreichbarkeit der betreffenden Plätze. Das Wortproblem wird auf diese Weise zu einem Suchproblem in einem unendlichen Irrgarten.

Damit die Besonderheiten der hier auftretenden Schwierigkeiten besser und klarer hervortreten, wollen wir vorher das eingeschränkte Wortproblem erörtern, welches folgendermaßen lautet:

Von je zwei Wörtern R und T einer vorgegebenen Halbgruppe ist festzustellen, ob sie durch höchstens k-malige Anwendung zulässiger Substitutionen ineinander übergeführt werden können (k ist eine beliebige, aber ein für allemal fest gewählte natürliche Zahl).

Für diese Problemstellung kann man den geforderten Algorithmus leicht konstruieren. Wir können hierbei auf den schon angeführten Durchmusterungsalgorithmus zurückgreifen. Es wird eine Liste aufgestellt, die mit dem Wort R beginnt, dann alle zu R benachbarten Wörter enthält, ferner die benachbarten dieser benachbarten usw., insgesamt k-mal. Die Antwort wird bejahend oder verneinend ausfallen, je nachdem, ob das Wort T in dieser Liste erscheint oder nicht.

Beim uneingeschränkten Wortproblem ist die Lage jedoch wesentlich anders. Da die Länge einer deduktiven Kette, die von R nach T führt, sehr groß sein kann (wenn eine solche existiert), ist es im allgemeinen ungewiss, von welcher Stelle ab man diesen Durchmusterungsprozess als abgeschlossen betrachten darf.
So habe man beispielsweise den Durchmusterungsprozess schon bis zur Stelle

$$10^{20} = 100000000000000000000$$

fortgesetzt und die Liste aller Wörter vor sich liegen, die man aus R mittels mehrfacher Anwendung der zulässigen Substitutionen erhalten kann, deren Anzahl nicht größer als 10^{20} ist, und das Wort T sei in dieser Liste nicht enthalten.

Gibt es nun irgendeinen Grund anzunehmen, dass die Wörter R und T nicht äquivalent sind? Selbstverständlich nicht; denn es besteht ja immer noch die Möglichkeit, dass sie äquivalent sind, jedoch könnte die kleinste deduktive Kette, die R und T verbindet, noch länger sein.

Um hier zu den gewünschten Ergebnissen zu gelangen, muss man von der einfachen Durchmusterung abgehen und andere Methoden heranziehen, die auf einer Analyse des Mechanismus der Transformation von Wörtern in andere mittels der zulässigen Substitutionen beruhen.
Wir wollen beispielsweise untersuchen, ob in der Zeitinschen Halbgruppe (siehe Beispiel 2) die Wörter $abaacd$ und $acbdad$ äquivalent sind. Die verneinende Antwort auf diese Frage ergibt sich aus den folgenden Überlegungen:

In jeder der zulässigen Substitutionen enthalten die linke und die rechte Seite den Buchstaben a gleich oft (oder sie enthalten ihn gar nicht). Daher müssen alle Wörter einer deduktiven Kette den Buchstaben a gleich oft enthalten. Da in den vorgegebenen beiden Wörtern der Buchstabe a verschieden oft auftritt, können sie nicht äquivalent sein, denn es kann keine deduktive Kette zwischen ihnen geben.

Mit Hilfe solcher deduktiver Invarianten, d.h. solcher Eigenschaften, die für alle Wörter einer

deduktiven Kette gelten, kann man in einigen Fällen die geforderten Algorithmen aufstellen.

Beispiel 3. Das Alphabet sei

$$\{a, b, c, d, e\}$$

Das System der zulässigen Substitutionen habe die Gestalt

$$\begin{array}{lllll} ab - ba, & ac - ca, & ad - da, & ae - ea, & be - eb, \\ bd - db, & be - eb, & cd - dc, & ce - ec, & de - ed. \end{array}$$

Diese zulässigen Substitutionen ändern die Anzahl der in einem Worte enthaltenen Buchstaben nicht, sondern nur ihre Stellung im Wort. Es ergibt sich unmittelbar, dass zwei Wörter dann und nur dann äquivalent sein können, wenn jeder Buchstabe in ihnen gleich oft auftritt.
Daher ist der Algorithmus zur Feststellung der Äquivalenz äußerst einfach. Man hat nur nachzuzählen, wie oft jeder Buchstabe in den einzelnen Wörtern vorkommt, und die ermittelten Zahlen miteinander zu vergleichen.

Weiter unten wollen wir noch ein etwas komplizierteres Beispiel betrachten. Zuvor müssen aber noch die Begriffe "Wort" und "zulässige Substitution" etwas verallgemeinert werden.
Außer den gewöhnlichen Wörtern eines vorgegebenen Alphabets wollen wir auch das leere Wort, das keinen Buchstaben enthält, als ein Wort unseres Alphabets betrachten und durch das große griechische Lambda Λ bezeichnen. Außerdem sollen auch Substitutionen der Form

$$P - \Lambda$$

zugelassen werden.

Das Ersetzen der linken Seite durch das leere Wort bedeutet. einfach, dass aus dem zu transformierenden Wort das darin eingebettete Wort P gestrichen wird. Das Ersetzen der rechten Seite durch die linke bedeutet, dass zwischen zwei beliebigen Buchstaben des zu transformierenden Wortes oder vor oder hinter dieses Wort das Wort P zu setzen ist.

Beispiel 4. Gegeben sei eine Halbgruppe durch das Alphabet $\{a, b, c\}$ mit den zulässigen Substitutionen

$$b - acc, \qquad ca - accc, \qquad aa - \Lambda, \qquad bb - \Lambda, \qquad cccc - \Lambda$$

Es wird ein Algorithmus zur Lösung des Äquivalenzproblems in dieser Halbgruppe gesucht.
Wir konstruieren zunächst einen Hilfsalgorithmus, mit dessen Hilfe wir zu jedem Wort ein äquivalentes spezielles Wort (das reduzierte Wort) ermitteln. Zu diesem Zweck betrachten wir das geordnete System der orientierten Substitutionen

$$b \to acc, \qquad ca \to accc, \qquad aa \to \Lambda, \qquad cccc \to \Lambda$$

und verabreden folgende Verfahrensweise bei der Anwendung des Algorithmus auf ein beliebiges Wort R.
Es wird die erste auf das Wort R anwendbare (orientierte) Substitution dieser Liste genommen und ausgeführt. Sollte eine substituierbare Buchstabengruppe mehrmals in R eingebettet sein, so wird die erste, am weitesten links stehende, ersetzt.

Auf das so erhaltene Wort R' wird wieder die in der Liste als erste auftretende Substitution, und zwar ebenfalls auf die am weitesten links stehende Buchstabengruppe angewandt. Hat man nach endlich vielen Schritten ein Wort S erhalten, auf das keine der Substitutionen mehr

anwendbar ist, so sagt man, dass der Algorithmus auf das Wort R angewandt wurde und es in das Wort S verwandelt hat.[11]

Wir zeigen, dass wir jedes Wort R durch Anwendung des Reduktionsalgorithmus in eines der folgenden acht Wörter (reduzierte Wörter) umwandeln können:

$$\lambda, c, cc, ccc, a, ac, acc, accc$$

In der Tat, tritt in R der Buchstabe b auf, so lässt er sich durch die erste Substitution $b \to acc$ beseitigen. Das wird gemäß der Vorschrift solange fortgeführt, bis es kein b mehr gibt.
Dann führt man die zweite Substitution aus, indem man alle a mit einem links stehenden c durch $accc$ ersetzt und zwar solange, bis kein einziges a mehr vorhanden ist, welches unmittelbar rechter Nachbar von einem c ist, d.h. solange nicht alle a vor allen c stehen.

Schließlich beseitigt man alle Paare benachbarter a und alle Quadrupel benachbarter c, bis nicht mehr als ein a und nicht mehr als drei c in dem Wort enthalten sind. Wörter dieser Form gibt es insgesamt acht, die oben angegeben wurden.

Offenbar entspricht jedem Wort ein reduziertes, dem es äquivalent ist. Daher sind zwei Wörter dann und nur dann äquivalent, wenn ihnen dasselbe reduzierte Wort entspricht oder wenn die ihnen entsprechenden reduzierten Wörter äquivalent sind. Wir werden später beweisen, dass keine zwei der reduzierten Wörter äquivalent sind.
Hieraus folgt, dass zwei Wörter dann und nur dann äquivalent sind, wenn ihnen ein und dasselbe reduzierte Wort entspricht. Außerdem werden wir einen Algorithmus für das gestellte Wortproblem angeben. Er besteht darin, dass der Reduktionsalgorithmus auf jedes der beiden Wörter angewandt wird und die erhaltenen reduzierten Wörter miteinander verglichen werden.

Es seien beispielsweise die Wörter $cacb$ und bb gegeben. Wir bestimmen die reduzierten Wörter:

1. cac**b**, **ca**$cacc$, $accc$**ca**cc, acc**ca**$ccccc$, ac**ca**$cccccccc$, a**ca**$ccccccccccc$, **aa**$ccccccccccccc$, **cccc**$ccccccccccc$, **cccc**$cccccc$, **cccc**cc, cc;
2. **b**b, acc**b**, ac**ca**cc, a**ca**$ccccc$, **aa**$cccccccc$, **cccc**$cccc$, **cccc**, Λ.

Schlussfolgerung: Die Wörter $cacb$ und bb sind nichtäquivalent, weil wir die beiden verschiedenen reduzierten Wörter cc und Λ erhielten.

Beweis, dass keine zwei der reduzierten Wörter paarweise äquivalent sind. Zunächst stellen wir fest: Gibt es eine deduktive Kette, die von einem den Buchstaben b nicht enthaltenden Worte R zu einem den Buchstaben b ebenfalls nicht enthaltenden Worte S führt, dann kann man zwischen R und S eine deduktive Kette bilden, deren Zwischenwörter den Buchstaben b ebenfalls nicht enthalten.

Ersetzt man nämlich in allen Wörtern einer gegebenen deduktiven Kette jedes darin enthaltene b durch das Wort acc, so erhält man eine Folge von Wörtern, in der zwei hintereinanderstehende Wörter benachbart (im Sinne der Halbgruppe) oder einfach gleich sind.
Lässt man bei nebeneinander stehenden Wörtern die Wiederholungen weg, so erhält man die gewünschte deduktive Kette. In deduktiven Ketten dieses Typs tritt die Substitution $b - acc$

[11] Algorithmen zur Umwandlung der Wörter eines gewissen Alphabets, die in, dieser Weise mittels eines geordneten Systems orientierter Substitutionen arbeiten, werden als normal bezeichnet. Die Theorie der normalen Algorithmen, die von großem praktischen und theoretischen Interesse ist, wurde von dem korrespondierenden Mitglied der Akademie der Wissenschaften d. UdSSR A.A. Markow geschaffen und in seinem Buch "Theorie der Algorithmen" dargelegt.

gar nicht auf.

In jeder der übrigen zulässigen Substitutionen tritt der Buchstabe a auf der rechten und auf der linken Seite gleichzeitig in gerader oder in ungerader Anzahl auf.

Die analoge Behauptung ist auch für den Buchstaben c richtig. Damit haben wir zwei deduktive Invarianten für deduktive Ketten der oben angegebenen Form ermittelt.
Daraus ergibt sich nun sofort, dass keines der vier reduzierten Wörter, die den Buchstaben a einmal enthalten, mit einem der vier reduzierten Wörter äquivalent sein kann, in denen dieser nicht vorkommt. Genauso kann keines der reduzierten Wörter, in denen der Buchstabe c ein- oder dreimal vorkommt, irgendeinem Wort äquivalent sein, das zwei c oder kein c enthält. Deshalb braucht man sich nur noch von der Nichtäquivalenz folgender Paare zu überzeugen:

$$\Lambda, cc; c, ccc; a, acc; ac, accc$$

Wären die Wörter auch nur in einem der ersten drei Paare äquivalent, so müssten gemäß dem Satz dieses Paragraphen auch die Wörter des vierten Paares einander äquivalent sein. Es genügt daher, die Nichtäquivalenz der Wörter ac, $accc$ zu beweisen, und das wollen wir jetzt tun.

Wir verabreden noch einige Bezeichnungen. Unter dem Index eines im Wort R enthaltenen Buchstabens a verstehen wir die Anzahl der c, die links von diesem a stehen. Die Summe aller Indizes der in einem Wort R enthaltenen a nennen wir den Index des Wortes R.[12]
Jede der Substitutionen $aa-\Lambda$ und $cccc-\Lambda$ ändert den Index des Wortes stets um eine gerade Zahl. In der Tat, substituiert man aa an Stelle des leeren Wortes, so erhöht sich der Index um die Summe der Indizes der beiden a; da beide Indizes gleich sind, also um eine gerade Zahl. Dementsprechend verkleinert sich der Index des Wortes um eine gerade Zahl, wenn man aa durch das leere Wort ersetzt.

Setzt man $cccc$ für das leere Wort ein, so vergrößern sich die Indizes gewisser a um vier, während die Indizes der anderen a unverändert bleiben. Insgesamt vergrößert sich also der Index des Wortes um eine gerade Zahl. Analog wird der Index um eine gerade Zahl verkleinert, wenn man $cccc$ streicht.

Jetzt zeigen wir noch, dass die Substitution $ca-accc$ den Index des Wortes um eine ungerade Zahl ändert. Dazu vergleichen wir die Wörter

$$PcaQ \qquad \text{und} \qquad PacccQ$$

Der Index jedes im Teil P des Wortes R enthaltenen a bleibt ungeändert, während der Index der a in Q sich genau um zwei ändert. Der Index des einzigen zwischen P und Q auftretenden a wird um 1 geändert. Der Index des ganzen Wortes ändert sich also um eine ungerade Zahl.

Die Wörter ac und $accc$ haben beide den Index Null, also gerade Indizes. Wenn sie also äquivalent sein sollen, muss in der deduktiven Kette (von der wir nach dem obigen annehmen können, dass in ihr der Buchstabe b nicht vorkommt) eine gerade Anzahl von Substitutionen $ca-accc$ auftreten.

Diese Annahme führt jedoch auf einen Widerspruch, wie den folgenden Überlegungen zu entnehmen ist. Jede Anwendung der Substitution $ca-accc$ ändert die Anzahl der vorkommenden

[12] Im Worte $acbca$ beispielsweise hat der erste Buchstabe a den Index 0 (links von ihm steht kein c), während das zweite a den Index 2 hat. Also hat das Wort den Index 2.

Buchstaben c um 2. Daher muss eine gerade Anzahl von Substitutionen die Anzahl der vorkommenden c um ein Vielfaches von vier ändern.
Die Substitution $cccc - \Lambda$ vermindert offenbar die Anzahl der c um genau vier, während $aa - \Lambda$ die Anzahl der c nicht ändert. Fassen wir alles zusammen, so können wir schließen:
Wäre $ac \sim accc$, so würde sich die Anzahl der vorkommenden Buchstaben c um ein Vielfaches von vier unterscheiden; das ist aber offenbar nicht der Fall. Also können die Wörter ac und $accc$ nicht äquivalent sein. Damit ist der vorgeschlagene Algorithmus als zweckentsprechend nachgewiesen.

Die im Beispiel 4 vorgeführte Lösung des Wortproblems für eine konkrete Halbgruppe charakterisiert in vielem die Begriffe und Methoden, die auch bei der Untersuchung des Wortproblems bei anderen Halbgruppen auftreten. Wir wollen uns jetzt noch über die inhaltliche Bedeutung des Wortproblems und seine Wichtigkeit für die moderne Algebra klar werden. Das lässt sich an einem konkreten Beispiel zeigen.

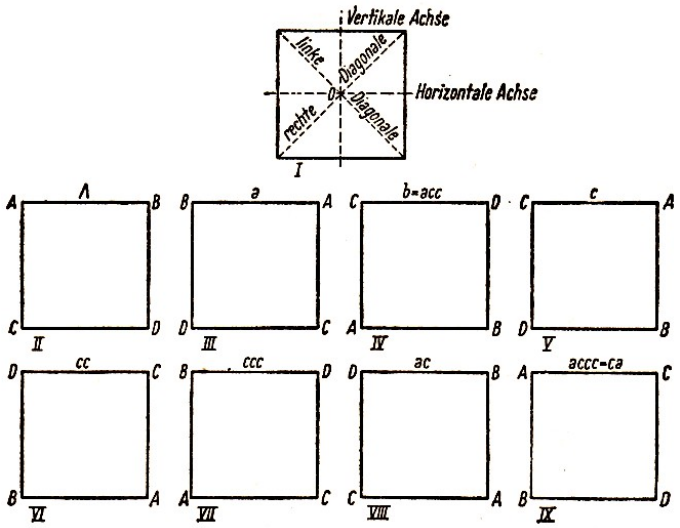


Abb. 3

Wir betrachten irgendein Quadrat (Abb. 3, I) und drei seiner Decktransformationen, d.h. Transformationen, die das Quadrat deckungsgleich in sich überführen:

l. Spiegelung an der vertikalen Achse durch den Mittelpunkt O des Quadrates;

2. Spiegelung an der horizontalen Achse durch den Mittelpunkt O des Quadrates;

3. Drehung um 90° im Uhrzeigersinn um den Mittelpunkt O.

Diese Transformationen wollen wir elementar nennen und mit den Buchstaben a, b, c bezeichnen. Abb. 3 (III, IV, V) zeigt, wie sich die Lage der Ecken des Quadrates II bei jeder der elementaren Transformationen ändert.

Wir weisen darauf hin, dass die mehrmalige Hintereinanderausführung von Decktransformationen wiederum eine Decktransformation des Quadrates ergibt. Wir wollen nun allgemeinverständlich definieren, was unter der Multiplikation zweier Transformationen (insbesondere zweier Decktransformationen eines Quadrates) zu verstehen ist.
Unter der Multiplikation zweier Transformationen verstehen wir ihre Hintereinanderausführung.

Wir wollen auch die für die Multiplikation übliche Symbolik und Bezeichnungsweise beibehalten. Mit dem Produkt zweier Transformationen ist also die resultierende Transformation

gemeint.
So ist beispielsweise das Produkt cc (Abb. 3, VI) das Ergebnis einer zweimaligen Drehung um $90°$, d.h. eine Drehung um $180°$.
Das Produkt ac (Abb. 3, VIII) ist das Resultat einer Spiegelung an der vertikalen Achse mit nachfolgender Drehung um $90°$, was einer Spiegelung an der linken Diagonalen gleichwertig ist (siehe Abb. 3,1). Das Produkt $(ac)(cc)$ der beiden vorigen Produkte entspricht einer Spiegelung an der rechten Diagonalen (siehe Abb. 3, I). Die hier eingeführte Multiplikation ist nicht kommutativ, d.h., die Faktoren eines Produktes dürfen im allgemeinen nicht miteinander vertauscht werden.

In Abb. 3, VIII und 3, IX sind die verschiedenen Lagen der Ecken des Quadrates dargestellt, die durch die Decktransformation ac bzw. ca des Ausgangsquadrates (II) erzeugt werden.
Von den Eigenschaften der Multiplikation von Zahlen bleibt aber die Assoziativität erhalten: Für beliebige Transformationen p, q, r gilt die Identität $(pq)r = p(qr)$. Also kann man die Stellung der Klammern in den Produkten beliebig ändern. So ergeben zum Beispiel $(ac)(cc)$ und $(((ac)c)c)$ die gleiche Decktransformation des Quadrates, und zwar eine Spiegelung an der linken Diagonalen.

Gegenstand unserer nachfolgenden Betrachtungen ist die Gesamtheit Ω der elementaren Transformationen a, b, c und aller jener Decktransformationen des Quadrates, die sich als Produkt endlich (aber beliebig) vieler elementarer Transformationen darstellen lassen.
Da die Multiplikation assoziativ ist, kann man bei der symbolischen Schreibweise der Elemente aus Ω die Klammern weglassen, also abb, $cabb$, $accc$ usw. schreiben.
Man muss nur auf die richtige Reihenfolge beim Ausführen der Transformationen achten. Somit kann man also jedes Produkt als Wort im Alphabet $\{a, b, c\}$ schreiben.

Fügt man zu einem Wort P von rechts ein Wort Q hinzu, so dass das Wort PQ entsteht, so ergibt sich aus der Assoziativität der Multiplikation, dass PQ das Produkt der Decktransformationen ist, welche den Wörtern P und Q entsprechen. Beispielsweise besteht das Wort $abccab$ aus dem Produkt der Decktransformationen, die den Wörtern abc und cab entsprechen.

Offenbar gibt es im Alphabet $\{a, b, c\}$ unendlich viele graphisch (d.h. dem Aussehen nach) verschiedene Wörter. Graphisch verschiedene Wörter können aber die gleiche Decktransformation aus Ω darstellen. Dann werden natürlich auch die Wörter als gleich angesehen; die Gleichheit wird in gewohnter Weise bezeichnet. Der Leser kann leicht die Richtigkeit folgender Gleichungen verifizieren:

$$b = acc, \quad (1) \qquad ca = accc \quad (2)$$

Zu diesem Zweck braucht er nur die durch die Transformationen auf den linken bzw. rechten Seiten dieser Gleichungen entstehenden Lagen der Ecken des Quadrats zu vergleichen.
Ferner sieht man ohne weiteres ein, dass jedes der Wörter aa, bb, $cccc$ dieselbe Decktransformation ergibt, und zwar die sogenannte identische Transformation, bei der alle Ecken in ihrer Ausgangsstellung bleiben.
Da diese Transformation keine Änderung hervorruft, wird man sie zweckmäßig mit dem leeren Symbol Λ bezeichnen. Somit gelten die Gleichungen

$$aa = \Lambda, \quad (3) \qquad bb = \Lambda, \quad (4) \qquad cccc = \Lambda. \quad (5)$$

Ein Vergleich von (1)-(5) mit den zulässigen Substitutionen der Halbgruppe des Beispiels 4 gestattet die folgende Aussage, die den engen Zusammenhang zwischen dieser Halbgruppe

und dem betrachteten System von Decktransformationen des Quadrates zeigt.

Zwei Produkte elementarer Decktransformationen eines Quadrates ergeben dann und nur dann die gleiche Transformation, wenn die ihnen entsprechenden Wörter in der Halbgruppe des Beispiels 4 äquivalent sind.

Aus den Gleichungen (1)-(5) folgt nämlich, dass bei jeder Anwendung einer zulässigen Substitution auf ein beliebiges Wort S dieses in ein gleiches Wort übergeführt wird. Wenn wir beispielsweise die Substitution $ca - accc$ auf das Wort $bcac$ anwenden, erhalten wir das Wort $bacccc$; wegen der Assoziativität der Multiplikation können wir aber $bcac = b(ca)c$ und $bacccc = b(accc)c$ schreiben.
Die rechten Seiten dieser Gleichungen sind als Produkte jeweils gleicher Faktoren ebenfalls gleich; dann müssen aber auch die linken Seiten übereinstimmen. Also sind je zwei benachbarte Wörter gleich.

Jetzt ist auch ohne weiteres zu verstehen, dass die Äquivalenz zweier Wörter in der Halbgruppe ihre Gleichheit nach sich zieht (d.h. die Gleichheit der Decktransformationen, die ihnen entsprechen). In der Tat, wenn $S \sim T$ ist, so sind in der deduktiven Kette je zwei benachbarte Elemente gleich; folglich ist auch $S = T$.

Es gilt auch die umgekehrte Behauptung: Wenn Wörter gleich sind, dann sind sie auch äquivalent. Denn wenn zwei Wörter gleich sind, so sind es auch die entsprechenden reduzierten Wörter (das ergibt sich aus der ursprünglichen Behauptung).
Damit lässt sich nun unmittelbar verifizieren, dass alle acht reduzierten Wörter acht paarweise verschiedene Decktransformationen ergeben (siehe Abb. 3, II-IX, wo die Lagen der Ecken des Quadrates (Abb. 3, II) nach den Decktransformationen zu ersehen sind, die den acht reduzierten Wörtern entsprechen).
Wenn also zwei Wörter gleich sind, so entspricht ihnen das gleiche reduzierte Wort; sie sind dann nach dem schon früher Bewiesenen äquivalent.

Auf diese Weise erhält die formale Äquivalenz zweier Wörter der Halbgruppe einen konkreten geometrischen Sinn, und dem Erkennen der Äquivalenz zweier Wörter entspricht die Lösung eines konkreten geometrischen Problems. Gleichzeitig hat sich der von uns beschriebene Algorithmus als eine allgemeine Methode zur Lösung einer beliebigen geometrischen Aufgabe des gegebenen Typs herausgestellt.

Ähnlich verhält es sich auch mit anderen Halbgruppen, in denen man der formalen Äquivalenz eine konkrete geometrische, algebraische oder auch andere Deutung geben kann.
Ohne Übertreibung kann man sagen, dass es in jedem Gebiet der Mathematik Sätze gibt, die nach einigen Umformulierungen als Aussagen über die Äquivalenz von Wörtern in einer gewissen Halbgruppe formuliert werden können.
Dieser Problemkreis kann natürlich in diesem Bändchen aus Platzmangel nicht behandelt werden; einige Erläuterungen werden wir noch im Verlauf der weiteren Darlegungen einfügen (siehe § 6).

Wir wollen noch hervorheben, dass wir, ausgehend von dieser geometrischen Deutung, die wir dem Wortproblem in unserer Halbgruppe gaben, einen noch einfacheren Algorithmus konstruieren können. Man braucht für jedes von zwei vorgelegten Produkten nur die Folge der entsprechenden Decktransformationen auszuführen (an einer Zeichnung) und die erhaltenen Ergebnisse zu vergleichen.

Übung. Man löse das Wortproblem für eine Halbgruppe, die durch das Alphabet $\{a, b\}$ mit

den zulässigen Substitutionen

$$aaa = bb, \qquad bbbb = \Lambda$$

definiert wird.

4 Rechenmaschinen mit automatischer Steuerung

Die Ausarbeitung eines Algorithmus (insbesondere eines "guten", bequemen Algorithmus) für Aufgaben eines gewissen vorgegebenen Typs hängt mit tiefliegenden und komplizierten Überlegungen zusammen, die hohe Qualifikation und große Erfindungsgabe erfordern.
Ist jedoch ein solcher Algorithmus einmal geschaffen, dann kann der Lösungsprozess auch von solchen Menschen durchgeführt werden, die von der Aufgabe selbst nur wenig verstehen.

Es wird von diesen Menschen nur gefordert, dass sie die paar einfachen elementaren Operationen, aus denen sich der Lösungsprozess aufbaut, ausführen können - und dass sie sich gewissenhaft und widerspruchslos nach der vorgelegten Beschreibung (dem Algorithmus) richten.
Ein solcher Mensch könnte sozusagen rein mechanisch jede beliebige Aufgabe des betrachteten Typs mit Erfolg lösen. Der Ausdruck "mechanisch" wird hier nur benutzt, um die Determiniertheit des Prozesses hervorzuheben. Auf Grund der heutigen Entwicklung von Wissenschaft und Technik verliert dieser Ausdruck seine ursprüngliche Bedeutung.

An Stelle des hypothetischen Menschen nämlich, der eine Aufgabe löst, deren Sinn er nicht versteht (oder nicht verstehen will), kann man wirklich eine Maschine setzen, die denselben Prozess ausführt - eben eine moderne Rechenmaschine mit automatischer Steuerung (einen elektronischen Rechenautomaten).

Unsere nächste Aufgabe besteht nun darin, die Grundprinzipien der Konstruktion und der Arbeitsweise dieser Maschinen zu erläutern. Zu diesem Zweck wenden wir uns noch einmal vorbereitend der Betrachtung des algorithmischen Prozesses zu, der von einem Menschen (Rechner) ausgeführt werden kann.

Durch den Algorithmus angeleitet führt der rechnende Mensch einen Prozess aus, bei dem von Bedeutung sind: das Verfahren, ferner die Aufbewahrung (Speicherung), die Verarbeitung und die Ausgabe gewisser Daten (gewisser Informationen). Diese Daten schreibt (speichert) der Rechner gewöhnlich auf einem Bogen Papier mit Hilfe von Ziffern, Buchstaben und anderen Symbolen auf. Die Gesamtheit aller dieser Symbole nennt man das Alphabet.
So wird beispielsweise in der Algebra ein Alphabet benutzt, das außer den gewöhnlichen Buchstaben und Ziffern auch die Zeichen für die algebraischen Operationen, Klammern usw. enthält.

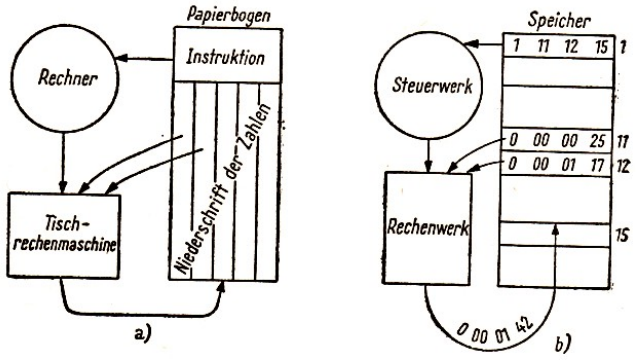


Abb. 4

Für den Rechenprozess, der von einem Menschen (dem Rechner) durchgeführt werden soll, ist das Auftreten folgender drei Faktoren charakteristisch (siehe das Schema in Abb. 4a).

1. Die Aufbewahrung der Informationen wird gewöhnlich durch das Aufschreiben aller Daten auf einen Bogen Papier gewährleistet, wobei zu den Daten auch die Anweisung (das Schema des Algorithmus) zur Lösung der Aufgaben gehört. Wir wollen nicht unerwähnt lassen, dass der Rechner faktisch nicht alles buchstäblich zu Papier bringt; einiges merkt er sich (bewahrt es nicht auf dem Bogen Papier, sondern im Gedächtnis auf), und gewisse Daten entnimmt er verschiedenen Nachschlagewerken und Tabellen.

Dadurch sollte jedoch nicht die Grundtatsache verschleiert werden, dass der Rechenprozess das Vorhandensein solcher Mittel voraussetzt, durch die die Aufbewahrung aller notwendigen Daten sichergestellt wird. In unserem Schema hat man also unter dem Bogen Papier die Gesamtheit aller Mittel zu verstehen, durch die die Aufbewahrung aller Daten gewährleistet wird.

2. Die Bearbeitung der Informationen setzt die Fähigkeit des Rechners voraus, die im Algorithmus vorgesehenen einzelnen elementaren Operationen auszuführen. Eventuell müssen dem Rechner zu diesem Zwecke spezielle Mechanismen zur Verfügung gestellt werden, z.B. können die gewöhnlichen arithmetischen Operationen auf einer Tischrechenmaschine durchgeführt werden. Jede einzelne Operation besteht im Prinzip in folgendem:

Der Rechner entnimmt gemäß den Anweisungen dem Bogen Papier gewisse Daten (z.B. Zahlen) und überträgt sie auf das Arbeitsgerät (Tischrechenmaschine). Das Ergebnis wird dann wieder an einem wohlbestimmten Platz des Bogens notiert.

3. Die Steuerung des Prozesses, d.h. die Vorbereitung und die Durchführung der einzelnen Operationen in jeder Etappe des Prozesses wird vom Rechner gemäß den Anweisungen besorgt.

Aus welchen Teilen setzt sich nun eine Maschine zusammen und wie greifen diese ineinander? Die Antwort auf diese Frage kann man vorwegnehmen. Die Maschine muss den eben beschriebenen Prozess ebenfalls ausführen können, aber ohne Eingreifen eines Rechners.

Die Maschine muss also erstens zur Niederschrift der Informationen ein bestimmtes Alphabet haben. Statt durch die gewöhnlichen graphischen Darstellungen, die sich durch ihre Gestalt voneinander unterscheiden, werden in der Maschine die verschiedenen Symbole des Alphabets durch voneinander verschiedene physikalische Zustände nachgebildet, z.B. durch verschiedene elektrische Potentiale oder verschiedene Magnetisierungszustände.

Vieles lässt sich schon durch die bevorzugte Anwendung eines Alphabetes mit zwei Symbolen (duales Alphabet) erreichen; man schreibt gewöhnlich 0 und 1.
Dieses Alphabet lässt sich ganz einfach physikalisch realisieren in Gestalt zweier Zustände: hohes Potential (oder ein Strom fließt) und niedriges Potential (oder kein Strom fließt).
Ferner kommt noch hinzu, dass die einfachsten logischen Operationen mit Veränderlichen auszuführen sind, die nur zwei Werte annehmen können: "wahr" und "falsch".

Die Wahl eines bestimmten Alphabets und die Methoden zur Darstellung der benötigten Daten sind jedoch für das Verständnis des Aufbaus und der Arbeitsweise der Maschine nicht entscheidend. Deshalb wollen wir uns auf den Hinweis beschränken, dass in den modernen Maschinen vorzugsweise das duale Alphabet und das Dualsystem (anstelle des üblichen Dezimalsystems) verwendet werden. Auf diese Einzelheiten werden wir aber im folgenden nicht näher eingehen.

Die in die Maschine eingegebenen Informationen sowie diejenigen, die im Verlaufe des Pro-

zesses erarbeitet werden, stellt man in Form gewisser physikalischer Parameter dar. In den uns interessierenden Fällen werden alle Daten durch Zahlen wiedergegeben. Insbesondere wird auch der Algorithmus, der die Maschine bei ihrer Arbeit "leitet", in Gestalt eines Satzes von Zahlen chiffriert. Algorithmen, die speziell für Maschinen entwickelt worden sind, nennt man gewöhnlich Programme. Das Programm ist der wichtigste Teil der Information, mit der die Maschine operiert.

Ferner gibt es in Übereinstimmung mit dem Schema der Abb. 4a in der Maschine Organe, welche die Funktion der Speicherung, der Ausgabe, der Bearbeitung und der Steuerung des Prozesses übernehmen (siehe das Schema der Abb. 4b).

1. Der Speicher übernimmt die Funktion des Bogens Papier. In der verabredeten Sprache fixiert die Maschine in ihm alle notwendigen Daten einschließlich des Programms.
Dass es möglich ist, ein Organ zu konstruieren, das diese Funktion erfüllt, dürfte keinem Zweifel unterliegen. Diese Tätigkeit kann z.B. ein Tonband übernehmen, auf dem die kodierten Daten festgehalten und von dem sie wieder abgenommen werden können wie beim normalen Tonbandgerät.
Der Speicher (das Gedächtnis der Maschine) besteht aus einer Menge von Zellen, die mit Hilfe der natürlichen Zahlen 1, 2, 3, .. nummeriert sind. Diese Zahlen nennt man die Adressen der Zellen. In jeder Zelle kann eine kodierte Nachricht gespeichert werden. In den Rechenautomaten wird jede solche Nachricht, wie schon erwähnt, in Form einer Zahl dargestellt.

In der Praxis werden in den elektronischen Automaten neben dem Tonband auch andere Speichermittel verwendet, z.B. Elektronenstrahlröhren; deren Arbeitsweise etwas an die Fernsehröhren erinnert, oder Magnettrommelspeicher; alle diese Organe werden zweckentsprechend zum Speichern eingesetzt. Wir werden jedoch die einzelnen Speicherarten nicht weiter unterscheiden. Ohne Nachteil für das Verständnis der Sache werden wir uns einen bestimmten Speicher vorstellen, etwa ein Tonband.

2. Das Rechenwerk übernimmt die Aufgabe der gewöhnlichen Tischrechenmaschine, obwohl die seiner Konstruktion zugrunde liegenden physikalischen Prinzipien völlig anders geartet sind. Die Verarbeitung der Eingabewerte zum gewünschten Ergebnis (z.B. die Addition von Zahlen) geschieht dadurch, dass die elektrischen Eingangssignale (die den Eingabewerten entsprechen) durch elektronische Geräte in elektrische Ausgangssignale (die den Ausgangsdaten entsprechen) verwandelt werden.

Die Eingangssignale kommen aus den Speicherzellen in das Rechenwerk, die Ausgangssignale verlassen das Rechenwerk und werden wieder in einer Speicherzelle untergebracht. Schematisch ist das in Abb. 4b dargestellt, wo die Zahlen aus den Zellen 11 und 12 addiert werden und das Ergebnis in die Zelle 15 gebracht wird.

Damit nun diese Operationen in der Maschine in einem gewissen Rhythmus ausgeführt werden, müssen zu Beginn des Taktes die Zellen 11 und 12 sowie die Zelle 15 mit dem Rechenwerk verbunden werden. Ferner muss auch das Rechenwerk auf die richtige Operation (im vorliegenden Fall die Addition) eingestellt sein. Alles dies fällt aber schon in die "Kompetenzen" des Steuerwerks.

3. Das Steuerwerk hat die Funktion auszuführen, die im Schema er Abb. 4a dem Rechner selbst zukommt. In jedem Arbeitsstadium muss das Steuerwerk die Bedingungen zur Realisierung der nachfolgenden Operation herstellen.
Dabei arbeitet es wie eine im Fernsprechbetrieb verwendete Selbstwählanlage, die diejenigen

"Teilnehmer" (Verteiler und Zellen der Maschine) miteinander verbindet, die an der jeweiligen Operation beteiligt sind. Bildlich gesprochen schlägt das Steuerwerk im Programm nach, was zu machen ist, und stellt dann die entsprechenden Verbindungen in der Maschine her, damit die nächste Operation ungestört ablaufen kann.

Zur genaueren Beschreibung der hier auftretenden Konstellationen weisen wir darauf hin, dass jede Maschine durch das in ihr anwendbare System von Befehlen (Kommandos) eindeutig charakterisiert wird. Jedes Programm, welches die Maschine verarbeiten kann, ist nur eine bestimmte Kombination dieser möglichen Befehle mit gewissen Hilfszahlen (Parametern), die sich in den Zellen des "Gedächtnisses" befinden.

Zum Beispiel wird in der BESM der Akademie der Wissenschaften der UdSSR das sogenannte Dreiadresssystem für die Befehle verwendet. Jeder Befehl ist eine Folge von vier Zahlen:

$$\alpha\beta\gamma\delta$$

von denen die erste die Nummer der vorzunehmenden Operation beschreibt, die nächsten beiden Zahlen die Adressen der beiden Zellen angeben, in denen die Zahlen gespeichert sind, mit denen die Operation ausgeführt werden soll, und die letzte die Adresse der Zelle, in der das Ergebnis gespeichert werden soll (insgesamt drei Adressen).

Jeder Befehl ist in einer Zelle untergebracht in Form einer Zahl, deren Ziffern in vier Gruppen zerlegt werden. Diesen Gruppen kommt dann die oben angegebene Bedeutung zu. Zum Beispiel stehen in der Zelle 1 des Speichers der Abb. 4b die Zahlen 1 11 12 15, die den folgenden chiffrierten Befehl symbolisieren:

"Es sind zu addieren (Operation Nr. 1) die Zahlen aus den Zellen 11 und 12, und das Ergebnis ist in der Zelle 15 zu speichern."

(Wir haben eine Zerlegung in Gruppen von rechts nach links zu je zwei Ziffern angenommen. Um keine Irrtümer aufkommen zu lassen, wollen wir auch fernerhin die Befehle in dieser Form beschreiben.)

Das übliche Befehlssystem umfasst einige Dutzend Befehle. Wir wollen hier nur die unbedingt erforderlichen angeben.

1. Arithmetische Befehle:

a) $1\beta\gamma\delta$ ... addiere die Zahl aus β zu der Zahl aus γ und speichere die Summe in δ;

b) $2\beta\gamma\delta$ ... subtrahiere von der in β stehenden Zahl die Zahl aus γ und speichere die Differenz in δ;

c) $3\beta\gamma\delta$ ... multipliziere die Zahl aus β mit der Zahl aus γ und speichere das Produkt in δ;

d) $4\beta\gamma\delta$ ... dividiere die Zahl aus β durch die Zahl aus γ und speichere den Quotienten in δ.

2. Sprungbefehle:

e) 5 00 00 δ ... gehe über zu dem in $delta$ stehenden Befehl (unbedingter Sprung);

f) 5 01 $\gamma\delta$ ... gehe über zu dem in δ stehenden Befehl unter der Bedingung, dass in der Zelle γ eine positive Zahl gespeichert ist;

g) 5 02 $\gamma\delta$ ... gehe über zu dem in δ stehenden Befehl unter der Bedingung, dass in der Zelle γ eine negative Zahl gespeichert ist,

3. Stopbefehl: 0 00 00 00

Außer den aufgezählten Befehlen gibt es noch die Befehle für die sogenannten logischen Operationen und auch andere, mit denen wir uns hier jedoch nicht weiter aufhalten wollen. Die aufgezählten Befehle genügen schon Völlig zur Aufstellung der verschiedenartigsten Programme. Einige Beispiele werden im folgenden Paragraphen behandelt.

Die Befehle f) und g) nennt man bedingte Sprungbefehle. Sie werden nur dann ausgeführt, wenn die betreffende Bedingung erfüllt ist, andernfalls haben diese Befehle keinerlei Auswirkungen.
Gewöhnlich werden die Befehle von der Maschine in der Reihenfolge ausgeführt, in der sie gespeichert sind. Von dieser Reihenfolge wird nur abgegangen, wenn ein entsprechender (unbedingter oder bedingter) Sprungbefehl erteilt wird.

Die Arbeit der Maschine erfolgt in sogenannten Takten, wobei in jedem Takt ein einziger Befehl ausgeführt wird. Zu Beginn jedes Taktes kommt aus einer Speicherzelle die in ihr enthaltene Zahl (die also einen Befehl darstellt) in das Steuerwerk. Wenn der Befehl dort ist, wird die entsprechende Verbindung hergestellt und dadurch die Ausführung der nächsten Operation des Prozesses sichergestellt.
Danach läuft in das Steuerwerk der folgende Befehl ein, und die Maschine führt die entsprechende Operation aus usw., bis der Befehl zum Stoppen der Maschine kommt.

Die technische Realisierung eines solchen Steuerwerks bringt keinerlei prinzipielle Schwierigkeiten mit sich. Es wird von diesem Werk nicht mehr gefordert, als was auch jede Selbstwählanlage zu leisten hat, in der die Nummer mittels eines elektrischen Signals gewählt wird.

Das sind die drei Grundbausteine eines Rechenautomaten. Zwar gibt es noch eine ganze Reihe wichtiger Organe, insbesondere für die Eingabe in die Maschine, für die Ausgabe der in ihr erarbeiteten Resultate usw.; diese Teile haben jedoch auf die Arbeitsprinzipien und die logischen und mathematischen Möglichkeiten einer Maschine keinen Einfluss.
Daher brauchen wir bei den folgenden Betrachtungen nicht darauf einzugehen. Wir wollen stets annehmen, dass die einzugebenden und die auszugebenden Informationen sich im Speicher befinden und auch dort verbleiben.

5 Das Programm (Maschinenalgorithmus)

In diesem Paragraphen wollen wir einige Beispiele von Programmen betrachten, die für elektronische Rechenautomaten mit Dreiadresssystemen aufgestellt worden sind.
Diese Programme werden aus den früher untersuchten Algorithmen abgeleitet, wobei berücksichtigt wird, dass nicht mehr ein Mensch, sondern ein Rechenautomat, der die im vorigen Paragraphen beschriebenen Befehle ausführen kann, nach diesem Algorithmus arbeiten soll.

Die Analyse der behandelten Beispiele klärt den realen Sinn der üblichen Redeweise "Die Maschine wird von dem ihr eingegebenen Programm gesteuert".
Es wird dargelegt, auf welche Weise die Reihenfolge der ins Steuerwerk einlaufenden Befehle reguliert wird und wie man mit einer relativ kleinen Anzahl von Befehlen recht umfangreiche Rechnungen ausführen lassen kann, wenn beispielsweise sehr viele Operationen notwendig sind, um irgendeine Variante des Aufgabentyps zu lösen.

Im folgenden wollen wir annehmen, die arithmetischen Operationen Addition, Subtraktion, Multiplikation und Division seien in den Befehlen durch die Nummern 1, 2, 3, 4 bezeichnet

(siehe auch § 4).

Beispiel 1. Wir wollen die Auflösung des Gleichungssystems

$$ax + by = c$$
$$dx + ey = f$$

programmieren. Um etwas Bestimmtes vor Augen zu haben, nehmen wir an, die Koeffizienten a, b, c, d, e, f seien der Reihe nach in den Speicherzellen von Nummer 51 an untergebracht.

Adresse	Inhalt
51	a
52	b
53	c
54	d
55	e
56	f

Ferner werden wir für die Zwischen- und Endresultate der Rechnung die Zellen 31-50 benutzen. Wie aus den Formeln

$$x = \frac{ce - fb}{ae - bd} \qquad , \qquad y = \frac{af - cd}{ae - bc}$$

zu ersehen ist, muss man, um das Ergebnis zu erhalten, sechs Multiplikationen, drei Subtraktionen und zwei Divisionen ausführen. (Der Nenner ist für beide Brüche derselbe! Er wird natürlich als von Null verschieden vorausgesetzt (Red.)).
Dementsprechend besteht das Programm aus 12 Befehlen, die in den Zellen. 1-12 stehen (Tabelle A):

Tabelle A

Adresse	Inhalt (Befehl)
1	3 53 55 31
2	3 56 52 32
3	3 51 56 33
4	3 53 54 34
5	3 51 55 35
6	3 52 54 36
7	2 31 32 37
8	2 33 34 38
9	2 35 36 39
10	4 37 39 40
11	4 38 39 41
12	0 00 00 00

Tabelle B

Adresse	Inhalt
31	ce
32	fb
33	af
34	cd
35	ae
36	bd
37	$ce - fb$
38	$af - cd$
39	$ae - bd$
40	$\frac{ce-fb}{ae-bd}$
41	$\frac{af-cd}{ae-bd}$

Die Befehle laufen in der Reihenfolge wachsender Adressen in das Steuerwerk und werden in dieser Reihenfolge ausgeführt. Nachdem der letzte Befehl ausgeführt ist, sind in den Zellen 31-41 die folgenden Zahlen gespeichert: siehe Tabelle B.

Das sind die Zwischenergebnisse und das Endresultat (in den Zellen 40 und 41) der Rechnung

Beispiel 2. Es seien die Lösungen von n vorgegebenen Gleichungssystemen gesucht:

$$a_i x + b_i y = c_i$$
$$d_i x + e_i y = f_i \qquad (i = 1, 2, ..., n)$$

Der lösende Algorithmus ist eine n-fache Wiederholung des Algorithmus zur Auflösung eines Systems dieser Art. Man könnte leicht ein entsprechendes Programm für die Maschine aus dem oben angegebenen Programm zusammenstellen.
Die $6n$ Koeffizienten würden in $6n$ Zellen untergebracht werden, und das Programm bestünde aus $11n + 1$ Befehlen. Der erste Zyklus von 11 Befehlen würde die Lösung des ersten Systems ermitteln, der zweite Zyklus von 11 Befehlen die Lösung des zweiten Systems usw., insgesamt n-mal. Der $(11n + 1)$-te Befehl ist dann der Stopbefehl.

Eine solche beträchtliche Vergrößerung des Programmaufwandes ist jedoch denkbar unzweckmäßig. Man kann sie umgehen.
Wir brauchen nur daran zu denken, dass jeder folgende Zyklus aus 11 Befehlen aus dem vorhergehenden erhalten werden kann, wenn man die in den Befehlen enthaltenen Adressen ändert. Wenn nämlich die $6n$ Koeffizienten in aufeinanderfolgenden Zellen beispielsweise mit der Nummer 51 beginnend angeordnet sind, so brauchen in den ersten sechs Befehlen nur die Adressen der Faktoren um jeweils sechs vergrößert werden. Auf diese Weise erhält man die Befehle für den folgenden Zyklus.

Damit nun die Lösungen der einzelnen Systeme, die jeweils zwei Zellen beanspruchen, ebenfalls aufeinanderfolgen, hat man jede der letzten Adressen im zehnten und elften Befehl um zwei zu erhöhen. Eine solche Adressenänderung kann mittels sogenannter Umadressierungs- oder Adressenänderungsbefehle ausgeführt werden.
In unserem Fall benötigen wir acht dieser Befehle. Zu diesem Zweck setzen wir in die Zellen 25 und 26 die Parameter (Konstanten):

Adresse	Inhalt
25	0 06 06 00
26	0 00 00 02

In die Zellen 12-19 kommen die acht Umadressierungsbefehle:

Adresse	Inhalt
12	1 01 25 01
13	1 02 25 02
14	1 03 25 03
15	1 04 25 04
16	1 05 25 05
17	1 06 25 06
18	1 10 26 10
19	1 11 26 11

Nachdem die Befehle aus den Zellen 1-19 ausgeführt sind, steht in den Zellen 40 und 41 genau wie früher die Lösung des ersten Gleichungssystems, während in den Zellen 1-6 und 10-11 jetzt schon die abgeänderten Befehle stehen:

Adresse	Inhalt
1	3 59 61 31
2	3 62 58 32
3	3 57 62 33
4	3 59 60 34
5	3 57 61 35
6	3 58 60 36
10	4 37 39 42
11	4 38 39 43

Werden jetzt die Befehle aus den Zellen 1-19 noch einmal ausgeführt, so liefert dieser zweite Zyklus schon die Lösung des zweiten Gleichungssystems, welche in den Zellen 42 und 43 gespeichert worden ist. Außerdem sind auch bereits wieder die Befehle 1-6 und 10 und 11 umadressiert worden, womit die Bedingungen für einen dritten Arbeitszyklus geschaffen werden sind usw.

Wie kann man aber nun erreichen, dass die Maschine diesen Zyklus von 19 Befehlen so oft ausführt, wie Gleichungssysteme vorgegeben sind, und dass dann, wenn die Lösungen aller Gleichungssysteme gefunden sind, die Maschine gestopt wird?
Zu diesem Zweck werden die Zellen 27 und 28 mit den Parametern 0 00 00 01 und 0 00 00 0n besetzt, wobei n gleich der Anzahl der zu lösenden Gleichungssysteme ist. Außerdem treten zu den vorhandenen 19 Befehlen noch 3 hinzu:

Adresse	Inhalt
20	2 28 27 28
21	5 01 28 01
22	0 00 00 00

Der Befehl 20 verkleinert den Inhalt der Zelle 28 nach jedem ausgeführten Befehlszyklus (1-19) um Eins. Der Befehl 21 ist ein bedingter Sprung zum Befehl 1: Er wird solange ausgeführt, solange in der Zelle 28 noch eine positive Zahl steht.
Mit diesem Befehl wird also der Übergang zum folgenden Befehlszyklus (1 bis 19), um das folgende Gleichungssystem zu lösen, sichergestellt.

Steht aber in der Zelle 28 die Zahl Null, und das ist nach genau n Befehlszyklen der Fall, so wird der bedingte Sprung nicht ausgeführt; statt dessen bringt der folgende Befehl 22 die Maschine zum Stillstand.

Es dürfte jetzt völlig klar sein, dass das Programm aus den oben angegebenen Befehlen 1-22 und den Parametern in den Zellen 25-28 zur Lösung der gestellten Aufgabe brauchbar ist.
Die Struktur dieses Programms lässt sich übersichtlich in einem Schema darstellen.

Befehle

1 2 3 4 ⋮ 11	Arithmetische Operationen
12 13 ⋮ 19	Adressenänderungen
20	Zählung der Zyklen
21	Bedingter Sprung
22	Stop

Parameter

25	0 06 06 00
26	0 00 00 02
27	0 00 00 01
28	0 00 00 0n

Beispiel 3. Wir stellen jetzt ein Programm zum Aufsuchen des größten gemeinsamen Teilers zweier Zahlen a und b auf.
In den Zellen 12 und 13 sollen die Ausgangswerte a und b und in den Zellen 14 und 15 die Zwischenwerte untergebracht werden, wobei das endgültige Ergebnis in der Zelle 15 verbleiben soll.

Das angegebene Programm wurde gemäß dem im Paragraphen 1 beschriebenen Euklidischen Algorithmus aufgestellt.

Programm zum Beispiel 3

Adresse	Inhalt	Erklärung
01	1 12 05 15	Transportiere die Zahl aus Zelle 12 in Zelle 15
02	2 12 13 14	Speichere die Differenz der Zahlen aus Zelle 12 und Zelle 13 in Zelle 14
03	5 02 14 06	Springe nach Zelle 06, wenn in Zelle 14 eine negative Zahl steht
04	5 01 14 09	Springe nach Zelle 09, wenn in Zelle 14 eine positive Zahl steht
05	0 00 00 00	Stop
06	1 13 05 12	Transportiere die Zahl aus Zelle 13 in Zelle 12
07	1 15 05 13	Transportiere die Zahl aus Zelle 15 in Zelle 13
08	5 00 00 01	Springe unbedingt nach Zelle 01
09	1 13 05 12	Transportiere die Zahl aus Zelle 13 in Zelle 12
10	1 14 05 13	Transportiere die Zahl aus Zelle 14 in Zelle 13
11	5 00 00 01	Springe unbedingt nach Zelle 01

Nach den ersten beiden Takten stehen in der Zellen 12-15 folgende Zahlen

Adresse	Inhalt
12	a
13	b
14	$a-b$
15	a

Wenn $a-b=0$ (d.h. $a=b$) ist, sind die bedingten Sprungbefehle ohne Auswirkungen, und der Befehl 05 kommt zur Ausführung, der die Maschine anhält. In diesem Moment befindet sich in der Zelle 15 tatsächlich das endgültige Ergebnis (vergleiche mit der Anweisung 3 des § 1).

Wenn $a-b<0$ (d.h. $a<b$) ist, folgt auf den Befehl 03 der Befehl 06, der zusammen mit dem Befehl 07 die Plätze der Zahlen a und b in den Zellen 12 und 13 vertauscht (vergleiche mit Anweisung 4). Der nachfolgende Befehl 08 verlangt einen unbedingten Sprung nach 01, und dort beginnt der zweite Arbeitszyklus der Maschine.

Wenn aber $a-b>0$ (d.h. $a>b$) ist, wird der bedingte Sprung 03 nicht ausgeführt, und der Befehl 04 zieht den Befehl 09 nach sich, der zusammen mit dem folgenden Befehl 10 in die Zellen 12 und 13 den früheren Subtrahenden und die Differenz, also b und $a-b$ transportiert (vergleiche mit Anweisung 4). Sodann verlangt der Befehl 11 den unbedingten Sprung zum Befehl 01. Dort beginnt dann der zweite Arbeitszyklus der Maschine.

Die aufeinanderfolgenden Arbeitszyklen der Maschine erzeugen in den Zellen 12 und 13 eine Folge von Zahlenpaaren

$$(a_1,b_1),(a_2,b_2),...,(a_i,b_i),(a_{i+1},b_{i+1}),...$$

und in der Zelle 15 eine Zahlenfolge

$$a_1,a_2,...,a_i,a_{i+1},...$$

bis das erste Paar gleicher Zahlen (a_k,b_k) auftritt. Dann bringt der Befehl 05 die Maschine zum Stehen, und in der Zelle 15 ist das gesuchte Resultat gespeichert.

In den analysierten Beispielen treten mit hinreichender Klarheit folgende zwei grundlegenden

Arbeitsprinzipien der elektronischen Rechenautomaten zutage:

1. Normalerweise werden die Befehle des Programms durch die Maschine in der Reihenfolge ausgeführt, in der sie in den Speicherzellen stehen. Die Maschine kann allerdings auch den Gang des Rechenprozesses in Abhängigkeit von den laufend erhaltenen Rechenergebnissen automatisch ändern. Das wird gerade durch die bedingten Sprungbefehle ermöglicht.

2. Bei einem verhältnismäßig umfangreichen Programm besteht unter Umständen die Möglichkeit, es dadurch zu verkürzen, dass die Maschine einzelne Teile des Programms oder sogar das ganze mehrfach wiederholt, indem sie selbst Adressen ändert. Das durch Ziffern verschlüsselte Programm wird nämlich in demselben Speicher wie die gewöhnlichen Zahlen aufbewahrt. Daher ist es der Maschine möglich, auch mit den bedingten Zahlen, d.h. mit den verschlüsselten Befehlen zu rechnen.
Auf diese Weise kann die Maschine beispielsweise Adressenänderungen in den Befehlen vornehmen.

Die Maschine ist auf Grund der für sie charakteristischen Arbeitsprinzipien in der Lage, auch solche Aufgaben zu lösen, die keine ausgesprochenen Rechenaufgaben sind.
So kann man z.B. den Algorithmus des Theseus (das Suchen in einem Labyrinth) oder gewisse Algorithmen für Wortprobleme in gewissen Halbgruppen programmieren und die entsprechenden Prozesse in der Maschine realisieren. Dazu ist es selbstverständlich notwendig, dass die Maschine außer den arithmetischen noch einige andere Operationen ausführen kann.
Es müssen auch noch einige Steuerbefehle hinzukommen, die für die oben betrachteten arithmetischen Aufgaben nicht erforderlich waren. In den wirklich arbeitenden elektronischen Rechenautomaten können diese einfachen Operationen ausgeführt werden (d.h., es sind die entsprechenden Befehle vorgesehen).

Es genügt daher, das Programm zu ändern, damit die Maschine auf die gewünschte Arbeitsweise umgeschaltet wird. Jedoch treten nicht nur in der Mathematik, sondern auch in den verschiedensten anderen Gebieten des menschlichen Lebens Prozesse auf, die nach streng festgelegten formalen Vorschriften (d.h. Algorithmen) ablaufen, die man ebenfalls programmieren kann.

So setzen sich beispielsweise in Buchhaltung und Planung die Analyse und die Bearbeitung der vorkommenden Daten und die Zusammenstellung der Bilanzmaterialien, um optimale Ergebnisse zu erzielen, aus einer langen Kette von elementaren Operationen zusammen. Es treten aber nur wenige Typen von Operationen auf, die in strenger Übereinstimmung mit speziellen Anweisungen und Schemata verwirklicht werden können. In anderen Fällen fehlt bisher noch ein entsprechend klarer und vollkommener Algorithmus.

Es ist jedoch möglich, einen zu entwickeln und bis zur formalen Vollkommenheit zu treiben. Dies bezieht sich insbesondere auf das Problem der Übersetzung aus einer Sprache in eine andere. Bei hinreichend tiefgehender Analyse des formalen Aufbaus und entsprechender Klassifizierung der grammatischen und stilistischen Grundregeln und der Verfahren, wie man Wörterbücher verwendet, kann man einen völlig befriedigenden Übersetzungsalgorithmus entwickeln, mit dessen Hilfe etwa wissenschaftliche und geschäftliche Texte schnell und fehlerfrei übertragen werden können (für einige Sprachen wurden solche Algorithmen schon entwickelt).

In diesem Zusammenhang ist noch eine Bemerkung interessant.
In vielen Spielen gibt es für die Regeln, nach denen die Figuren bewegt werden, ganz charakteristische, genau definierte formale Vorschriften. Dieser Umstand wirft sofort die Frage auf, ob

man Algorithmen aufstellen kann, die eine erfolgreiche Spielführung gewährleisten. Ein solcher Algorithmus muss auf einer gewissen Spieltaktik basieren und dem Spieler in jeder Stellung einen einzigen Zug (den besten im Sinne der Taktik) vorschreiben, den er machen muss.

So kann man beispielsweise im Schach für die Figuren ein gewisses Bewertungssystem einführen. Hierbei muss der König sehr hoch bewertet werden, die Dame etwas weniger, der Turm noch weniger usw., einem Bauern kommt der geringste Wert zu.
Außerdem müssen aber auch die Stellungen in bestimmter Weise bewertet werden (Stellung der Figuren auf dem Brett, ihre Beweglichkeit usw.). Die Differenz der Summe der Bewertungen für die weißen Figuren und der Summe der Bewertungen für die schwarzen Figuren charakterisiert dann (im Sinne der vorgegebenen Taktik) die materiellen und positionellen Vorteile der weißen über die schwarzen Figuren im vorliegenden Spielstadium.

Der einfachste Algorithmus besteht in der Durchmusterung aller Züge, die im Moment möglich sind, und der Auswahl desjenigen Zuges, der vom Standpunkt des vorgegebenen Bewertungssystems eine größere Überlegenheit sichert.
Besser, aber auch wesentlich komplizierter, ist ein Algorithmus, der alle möglichen Kombinationen der nächsten drei oder sogar fünf Züge berücksichtigt und nach dem vorgegebenen Auswahlprinzip den optimalen Zug ermittelt.[13]

Hieraus geht klar hervor, wie vielfältig geistige Arbeit sein kann, die mit bestimmten Algorithmen ausgeführt wird und ausgeführt werden kann. In allen Fällen kann man diese Algorithmen prinzipiell auch programmieren und die entsprechenden Arbeiten auf programmgesteuerten Automaten ausführen lassen. Es existieren insbesondere schon heute Programme zur Übersetzung aus einer Sprache in eine andere und zum Schachspielen, nach denen von programmgesteuerten Automaten, wie z.B. der BESM der Akademie der Wissenschaften der UdSSR, erfolgreich gearbeitet werden kann.

6 Notwendigkeit der Präzisierung des Begriffs Algorithmus

Aus unseren bisherigen Ausführungen ist schon zu ersehen, wie eng die Algorithmen mit den programmgesteuerten Rechenautomaten zusammenhängen.

Offenbar lässt sich jeder Prozess, dessen einzelne Schritte nacheinander auf einer automatisch arbeitenden Maschine verwirklicht werden können, durch einen Algorithmus beschreiben.
Andererseits lassen sich alle bis jetzt bekannten Algorithmen sowie diejenigen, die man beim heutigen Stande der Wissenschaften erwarten kann, prinzipiell in programmgesteuerten Rechenautomaten realisieren.

Die letzte Behauptung erfordert noch einige Erläuterungen.
Wie schon erwähnt, kann der algorithmische Prozess zur Lösung einer Aufgabe bestimmten Typs beliebig lang sein und die Menge der Daten, die mit dem Algorithmus bearbeitet wird,

[13] Ja, noch mehr, theoretisch gibt es einen besten Algorithmus, der stets den Gewinn sichert, wenn er nur möglich ist. Für ein Schachproblem vom Typ "Matt in N Zügen" sieht dieser Algorithmus etwa folgendermaßen aus: Stelle alle möglichen Zugfolgen von N Zügen $W_1, S_2, W_3, S_4, ..., S_{n-1}, W_n$ auf (wobei $W_1, W_3, ...$ die Züge von Weiß und $S_2, S_4, ...$ die Züge von Schwarz sind) und wähle eine solche Zugfolge $W_1, W_3, ...$ aus, die unabhängig von den schwarzen Zügen $S_2, S_4, ...$ zum Gewinn führt. Dieser Algorithmus ist jedoch so kompliziert, dass er selbst auf den modernen schnelloperierenden Rechenautomaten praktisch nicht realisiert werden kann.

alle Vorstellungen übersteigen. Andererseits hat aber der Speicher eines modernen Automaten nur ein beschränktes Volumen (da erstens die Anzahl der Zellen und zweitens auch das Fassungsvermögen jeder Zelle beschränkt ist). Deshalb kann es vorkommen, dass man einen Algorithmus auf Grund der bestehenden technischen Voraussetzungen nicht realisieren kann.

Dies lässt sich schon am Beispiel des Euklidischen Algorithmus illustrieren. Die einfache Aufgabe des Bestimmens des größten gemeinsamen Teilers zweier Zahlen kann für einen Rechner undurchführbar sein, wenn er zur Lösung dieser Aufgabe mehr Papier und Tinte benötigt, als er sich jemals beschaffen könnte. Genauso, wie es dem Rechner mit dem Euklidischen Algorithmus gehen kann, kann für die Maschine eine gegebene konkrete Aufgabe undurchführbar sein, wenn zu ihrer Lösung mehr Speicherraum erforderlich ist, als der Maschine zur Verfügung steht.

In ähnlich gelagerten Fällen muss man also den algorithmischen Prozess, wie schon betont, als potentiell durchführbaren Prozess ansehen, der nach endlich vielen Schritten (obwohl die Anzahl dieser Schritte sehr groß sein kann) zum gesuchten Ergebnis führt.
Wenn von der Möglichkeit, einen Algorithmus in einer Maschine zu verwirklichen, die Rede ist, denkt man stets an die potentielle Möglichkeit, das Speichervolumen unbegrenzt zu vergrößern.

Auf Grund des erwähnten Zusammenhanges zwischen dem Begriff "Algorithmus" und dem Begriff "programmgesteuerter Rechenautomat mit potentiell unbeschränktem Speichervolumen" lassen sich ihre wesentlichen Züge besser hervorheben.
Jede Präzisierung des einen Begriffes ist gleichzeitig eine Präzisierung des anderen.

Trotz aller Betonung der Allgemeinheit dieser Begriffe ist keiner bisher von uns exakt definiert worden. Eine exakte mathematische Definition des Begriffs des Algorithmus (und gleichzeitig damit eine genaue Definition des artverwandten Begriffs des programmgesteuerten Rechenautomaten) wurde schon in den dreißiger Jahren unseres Jahrhunderts von den Wissenschaftlern erarbeitet.
Warum haben aber die Mathematiker vieler Jahrhunderte, ohne besonders beunruhigt zu sein, stets den verschwommenen Begriff des Algorithmus übernommen?
Warum entstand in dieser relativ kurzen Zeit ein so dringendes Bedürfnis nach einer streng mathematischen Definition dieses Begriffs, damit er als Gegenstand mathematischer Untersuchungen dienen könne?

Bis vor kurzem trat der Begriff des Algorithmus in der Mathematik nur im Zusammenhang mit der Konstruktion (Entwicklung) konkreter Algorithmen auf; denn die Aussage, dass für Aufgaben eines bestimmten Typs ein Algorithmus existiere, war praktisch stets mit einer ausführlichen Beschreibung dieses Algorithmus verbunden. Unter diesen
Umständen brauchte jemand, dem ein System formaler Regeln mitgeteilt wurde, nur Daten einzusetzen und sich bei der Anwendung der Regeln davon zu überzeugen, dass er automatisch auf das Ergebnis geführt wurde.

Daher tauchte das Problem einer streng mathematischen Definition des Begriffes "Algorithmus" gar nicht auf, und man konnte sich mit dem verschwommenen, von jedem Mathematiker mit der richtigen Vorstellung verknüpften Begriff des Algorithmus begnügen.
Im Zuge der Entwicklung häuften sich in der Mathematik jedoch Tatsachen an, die diese Situation schlagartig änderten. Die Ursachen dafür lagen in dem natürlichen Streben der Mathematiker begründet, immer mächtigere Algorithmen zu schaffen, die nach Möglichkeit umfangreichere Aufgabenklassen (Aufgaben allgemeineren Typs) lösen können. Der Betrachtung

solcher Tatsachen wollen wir uns nun zuwenden.

Wir erinnern an den Algorithmus zur Berechnung von Quadratwurzeln, der in allen Schullehrbüchern beschrieben wird. Man kann ein allgemeineres Ziel stellen:
Es ist ein Algorithmus zu konstruieren, mit dem man eine Wurzel beliebigen Grades aus jeder vorgegebenen Zahl ziehen kann. Es ist natürlich zu erwarten, dass ein solcher Algorithmus etwas schwieriger zu konstruieren sein wird.

Die Perspektiven, die sich aus dem Besitz eines solchen Algorithmus ergeben, sind schon sehr verlockend. Man kann jedoch noch weiter gehen. Das Ausziehen einer Wurzel n-ten Grades aus einer Zahl a ist doch gleichbedeutend mit dem Auflösen der Gleichung $x^n - a = 0$ (dem Bestimmen einer Wurzel der Gleichung). Nun kann man aber eine noch allgemeinere Aufgabe formulieren.

Es ist ein Algorithmus zu konstruieren, mit dessen Hilfe man für jede Gleichung der Form

$$a_n x^n + a_{n-1} x^{n-1} + ... + a_1 x + a_0 = 0 \qquad (*)$$

(n eine beliebige natürliche Zahl) alle Wurzeln bestimmen kann.[14]

Einen solchen Algorithmus zu konstruieren ist natürlich noch schwieriger. Dem Wesen nach gehört diese Aufgabe in ein Teilgebiet der höheren Algebra, in die sogenannte Algebra der Polynome, und verlangt die Konstruktion und die Begründung dieses Algorithmus, der natürlich von außerordentlich praktischer Bedeutung ist.

Schon die hier angeführten Beispiele charakterisieren in ausreichendem Maße das natürliche Streben der Mathematiker nach immer mächtigeren Algorithmen, mit denen man immer umfangreichere Klassen von Aufgaben (Aufgaben immer allgemeineren Typs) lösen kann.
Selbstverständlich ist das Lösen von Gleichungen der Form (*) noch lange nicht die Grenze, die man überhaupt erreichen kann. Mehr noch, wollte der Mathematiker in seinem unaufhaltsamen Streben die Aufgabenklassen, für die ein einziger lösender Algorithmus gesucht wird, immer weiter ausdehnen, so müsste er unweigerlich zu der folgenden Problemstellung kommen:

Es ist ein Algorithmus zu konstruieren, der es gestattet, jede beliebige mathematische Aufgabe zu lösen.

Diese Problemstellung ist schon so allgemein, dass sie wahrlich als eine verwegene Herausforderung der gesamten Mathematik eingeschätzt werden kann; außerdem könnte man an ihrer Formulierung bemängeln, dass keinesfalls klar ist, was eigentlich "jede beliebige mathematische Aufgabe" bedeutet.
Allerdings braucht man wegen der großen Anziehungskraft und des Reizes derartiger Probleme sie nicht noch besonders zu propagieren.

Das Problem hat seine Geschichte. Schon der große deutsche Mathematiker und Philosoph Leibniz (1646-1716) träumte von der Entwicklung einer allumfassenden Methode, die eine effektive Lösung jeder Aufgabe gestatten würde.
Obwohl Leibniz einen solchen allumfassenden Algorithmus nicht finden konnte, nahm er an, dass man einmal einen finden würde und dass dann jede Meinungsverschiedenheit zwischen Mathematikern automatisch mit Bleistift und Papier nach diesem Universal-Algorithmus entschieden werden könne.

[14] Genauer: Zu jeder natürlichen Zahl k soll man Näherungswerte für die Wurzeln angeben können, die sich von dem genauen Wert um weniger als $\frac{1}{10^k}$ unterscheiden.

In der weiteren Entwicklung wurde die Fragestellung etwas präzisiert durch eines der wichtigsten Probleme der mathematischen Logik, und zwar das Entscheidungsproblem. Da wir hier nicht die Möglichkeit haben, eine erschöpfende und exakte Darstellung dieses Problemkreises zu geben, beschränken wir uns auf die wichtigsten Züge.

Bekanntlich besteht die axiomatische Methode in der Mathematik darin, dass alle Aussagen (Sätze) einer vorliegenden Theorie durch formal-logische Schlussfolgerungen aus gewissen Aussagen (Axiomen), die man in dieser Theorie ohne Beweis als gültig annimmt, abgeleitet werden.
Die Geometrie war die erste mathematische Disziplin, in der die Axiomatisierung verwirklicht werden konnte. Heute sind bereits fast alle mathematischen Theorien axiomatisiert. In der mathematischen Logik hat man eine spezielle Formelsprache entwickelt, mit der man jede Behauptung einer mathematischen Theorie in Gestalt einer wohlbestimmten Formel schreiben kann.

Wenn wir die Terminologie benutzen, die bei der Untersuchung der Halbgruppen eingeführt wurde, können wir sagen, dass jede solche Formel ein Wort in einem gewissen speziellen Alphabet ist.
Dieses Alphabet enthält neben den in der Mathematik üblichen Zeichen, wie Buchstaben, Klammern usw., noch andere spezielle Zeichen, mit denen die logischen Operationen dargestellt werden (z.B. Verneinung, Konjunktion (Verknüpfung durch "und"), Alternative (Verknüpfung durch "oder"), Implikation (wenn ..., so ...) usw.).

Die wesentliche Analogie zu den Halbgruppen besteht darin, dass der Prozess der logischen Ableitung einer Aussage S aus einer Aussage R mittels formaler Transformationen von Wörtern beschrieben werden kann.

Dieses Verfahren ist der Anwendung der zulässigen Substitutionen in Halbgruppen sehr ähnlich. Deshalb werden wir hier von logischen Kalkülen (Halbgruppen) sprechen, in denen ein System zulässiger Transformationen definiert ist. Diese zulässigen Transformationen sind die elementaren Operationen, aus denen sich beliebige formal-logische Aussagen aufbauen lassen. Ein Beispiel einer solchen zulässigen Transformation ist die Streichung zweier hintereinanderstehender Negationszeichen.
Beispielsweise kann "nicht nicht rot" in "rot" umgeformt werden. (Es ist interessant, diese zulässige Transformation mit der Substitution $aa - \Lambda$ in der Halbgruppe des Beispiels 4 zu vergleichen.)

Die Frage, ob eine Behauptung S aus einer Aussage R innerhalb des logischen Kalküls abgeleitet werden kann, ist die Frage nach der Existenz einer deduktiven Kette, die von dem Wort, welches die Aussage R repräsentiert, zu dem Worte führt, welches die Behauptung S darstellt. Das Entscheidungsproblem lässt sich jetzt so formulieren:

Von je zwei Wörtern (Formeln) R und S eines logischen Kalküls soll man aussagen, ob eine deduktive Kette existiert, die von R nach S führt oder nicht.

Unter der Lösung dieses Problems ist die Angabe eines Algorithmus zu verstehen, der alle Fragen (beliebige R und S) dieses Typs zu beantworten gestattet.

Hieraus ist unschwer zu ersehen, dass ein solcher Algorithmus eine allgemeine Methode zur automatischen Lösung der verschiedenartigsten Probleme aus allen axiomatisch aufgebauten mathematischen Theorien wäre. In der Tat, die Richtigkeit irgendeiner Aussage S (z.B. die Formulierung eines Satzes) bedeutet in dieser Theorie, dass sie formal-logisch aus einem Axio-

mensystem, welches hier die Aussage R vertritt, abgeleitet werden kann.
Verwendet man den Entscheidungsalgorithmus, so kann man sagen, ob die Aussage S in der betrachteten Theorie richtig ist oder nicht.

Mehr noch, im Falle einer bejahenden Antwort könnte man eine entsprechende deduktive Kette finden, und aus ihr ließe sich eine Kette von Schlussfolgerungen aufbauen, die den Beweis der Aussage S bilden würde.
Faktisch würde dieser Algorithmus eine einheitliche effektive Methode sein, mit der man fast alle formulierten und bis heute noch nicht gelösten mathematischen Probleme entscheiden könnte. Diese Tatsache erklärt nicht nur den Reiz, den die Konstruktion eines solchen "allumfassenden" Algorithmus ausstrahlt, und die Anziehungskraft, die die Konstruktion einer "alles könnenden Maschine" ausübt, sondern auch die Schwierigkeiten einer solchen Konstruktion.

Ungeachtet der langen und hartnäckigen Bemühungen vieler berühmter Spezialisten waren die mit der Konstruktion verbundenen Schwierigkeiten unüberwindlich. Ja, ähnliche Schwierigkeiten traten schon auf, als man versuchte, für mathematische Probleme bedeutend spezielleren Typs einen Algorithmus aufzustellen.
Zu diesen Problemen gehört auch das Hilbertsche Problem über diophantische Gleichungen (siehe hierzu § 1) sowie eine ganze Reihe anderer Probleme, auf die wir später zu sprechen kommen.

Im Ergebnis dieser zahlreichen aber vergeblichen Bemühungen um einen solchen Algorithmus kam man zu dem Schluss, dass hier Schwierigkeiten prinzipiellen Charakters vorliegen müssen, und es entstand die Vermutung, dass man nicht für jede Klasse von Aufgaben einen Lösungsalgorithmus konstruieren kann.

Die Behauptung von der algorithmischen Unlösbarkeit gewisser Aufgabenklassen, d.h., der Unmöglichkeit, dafür einen Lösungsalgorithmus anzugeben, ist nicht einfach eine Anerkennung der Tatsache, dass uns kein solcher Algorithmus bekannt ist, oder dass ihn noch keiner gefunden hat.
Diese Behauptung ist gleichzeitig eine Prognose für die Zukunft.

Sie besagt, dass ein solcher Algorithmus niemals gefunden werden kann (mit anderen Worten, dass es ihn nicht gibt). Diese Behauptung bedarf natürlich eines strengen mathematischen Beweises. Über einen Beweis nachzudenken, hat im Moment keinen Sinn, weil wir noch keine exakte Definition des Begriffes "Algorithmus" haben.

Dann ist es natürlich völlig schleierhaft, wie die Nichtexistenz eigentlich bewiesen werden soll. Hier ist es nützlich, daran zu erinnern, dass auch schon früher in der Mathematik Probleme bekannt waren, die sich nicht lösen lassen wollten. Später stellte man dann fest, dass sie mit den damals zur Verfügung stehenden Mitteln gar nicht gelöst werden konnten.
Wir verweisen nur auf die Probleme der Dreiteilung eines Winkels und der Auflösung einer Gleichung durch Radikale.

Aus dem Schulunterricht ist allgemein bekannt, wie man jeden Winkel mit Zirkel und Lineal halbieren kann. Schon im alten Griechenland beschäftigte man sich mit der ähnlichen Aufgabe der Dreiteilung eines Winkels mit Hilfe von Zirkel und Lineal.
Es ist jedoch inzwischen bewiesen worden, dass die Dreiteilung jedes beliebigen Winkels mit diesen Mitteln niemals erreicht werden kann.[15]

[15] Vgl. hierzu etwa W. Breidenbach, Die Dreiteilung des Winkels, Leipzig 1951 oder R. Kochendörffer, Einführung in die Algebra, VEB Deutscher Verlag der Wissenschaften, 2. Aufl., Berlin 1962, S. 203. (Red.)

Aus dem Schulunterricht ist auch bekannt, dass man die Wurzeln einer quadratischen Gleichung mittels einer Formel, in der nur die Koeffizienten der Gleichung auftreten, ausdrücken kann. In dieser Formel tritt außer den arithmetischen Operationen nur noch das Quadratwurzelzeichen auf.
Für Gleichungen dritten und vierten Grades wurden ähnliche Formeln mit Wurzelausdrücken (Radikalen) angegeben, die natürlich wesentlich komplizierter aufgebaut sind und in denen "mehrstufige" ("geschachtelte") Radikale auftreten.[16]

Jedoch war das Suchen nach ähnlichen Formeln für Gleichungen höheren als vierten Grades (bis zum Beginn des XIX. Jahrhunderts) erfolglos, bis schließlich das folgende bemerkenswerte Resultat ermittelt werden konnte:

Für $n \geq 5$ existiert keine Formel, die die Wurzeln einer beliebigen Gleichung n-ten Grades durch die Koeffizienten der Gleichung mittels Radikale ausdrückt.[17]

In den beiden Fällen war der Unmöglichkeitsbeweis zu erbringen, weil man zwei genaue Definitionen hatte, die auf folgende Fragen Antwort gaben:
"Was bedeutet Konstruktion mit Zirkel und Lineal?" und "Was bedeutet Auflösung einer Gleichung in Radikalen ?". Wir erwähnen beiläufig, dass in diesen beiden Definitionen der Sinn gewisser spezieller Algorithmen präzisiert ist, und zwar des Algorithmus zur Lösung einer Gleichung in Radikalen (und nicht eines allgemeinen Algorithmus zur Lösung von Gleichungen) und des Algorithmus zur Dreiteilung beliebiger Winkel mit Hilfe von Zirkel und Lineal (und nicht ein allgemeiner Algorithmus zur Winkeldreiteilung).

Eine exakte Definition für den allgemeinen Begriff des Algorithmus gibt es indessen noch nicht. Deshalb ist die Erarbeitung einer solchen Definition eine der wichtigsten Aufgaben der modernen Mathematik.
Es muss jedoch hervorgehoben werden, dass die Definition des Begriffs des Algorithmus (wie auch jede andere mathematische Definition) nicht nur irgendeine Vereinbarung der Mathematiker darüber ist, was sie unter dem Terminus Algorithmus verstehen wollen. Es ist wesentlich mehr.

Bei der Formulierung sind große Schwierigkeiten zu überwinden. Sie bestehen vor allem darin, dass die vorgeschlagene Definition das Wesen dieses Begriffes richtig wiedergeben muss, denn der Begriff ist. ja faktisch schon da, wenn auch verschwommen. Wir haben ihn auch schon an vielen Beispielen demonstriert.

Um zu einer brauchbaren Formulierung zu kommen, wurden Anfang der dreißiger Jahre des XX. Jahrhunderts viele Untersuchungen zur Sichtung aller jener Mittel durchgeführt, die zur Konstruktion von Algorithmen herangezogen werden dürfen. Auf dieser Grundlage wäre dann eine Definition des Begriffs des Algorithmus zu geben, der dann nicht nur vom Standpunkt der formalen Genauigkeit vollkommen wäre, sondern auch mit dem Wesen des zu definierenden Begriffs tatsächlich übereinstimmte.
Dabei gingen die Forscher von verschiedenen technischen und logischen Überlegungen aus; infolgedessen entstanden mehrere Definitionen des Begriffs des Algorithmus.
Später ergab sich dann auch, dass alle diese Definitionen einander äquivalent waren und folglich den gleichen Begriff definieren; das ist eben der heutige exakte Begriff des Algorithmus.

[16] Vgl. hierzu etwa A. G. Kurosch, Algebraische Gleichungen beliebigen Grades, VEB Deutscher Verlag der Wissenschaften, 2. Aufl., Berlin 1960.
[17] Vgl. Kochendörffer, a. a. O.

Die Tatsache, dass alle Verfahren zur Präzisierung des Begriffs des Algorithmus ungeachtet seiner Vielseitigkeit bisher stets zu dem gleichen Ergebnis führten und im Grunde auch immer führen werden, hat erkenntnistheoretische Bedeutung. Sie bezeugt, dass die erarbeitete Definition das Wesen der Sache erfasst.
Vom Standpunkt der Maschinenmathematik ist diejenige Definition besonders interessant, in der das Wesen des Begriffs durch Analyse von Prozessen offenbar wird, die in einer Maschine realisiert werden können.

Für eine solche strenge mathematische Definition muss man den Arbeitsmechanismus der Maschine in Form eines gewissen Standardschemas darstellen, das seiner logischen Struktur nach einfach, aber trotzdem so präzise sein muss, dass es zum Gegenstand mathematischer Untersuchungen werden kann.
Das wurde zuerst von dem englischen Mathematiker A. M. Turing gemacht, der eine allgemeine und trotzdem einfache Konzeption einer Rechenmaschine vorschlug. Es sei bemerkt, dass die Turing-Maschine schon 1936 beschrieben wurde, d.h. Jahre bevor der erste programmgesteuerte Rechenautomat konstruiert worden war.
Dabei ging Turing nur von der Idee aus, die Arbeit der Maschine mit der Arbeit eines nach einer gewissen strengen Vorschrift operierenden Rechners zu vergleichen. Unsere Darlegung dagegen basiert bereits auf dem allgemeinen Arbeitsprinzip der existierenden elektronischen Rechenautomaten.

7 Die Turing-Maschine

Die Besonderheiten, die eine Turing-Maschine gegenüber den in den §§ 4, 5 beschriebenen elektronischen Rechenautomaten auszeichnen, bestehen in folgendem:

1. In einer Turing-Maschine ist der Prozess der Zerlegung in einfache elementare Operationen in gewissem Sinne bis an die Grenze des Möglichen durchgeführt. So wird beispielsweise die Addition, die in elektronischen Rechenautomaten als einheitliche Elementaroperation behandelt wird, in eine Kette noch einfacherer Operationen zerlegt.

Das verlängert natürlich den Prozess in einer Turing-Maschine wesentlich. Jedoch wird dadurch die logische Struktur stark vereinfacht und in eine für theoretische Untersuchungen geeignete Standardform gebracht.

2. In einer Turing-Maschine hat man sich einen Speicherteil[18] als ein nach beiden Seiten unbeschränktes Band vorzustellen, welches in einzelne Zellen unterteilt ist.

Offenbar kann es in keiner wirklich gebauten Maschine einen unendlichen Speicher (ein unendliches Band) geben. In diesem Sinne ist eine Turing-Maschine nur ein idealisiertes Schema, bei dem eine Vergrößerung des Speichervolumens potentiell möglich ist.
Diese Idealisierung wird durch den schon früher erwähnten Zusammenhang zwischen dem Begriff des Algorithmus und dem Begriff der Maschine mit potentiell unbeschränktem Speicher gerechtfertigt.

Wir gehen jetzt zur ausführlichen Beschreibung einer Turing-Maschine über.

1. Die Maschine verfügt über endlich viele Zeichen (Symbole)

$$s_1, s_2, ..., s_k$$

[18] Und zwar den sogenannten äußeren Speicher.

die das sogenannte äußere Alphabet bilden. In diesem Alphabet werden die Eingabedaten sowie die in der Maschine erarbeiteten Daten geschrieben. Darunter befinde sich auch das Leerzeichen (und zwar sei es durch s_1 angedeutet).
Wird das Leerzeichen in irgendeine Zelle des Bandes (Speichers) eingegeben, so wird das vorher in der Zelle vorhandene Zeichen ausgelöscht, und die Zelle wird leer. Von einer leeren Zelle wollen wir sagen, in ihr stehe das Leerzeichen.
In jedem Stadium der Arbeit der Maschine kann sich in keiner Zelle mehr als ein Zeichen befinden. Alle Daten, die auf dem Bande gespeichert sind, werden durch endlich viele Zeichen des äußeren Alphabets dargestellt. Diese endlich vielen Zeichen (Zeichensatz) sind vom Leerzeichen verschieden und einzeln in gewissen Zellen des Bandes untergebracht.

Zu Arbeitsbeginn befinden sich auf dem Band die Anfangsdaten (Anfangsinformation). Die Maschine arbeitet in aufeinanderfolgenden Takten.
Jedem Takt entspricht eine Transformation der Anfangsinformation in eine Zwischeninformation (am Ende jedes Taktes bildet die Gesamtheit der auf dem Bande befindlichen Zeichen die entsprechende Zwischeninformation).

Als Anfangsinformation kann sich auf dem Band ein beliebiges endliches System von Zeichen des äußeren Alphabets befinden (ein beliebiges Wort dieses Alphabets), die in beliebiger Weise auf die Zellen verteilt sind. Je nach der Anfangsinformation $\mathfrak{A}$ sind zwei Fälle möglich:

a) Nach endlich vielen Takten hält die Maschine an, weil sie das Stopsignal erhielt. Bis zu diesem Zeitpunkt hat sich auf dem Bande eine gewisse Information $\mathfrak{B}$ herausgebildet. Wir wollen dann sagen, dass die Maschine auf die Anfangsinformation $\mathfrak{A}$ anwendbar ist und diese in die Schlussinformation $\mathfrak{B}$ umformt.

b) Die Maschine hält nicht an, und das Stopsignal tritt nicht auf. Wir sagen dann, die Maschine sei auf die Anfangsinformation $\mathfrak{A}$ nicht anwendbar.

Man sagt, dass eine Maschine eine gewisse Klasse von Aufgaben lösen kann, wenn sie auf jede Information anwendbar ist, die in einem bestimmten Kode die Bedingungen für jede einzelne Aufgabe dieses Typs wiedergibt, und diese in eine Information umformt, die in demselben Kode die Lösung dieser Aufgabe darstellt.

2. Das Dreiadresssystem, welches in vielen programmgesteuerten Rechenautomaten angewandt wird, verlangt das Vorhandensein elementarer Operationen, an denen der Inhalt dreier Speicherzellen direkt beteiligt ist.

In einigen programmgesteuerten Rechenautomaten wird jedoch das sogenannte Einadresssystem benutzt. Hier ist an jedem Arbeitstakt nur eine Speicherzelle beteiligt. (Diese wollen wir dann als die aufgerufene Zelle bezeichnen.) So kann z.B. die Addition der Zahlen aus den Zellen β und γ mit anschließendem Transport des Resultats in die Zelle δ des Dreiadresssystems durch drei aufeinanderfolgende Befehle im Einadresssystem ersetzt werden:

a) Transport der Zahl aus der Zelle β in den Addiator,
b) Transport der Zahl aus der Zelle γ in den Addiator,
c) Transport des Resultats in die Zelle δ.

In einer Turing-Maschine ist das System der elementaren Operationen und damit auch das Einadresssystem noch weiter vereinfacht:
In jedem Einzeltakt wird durch den Befehl nur das Zeichen s_i, welches sich in der aufgerufenen Zelle befindet, durch irgendein anderes Zeichen s_j ersetzt. Für $j = i$ bedeutet das, dass der

Inhalt der aufgerufenen Zelle unverändert bleibt; $j = 1$ hat zur Folge, dass das in der aufgerufenen Zelle gespeicherte Zeichen gelöscht wird.

Eine weitere Vereinfachung besteht darin, dass von einem Takt zum unmittelbar folgenden Takt die Adresse der aufgerufenen Zelle sich um nicht mehr als Eins ändern kann, d.h., die im nächsten Takt aufgerufene Zelle ist unmittelbar linker oder rechter Nachbar der zuletzt aufgerufenen Zelle oder aber sie selbst.

Die Idee dieser Vereinfachung besteht darin, dass der für den Prozess benötigte Inhalt irgendeiner Zelle durch schrittweises Abtasten aller Zellen solange gesucht wird, bis die gewünschte Zelle gefunden ist. Das verlängert selbstverständlich den Prozess sehr, bringt aber folgende Annehmlichkeit mit sich:

In den Befehlen des Programms kann man sich an Stelle der beliebigen Adressen der aufzurufenden Zellen auf die Verwendung von nur drei Standardadressen beschränken. Wir werden sie durch spezielle Zeichen darstellen:

R - die rechts benachbarte Zelle ist aufzurufen,
L - die links benachbarte Zelle ist aufzurufen,
M - es ist die gleiche Zelle wie im letzten Takt aufzurufen.

3. Zur Verarbeitung der im Speicher aufbewahrten numerischen Informationen besitzt ein programmgesteuerter Rechenautomat, wie in den §§ 4, 5 beschrieben, ein Rechenwerk $\mathfrak{L}$, welches endlich viele Zustände annehmen kann, die der Addition, der Subtraktion usw. entsprechen.

Damit eine Operation im Rechenwerk durchgeführt werden kann, müssen über bestimmte Kanäle nicht nur die Zahlen, mit denen sie auszuführen ist, sondern auch ein Signal kommen, welches das Rechenwerk auf die betreffende Operation abstimmt, d.h. in den entsprechenden Zustand überführt (siehe Abb. 4b).
Eine Turing-Maschine verarbeitet die Informationen im sogenannten logischen Block, der ebenfalls endlich viele Zustände annehmen kann, welche mit

$$q_1, q_2, ..., q_m$$

bezeichnet seien. Der Block besitzt zwei Eingänge. Durch den einen läuft in jedem Arbeitsstadium der Maschine (in jedem Takt) das Zeichen s_i aus der aufgerufenen Zelle, durch den anderen das Zeichen q_l des Zustandes, der für den Block im gegebenen Takt vorgeschrieben ist.
Das entsprechende "erarbeitete" Zeichen s_j, welches eine eindeutige Funktion der Signale s_i und q_l ist, gelangt durch den Ausgang des Blockes in die aufgerufene Zelle. Die Befehle, die die Arbeit der Maschine in jedem Einzeltakt bestimmen, haben die Form:

$$Rq_l,\ Lq_l,\ Mq_l \qquad (l = 1, 2, ..., m)$$

wobei das erste Zeichen die aufzurufende Zelle bezeichnet (siehe oben) und das zweite den entsprechenden Zustand des logischen Blocks vorschreibt. Die Zeichen R, L, M, q_1, ..., q_m bilden das innere Alphabet der Maschine.

Der logische Block einer Turing-Maschine weist noch eine spezifische Besonderheit auf. In diesem Block werden in jedem Takte auch die Befehle erarbeitet, die zu Beginn des folgenden Taktes in das Steuerwerk gegeben werden.

Deshalb hat der logische Block außer dem Ausgangskanal für das Zeichen s_j noch zwei Ausgangskanäle für die Zeichen des folgenden Befehls. [19]

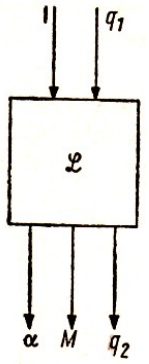


Abb. 5

Abb. 5 zeigt eine schematische Darstellung.

Hierbei ist wesentlich, dass das Ausgangstripel s_j, P, q_l ausschließlich davon abhängt, welches Eingangspaar s_i, q_n in diesem Takt in den Block einlief.
Das bedeutet, dass der logische Block eine Funktion realisiert, die jedem Paar von Zeichen s_i, q_n (es gibt insgesamt km Paare) ein Tripel von Zeichen s_j, P, q_l zuordnet.
Diese Funktion, die wir die logische Funktion der Maschine nennen wollen, lässt sich bequem in einer rechteckigen Tabelle darstellen, deren Spalten durch die Zeichen der Zustände und deren Zeilen durch die Zeichen des äußeren Alphabets markiert sind.

In jedem Feld dieser Tabelle steht das entsprechende Ausgangstripel. Diese Tabelle nennen wir das Funktionsschema der Maschine; Abb. 6 zeigt ein Beispiel.

Abb. 6

	q_1	q_2	q_3	q_4	q_5
Λ	ΛRq_4	ΛLq_3	ΛRq_1	ΛMq_5	ΛMq_5
$\|$	αMq_2	βMq_1	$\|Rq_1$	$\|Lq_1$	$\|Mq_5$
α	αLq_1	αRq_2	$\|Lq_3$	ΛRq_4	αMq_5
β	βLq_1	βRq_2	ΛLq_3	$\|Rq_4$	βMq_5

Aus dieser Beschreibung ergibt sich, dass die Arbeit einer Turing-Maschine Maschine völlig bestimmt ist durch die logische Funktion, die durch den logischen Block realisiert wird. Mit anderen Worten, zwei Turing-Maschinen mit gleichem Funktionsschema sind nicht voneinander zu unterscheiden, solange wir uns nur dafür interessieren, wie sie arbeiten.

Andererseits kann man die Struktur einer Maschine in Form eines Strukturschemas angeben. Daraus ist zu ersehen, aus welchen Organen die Maschine besteht und wie diese ineinandergreifen. Alle Turing-Maschinen haben das gleiche Strukturschema (siehe Abb. 7).

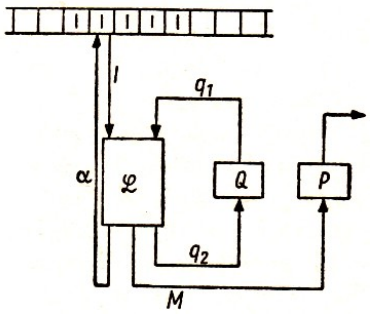


Abb. 7

In diesem Schema ist der Speicher in einen inneren und einen äußeren unterteilt. Der äußere Speicher wird von den Zellen des unendlichen Bandes gebildet, die zur Speicherung der Information in den Symbolen des äußeren Alphabets bestimmt sind.
Der innere Speicher besteht aus zwei Zellen zur Speicherung des nächsten Befehls: Die Q-Zelle enthält das Zeichen des Zustandes und die P-Zelle das Verschiebungszeichen.
In diesen beiden Zellen werden die Zeichen q_l, P bis zum Beginn des folgenden Arbeitstaktes verzögert, da sie bereits während des laufenden Taktes am Ausgang des logischen Blocks auftreten und erst im nächsten Takt im Steuerwerk benötigt werden.

[19] Unter P wird hier irgendeines der Zeichen R, L, M verstanden.

Die Funktion des Steuerwerks hat sich dadurch außerordentlich vereinfacht. Es hat im wesentlichen nur noch die Verschiebung des Bandes entsprechend dem eingelaufenen P-Wert sicherzustellen. Das Zustandszeichen aus der Q-Zelle könnte man faktisch auch direkt in $\mathfrak{L}$ eingeben, als sogenannte Rückkopplung, bei der das im vorigen Takt in L verarbeitete Zeichen q_l in $\mathfrak{L}$ eingeht.

Die Arbeit einer Turing-Maschine läuft auf folgende Weise ab.
Vor Arbeitsbeginn wird die Anfangsinformation (in unserer Abb. 7 die fünf aufeinanderfolgenden Striche) auf das Band aufgetragen und eine bestimmte Zelle (in der Zeichnung die Zelle, die den vierten Strich, von rechts gezählt, enthält) in das "Gesichtsfeld" der Maschine gerückt. In die Zellen Q und P werden außerdem die Zeichen für den Anfangszustand und die Anfangsverschiebung (sagen wir q_1 und M) eingegeben.

Der weitere Prozess läuft dann automatisch ab und wird auf eindeutige Weise durch das Funktionsschema der Maschine bestimmt. Wir wollen uns ansehen, wie die Maschine beispielsweise arbeiten würde, wenn das Funktionsschema der Abb. 6 vorgegeben ist.

Erster Takt. Aufgerufen wird die Anfangszelle (Verschiebung M) im Zustand q_1. Dieser Zelle wird das Zeichen | (Strich) entnommen.
Ergebnis: Ausgangstripel $\alpha R q_2$, d.h., das Zeichen | wird durch das Zeichen α ersetzt, und in den Zellen P, Q wird bis zum folgenden Takt der nächste Befehl Mq_2 gespeichert.

Zweiter Takt. Aufgerufen wird das Zeichen α aus der gleichen Zelle (Verschiebung M) im Zustand q_2.
Ausgangstripel: $\alpha R q_2$, d.h., das Zeichen α wird nicht geändert, und der neue Befehl lautet Rq_2.

Dritter Takt. Es wird der Strich aus der rechts benachbarten Zelle (Verschiebung R) im Zustand q_2 geholt.
Resultat: Das Zeichen | wird durch β ersetzt, und der neue Befehl Mq_1 gespeichert usw.

Wie aus der letzten Spalte des Funktionsschemas zu ersehen ist, hält die Maschine nur unter der Bedingung an, dass in einem gewissen Stadium des Prozesses der Zustand q_5 auftritt.
In der Tat, keines der in den aufgerufenen Zellen stehenden Zeichen wird durch ein anderes ersetzt. Die Maschine wird diese Arbeit ohne je anzuhalten immer wieder ausführen, weil die Verschiebung M und der Zustand q_5 bestehen bleiben.

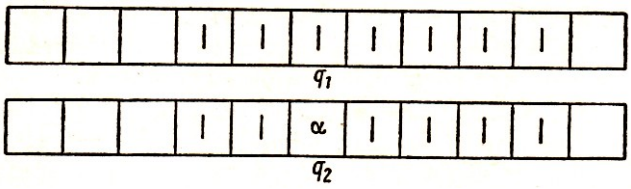
Das ist aber gerade ein gewisser Stopzustand, der den Endzustand des Prozesses in solchen Fällen signalisiert, in denen die Maschine auf die ihr vor Beginn eingegebene Information anwendbar war.

Im Grunde kann auch ein rechnender Mensch das Funktionsschema verwenden. Es ist für ihn ein gewisses Standardverfahren, das den Algorithmus zur Transformation der in einem äußeren Alphabet geschriebenen Anfangsdaten in das entsprechende im gleichen Alphabet geschriebene Resultat beschreibt.
Faktisch haben wir mit dem Funktionsschema der Abb. 6 das Wort aus fünf Strichen gerade so verarbeitet und hierbei erkannt, dass es die Turing-Maschine ebenso macht.

Der größeren Anschaulichkeit halber werden wir künftig in ähnlichen Fällen sogenannte Konfigurationen verwenden. Unter der k-ten Konfiguration verstehen wir die graphische Darstellung des Bandes mit der Information, die sich zu Beginn des k-ten Taktes darauf befindet, wobei unter der aufgerufenen Zelle das Zeichen des Zustandes geschrieben steht, in dem sich der

Block $\mathfrak{L}$ zu Beginn desselben Taktes befindet.
Dadurch ist aus der k-ten Konfiguration deutlich das Eingangspaar ablesbar, und im Funktionsschema kann man das Ausgangstripel bestimmen und damit auch die $(k+1)$-te Konfiguration.

Im oben analysierten Beispiel haben die erste und die zweite Konfiguration folgendes Aussehen



mit den Eingangspaaren $|q_1$, αq_2. Der Übergang von der ersten zur zweiten Konfiguration wird durch das Ausgangstripel $\alpha M q_2$, welches nach dem Schema der Abb. 6 dem Eingangspaar $|q_1$ entspricht, vermittelt.

Das Funktionsschema wollen wir noch etwas einfacher schreiben, wodurch es übersichtlicher und zur Auswertung der Konfigurationen handlicher wird. Wir wollen nämlich nicht mehr die vollständigen Ausgangstripel $s_j P q_l$ hinschreiben, sondern die s_j und q_l dann weglassen, wenn sie gleich den Eingangszeichen sind. Auch das Zeichen M, das andeutet, dass keine Verschiebung stattfindet, soll in Zukunft fortgelassen werden.
Dadurch kann man insbesondere die Spalten streichen, welche dem Stopzustand entsprechen. Das vereinfachte Schema der Abb. 6 ist in Abb. 8 gezeigt, in der der Stopzustand durch das Symbol "!" angedeutet ist.

Abb. 8

	q_1	q_2	q_3	q_4
Λ	$\Lambda R q_4$	$\Lambda L q_3$	$\Lambda R q_1$	!
$\|$	αq_2	βq_1	$R q_1$	$L q_1$
α	L	R	$\|L$	ΛR
β	L	R	ΛL	$\|R$

Aus der Spalte q_1 des Schemas der Abb. 8 ist wesentlich einfacher als aus der Abb. 6 zu ersehen, dass nach einem α im Zustand q_1 die Maschine eine ganze Serie von Linksverschiebungen über alle hintereinanderstehenden α und β ausführt, ohne den Zustand q_1 und den Inhalt der aufgerufenen Zelle zu verändern.

Erst der erste Strich oder die erste leere Zelle unterbrechen diese Arbeitsweise, und nur unter diesen Voraussetzungen verlässt die Maschine den Zustand q_1. Das Zeichen "!" werden wir künftig stets zur Bezeichnung des Stopzustandes verwenden.

8 Realisierung eines Algorithmus in einer Turing-Maschine

In diesem Paragraphen zeigen wir an einer Reihe von Beispielen, wie eine Turing-Maschine aufgebaut ist, die einige einfache arithmetische Algorithmen realisieren kann, und wie dieser Prozess in der Maschine realisiert wird.
In Übereinstimmung mit dem Inhalt des vorhergehenden Paragraphen werden wir unter der Struktur einer Maschine die Struktur des Funktionsschemas ihres logischen Blocks verstehen.

Dieses Funktionsschema ist eine gewisse Standardform zur Beschreibung eines Algorithmus.

Außerdem werden wir noch einige allgemeinere Überlegungen über die Methodik des Aufbaus von Turing-Maschinen (von Funktionsschemata) anstellen.

I. Algorithmus des Übergangs von n zu $n+1$ im Dezimalsystem

Es sei eine Aufgabe folgenden Typs zu lösen:
Gegeben ist die Dezimaldarstellung einer natürlichen Zahl n. Gesucht ist die Darstellung von $n+1$ im Dezimalsystem.

Abb. 9

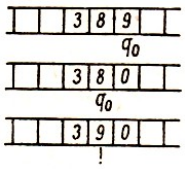
	q_0	q_1
0	1 !	
1	2 !	
2	3 !	
3	4 !	
4	5 !	
5	6 !	
6	7 !	
7	8 !	
8	9 !	
9	0 L	
Λ	L	
|	L	$\Lambda L q_0$

Dazu nehmen wir ein äußeres Alphabet, welches aus den zehn Ziffern 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 und dem Leerzeichen Λ besteht. Die Maschine besitzt nur zwei Zustände: q_0 (Arbeitszustand) und ! (Stop).

Die gegebene Zahl n und das Resultat $n+1$ werden im Dezimalsystem geschrieben, wobei jede Ziffer in einer Zelle untergebracht wird (die Zellen folgen lückenlos aufeinander). Das entsprechende Funktionsschema ist in Abb. 9 dargestellt; vorläufig sind die letzte Zeile und die letzte Spalte nicht zu berücksichtigen (der Sinn der erweiterten Tabelle wird später erklärt werden).

Zu Arbeitsbeginn möge im Gesichtsfeld der Maschine die Stelle der Einer der Zahl n eingestellt sein und die Maschine sich im Zustand q_0 befinden. Wenn die eingestellte Ziffer von 9 verschieden ist, so bleibt die Maschine schon nach dem ersten Arbeitstakt stehen, nachdem diese Ziffer durch die ihr gemäß dem Schema entsprechende ersetzt wurde.

War aber die letzte Ziffer eine 9, so ersetzt sie die Maschine durch eine 0 und führt eine Linksverschiebung aus (zur benachbarten nächsthöheren Stelle) und setzt ihre Arbeit im Zustand q_0 fort (auf diese Weise erfolgt die Zehner-Übertragung).

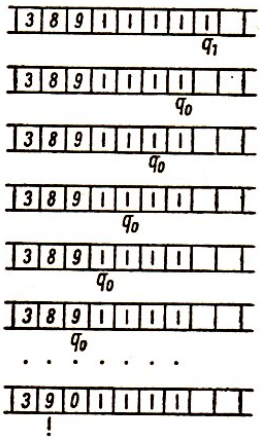


Endet die Zahl auf k Neunen, so bleibt die Maschine nach genau $k+1$ Takten stehen. In Abb. 10 sind die entsprechenden Konfigurationen für $n = 389$ dargestellt.
Wir wollen uns jetzt den Sinn der erweiterten Tabelle in Abb. 9 klarmachen.

Sie gibt das Funktionsschema einer Maschine wieder, in der es noch einen weiteren Zustand q_1 gibt. Außerdem ist auch das äußere Alphabet um ein Symbol, den "Strich", erweitert werden. Befindet sich die Maschine zu Arbeitsbeginn im Zustand q_0 und ist auf dem Bande kein Strich, so wird die Arbeitsweise dieselbe sein, wie bei der Maschine des vorhergehenden Beispiels. Das geht unmittelbar daraus hervor, dass die letzte Spalte und die letzte Zeile der Tabelle gar nicht zur Anwendung kommen. Also kann diese Maschine auch zur Realisierung des vorhergehenden Algorithmus verwendet werden. Da die Maschine aber noch einiges mehr leistet, wollen wir sie etwas näher untersuchen.

Es sei auf das Band die Zahl n im Dezimalsystem aufgetragen.
Außerdem mögen in einigen unmittelbar rechts benachbarten Zellen Striche gespeichert sein, je einer in einer Zelle. Wir wollen nun untersuchen, wie die Maschine arbeitet, wenn sich zu Arbeitsbeginn der am weitesten rechts stehende Strich im Gesichtsfeld der Maschine befindet und sie selbst auf den Zustand q_1 abgestimmt ist.

Im ersten Takt (Eingangspaar $q_1|$) wird dieser Strich gelöscht, und es erfolgt eine Verschiebung nach links mit Übergang in den Zustand q_0 (Ausgangstripel $\Lambda L q_0$). In den folgenden Takten setzt die Maschine die Verschiebung nach links im Zustand q_0 fort, und zwar solange, bis sie auf die Einerstelle der Zahl n stößt.



Von diesem Moment an läuft unser vorher behandelter Algorithmus ab, d.h., die Darstellung der Zahl n geht in die der Zahl $n+1$ über, und der Prozess ist abgeschlossen.
Kürzer ausgedrückt, die Maschine verkleinert die Anzahl der Striche um Eins und führt im Dezimalsystem den Übergang von der Zahl n zur Zahl $n+1$ aus.

Diesen Prozess wollen wir als Kontrollübergang von der Dezimaldarstellung von n zur Dezimaldarstellung $n+1$ bezeichnen. In der Abb. 11 sind die Konfigurationen für einen Satz von fünf Strichen und für $n = 389$ dargestellt.

II. Der Algorithmus zur Überführung in das Dezimalsystem

Wir konstruieren das Funktionsschema einer Maschine (Algorithmus), welche eine Aufgabe des folgenden Typs löst:
Gegeben ist eine endliche Gesamtheit von Strichen, die lückenlos in aufeinanderfolgenden Zellen stehen (eine solche Gesamtheit wollen wir einen Satz nennen). Die Anzahl dieser Striche soll im Dezimalsystem niedergeschrieben werden. Kürzer: Es ist ein Satz Striche abzuzählen.

Abb. 12

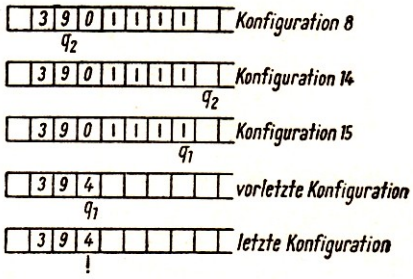
	q_0	q_1	q_2
0	1 q_2	!	R
1	2 q_2	!	R
2	3 q_2	!	R
3	4 q_2	!	R
4	5 q_2	!	R
5	6 q_2	!	R
6	7 q_2	!	R
7	8 q_2	!	R
8	9 q_2	!	R
9	9 L	!	R
Λ	1 q_2	!	Lq_1
|	L	$\Lambda L q_0$	R

In Abb. 12 ist ein solches Schema beschrieben. Um sich davon zu überzeugen, dass dieses Schema wirklich die verlangte Maschine (den gewünschten Algorithmus) beschreibt, vergleichen wir dieses Schema zweckmäßigerweise mit der erweiterten Tabelle der Abb. 9.
Die Spalte q_0 im Schema der Abb. 12 unterscheidet sich von der Spalte q_0 im Schema der Abb. 9 nur dadurch, dass in ihr überall an Stelle des Zustandes "!" der neue Zustand q_2 auftritt.

Die Unterschiede in den Spalten q_1 haben keinen wesentlichen Einfluss auf die Arbeitsweise des Schemas der Abb. 9.
Wenn sich also auf dem Band die Zahl n im Dezimalsystem befindet und rechts von ihr ein Satz Striche steht, wenn ferner in das Gesichtsfeld der Maschine wie früher der am weitesten rechts stehende Strich eingestellt und die Maschine selbst auf den Zustand q_1 abgestimmt ist, dann wird in der Maschine derselbe Prozess wie beim Schema der Abb. 9 ablaufen.

Es wird ein Strich des Satzes gelöscht und die Dezimaldarstellung der Zahl n durch die von $n+1$ ersetzt. Während aber im Schema der Abb. 9 jetzt der Zustand "!" eintritt, d.h. der Prozess abbricht, tritt gemäß dem Schema der Abb. 12 der Zustand q_2 ein, und der Prozess wird fortgesetzt.

Nimmt man insbesondere als erste Konfiguration die von Abb. 11 an, so ergibt sich die achte Konfiguration wie sie in Abb. 13 dargestellt ist. Wie der Prozess fortgesetzt wird, ist aus der Zustandsspalte q_2 zu ersehen:



Es erfolgt zuerst eine Serie Rechtsverschiebungen über alle Ziffern und Striche hinweg bis zur ersten leeren Zelle rechts. Dann liegt das Eingangspaar Λq_2 (siehe Konfiguration 14 der Abb. 13) vor, welches eine Verschiebung nach links mit gleichzeitigem Zustandswechsel nach q_1 (Konfiguration 15 der Abb. 13) bewirkt.

Auf diese Weise erscheint im Gesichtsfeld der Maschine wiederum der am weitesten rechts stehende Strich des Satzes im Zustand q_1, womit ein Arbeitszyklus abgeschlossen ist und der zweite analog dem ersten begonnen wird. Beim zweiten Zyklus wird ein weiterer Strich gelöscht, und die Darstellung der Zahl $n+1$ wird durch die der Zahl $n+2$ ersetzt.
Wenn ursprünglich in dem Satz k Striche waren, so werden nach k Arbeitszyklen alle Striche gelöscht sein, und an Stelle der ursprünglichen Darstellung der Zahl n wird die der Zahl $n+k$ auftreten.

Nach Beendigung des k-ten Zyklus geht die Maschine wiederum in den Zustand q_1 über. Im Gesichtsfeld der Maschine taucht aber kein Strich mehr auf (denn die Striche sind alle gelöscht) sondern die erste Ziffer (d.h. die Einerstelle) der Zahl $n+k$ (vorletzte Konfiguration der Abb. 13). Wie aus dem Schema der Abb. 12 zu ersehen ist, wird in diesem Falle der Stopbefehl ausgeführt (letzte Konfiguration der Abb. 13).

Wenn bei Arbeitsbeginn der Maschine auf dem Band die Zahl Null stand und ein Satz von k Strichen vorhanden war, so ergibt sich aus diesen Ausführungen, dass die Maschine alle Striche auslöscht und an Stelle der Null die Zahl $0+k$, d.h. k setzt.
Zu Beginn kann man aber auch ohne Null auskommen, wenn an Stelle der Null das Leerzeichen Λ steht. In den Zuständen q_0 und q_1 benimmt sich die Maschine genau so, als wenn dort eine Null stehen würde (siehe das Schema der Abb. 12).

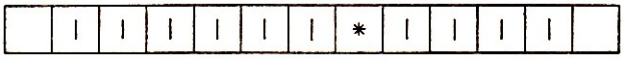
Also beschreibt das vorgeschlagene Schema der Abb. 12 tatsächlich einen Algorithmus zum Umschreiben eines Satzes von Strichen in eine Dezimalzahl.

Übungsaufgabe.
Analog zu I ist das Funktionsschema einer Maschine (Algorithmus) zum Übergang von der Dezimaldarstellung einer Zahl n zu der von $n-1$ (für $n \geq 1$) anzugeben. Ferner stelle man analog zu II das Funktionsschema einer Maschine für das Umschreiben der Dezimaldarstellung einer Zahl n in einen Satz von n Strichen auf.

Wir betrachten noch einige Beispiele für Turing-Maschinen zur Lösung arithmetischer Aufgaben. In diesen Aufgaben sind sowohl die Anfangswerte als auch die Resultate natürliche Zahlen.
Wie wollen annehmen, dass in einer Maschine jede natürliche Zahl durch einen Satz mit der ihr entsprechenden Strichzahl gegeben ist. Kommen in einer Aufgabe mehrere natürliche Zahlen vor, so werden die Striche durch irgendwelche speziellen Zeichen getrennt, z.B. durch einen Stern *. Auch dieses Zeichen gehört dann zum äußeren Alphabet.

III. Additionsalgorithmus

Auf das Band sei ein Paar von Zahlen aufgetragen, beispielsweise



Gesucht wird die Summe, also in diesem Fall

Wir bemerken, dass man diesen Prozess nicht einfach auf die Löschung des Sterns zurückführen kann, weil dann zwischen den Strichen eine Leerzelle auftreten würde, folglich der Satz übrigbleibender Striche (wegen der Lücke) nicht als die Darstellung

Abb. 14

	q_0	q_1	q_2
$\|$	$\Lambda R q_2$	L	R
Λ	R	$R q_0$	$\|q_1$
$*$	$\Lambda!$	L	R

einer natürlichen Zahl angesehen werden könnte. Gemäß dem in Abb. 14 dargestellten Funktionsschema wird die Arbeit der Maschine wie folgt verlaufen:
Anfangsbedingungen: Im Gesichtsfeld steht der am weitesten links stehende Strich, die Maschine ist im Zustand q_0 (Konfiguration 1 der Abb. 15).

Erster Takt. Löschung des Strichs in der aufgerufenen Zelle, Rechtsverschiebung (im Gesichtsfeld steht der folgende Strich) und Übergang in den Zustand q_2 (Konfiguration 2).
Die folgenden Takte sind, wie aus der Spalte q_2 zu ersehen ist, Rechtsverschiebungen über alle Striche und den Stern hinweg bis zur ersten Leerzelle (Konfiguration 12).

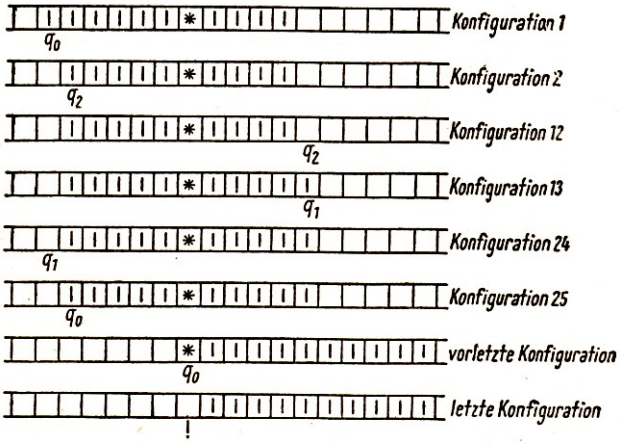


Abb. 15

Dann wird in diese Leerzelle (Eingangspaar Λq_2) ein Strich geschrieben und die Maschine in den Zustand q_1 übergeführt (Konfiguration 13). Im Zustand q_1 führt sie dann wieder Linksverschiebungen über alle Striche und den Stern hinweg bis zur ersten links stehenden Leerzelle aus (Konfiguration 24).
Sodann (Eingangspaar Λq_1) kommt es zu einer Rechtsverschiebung, und im Gesichtsfeld erscheint der erste stehengebliebene Strich links vom Stern, während die Maschine in den Zustand q_0 übergeht (Konfiguration 25).

In diesem Zyklus wurde ein Strich des linken Summanden auf den rechten Summanden übertragen. Wenn ursprünglich links vom Stern k Striche standen, so stehen sie nach k Zyklen

rechts vom Stern. Beim $(k+1)$-ten Lauf nach rechts im Zustand q_0 erscheint im Gesichtsfeld der Maschine kein Strich mehr (es gibt links vom Stern keine Striche mehr), sondern der Stern selbst (vorletzte Konfiguration).

Danach (Eingangspaar $*q_0$) wird der Stern gestrichen, und die Maschine hält an (letzte Konfiguration). Damit ist aber schon die gewünschte Summe ermittelt.

IV. Algorithmus zur wiederholten Addition und zur Multiplikation

Wir untersuchen, wie man das Schema der Abb. 14 abändern muss, damit ein endloser Prozess entsteht. Es soll allerdings vorausgesetzt werden, dass das Zahlenpaar m, n so auf das Band geschrieben ist, wie bei der oben behandelten Addition, z.B.

Der Prozess soll so verlaufen, dass die linke Zahl m zur rechten, dann die linke zur Summe $n+m$, danach zur Summe $n+2m$ usw. hinzugefügt wird, ohne dass der Prozess irgendwann abbricht.
Dabei darf der linke Summand offenbar nach der ersten Addition nicht verschwinden, sondern er muss nach jeder Addition wieder hergestellt werden, damit man ihn wieder zum rechts vom Stern stehenden Summanden hinzufügen kann. Das kann man beispielsweise dadurch erreichen, dass die Striche des linken Satzes nicht gelöscht, sondern durch irgend ein Zeichen ersetzt werden.

Abb. 16

	q_0	q_1	q_2	q_3
|	$\alpha R q_2$	L	R	
Λ	R	Rq_0	$\|q_1$	Rq_0
$*$	q_3	L	R	L
α	R	Rq_0	$\|q_1$	$\|L$

In Abb. 16 ist ein Schema dargestellt, in dem der Buchstabe α die Rolle dieses Merkzeichens übernimmt. Aus den angeführten Gründen wird in der ersten Zeile des Schemas der Abb. 14 statt des Λ das Zeichen α gesetzt.
Außerdem gibt es im Schema der Abb. 16 noch eine vierte Zeile für den Buchstaben α. Die ersten drei Fächer stimmen mit dem Schema der Abb. 14 überein, wenn man für α wieder Λ setzt.
Damit der Prozess nicht nach der ersten Addition abbricht, muss im Schema der Abb. 16 anstelle des Zeichens "!", welches in Abb. 14 beim Eingangspaar $*q_0$ erscheint, ein anderer Zustand auftreten, der die Fortsetzung des Prozesses sicherstellt. In unserem Falle ist es der zusätzliche Zustand q_3. Für ihn muss nun noch die entsprechende Spalte eingeführt werden. Da im Schema der Abb. 16 das Zeichen "!" (Stop) nicht auftritt, wird der Prozess endlos laufen.

Der Leser verifiziert nun ohne Mühe, dass die Maschine auch wirklich die links vom Stern stehende Zahl zu der rechts vom Stern stehenden unbeschränkt oft hinzufügt. Standen bei Arbeitsbeginn rechts vom Stern keine Striche (d.h., war die zweite Zahl Null), so erscheinen rechts vom Stern erst m, dann $2m$, sodann $3m$ Striche usw. ohne Ende.

Übung. Man stelle ein Funktionsschema für den Multiplikationsalgorithmus auf.

Hinweis. Als Grundlage nehme man das vorhergehende Schema und ändere es so ab, dass der wiederholte Additionsprozess nicht unbeschränkt fortgesetzt wird, sondern nur so oft, wie

Einheiten im Multiplikator stehen (nach jedem Additionszyklus wird ein Strich aus dem Multiplikatorsatz gelöscht).

V. Der Euklidische Algorithmus

Wir untersuchen jetzt, wie eine Turing-Maschine für den Euklidischen Algorithmus zum Aufsuchen des größten gemeinsamen Teilers zweier Zahlen a und b aussieht.
Dieser Algorithmus ist zweimal von uns beschrieben worden: Das erste Mal in Worten, das zweite Mal in Form eines Programms für einen elektronischen Rechenautomaten.

Dieses Mal geben wir den Algorithmus in Form eines Funktionsschemas einer Turing-Maschine an und verfolgen den Subtraktionsprozess in der Maschine. Der Algorithmus setzt sich zusammen aus abwechselnden Vergleichs- und Subtraktionszyklen, die den elementaren Operationen Vergleich und Subtraktion in einem programmgesteuerten Automaten entsprechen.
Das entsprechende Funktionsschema ist in Abb. 8 dargestellt. Das äußere Alphabet besteht aus den vier Zeichen

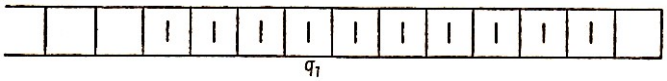
$$\Lambda, |, \alpha, \beta$$

Die natürlichen Zahlen werden hier nach wie vor durch die entsprechenden Sätze von Strichen dargestellt. Um uns nicht in Einzelheiten zu verlieren, die nichts mit dem Wesen der Sache zu tun haben und unsere Untersuchungen nur unnötig verkomplizieren würden, werden wir die Sätze von Strichen der gegebenen Zahlen ohne Lücke oder Trennungszeichen aufeinander folgen lassen.

Zu Beginn des Prozesses liege die Zelle mit dem am weitesten rechts liegenden Strich der ersten Zahl im Gesichtsfeld der Maschine. Nach der Analyse des Algorithmus wird der Leser ohne besondere Schwierigkeiten das verwendete Funktionsschema (als Übungsaufgabe) so abändern können, dass die Maschine auch bei anderen Anfangsbedingungen die Aufgabe einwandfrei löst (z.B. wenn die Sätze durch einen Stern getrennt sind und im Gesichtsfeld der Maschine eine Leerzelle steht).

Wir wollen noch darauf hinweisen, dass die Buchstaben α und β die Rolle von Merkzeichen spielen, wie sie auch der Rechner (gewöhnlich in Form von Strichen) macht, um an gewisse Umstände, die sich im Verlauf des Prozesses ergeben, erinnert zu werden.

Die weitere Beschreibung werden wir für den Fall $a = 4$, $b = 6$ mittels Konfigurationen anschaulich machen. Die erste Konfiguration sieht so aus:



Im Vergleichszyklus treten nur die Zustände q_1, q_2 und im Subtraktionszyklus nur die Zustände q_3, q_4 auf.

Wir verfolgen jetzt den Prozess im einzelnen. Zuerst vergleicht die Maschine die Zahlen, die auf dem Band dargestellt sind, um die größere von ihnen zu ermitteln. Hierbei geht sie genau so vor, wie es auch ein Mensch machen würde, der zwei lange, nicht sofort zu übersehende Folgen von Einsen vergleichen soll.
Er würde abwechselnd in beiden Folgen je eine Eins markieren (z.B., indem er sie mit Strichen versieht). Ist eine Folge ausgeschöpft, so ist sofort geklärt, welche der beiden Folgen länger ist.

Die Maschine ersetzt einen Strich der ersten Zahl durch das Symbol α und danach einen Strich

der zweiten durch das Symbol β. Dann kehrt sie wiederum zu den Strichen der ersten Zahl zurück und ersetzt einen weiteren Strich durch α und danach wieder einen Strich der zweiten Zahl durch β usw.

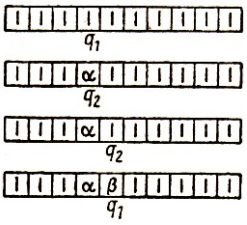


Abb. 17

Die Konfigurationen der ersten vier Takte sind in Abb. 17 dargestellt. Am Ende des vierten Taktes hat die Maschine schon einen Strich jeder Zahl markiert und begibt sich durch Linksverschiebungen auf die Suche nach dem nächsten noch nicht markierten Strich der linken Zahl. Nach einigen Takten liegt die Konfiguration I der Abb. 18 vor, d.h., die erste Zahl ist schon ausgeschöpft, die zweite noch nicht.

Da das Suchen erfolglos ist, entsteht die Konfiguration II, und der Vergleichszyklus ist schon abgeschlossen, wobei nur die Zustände q_1 und q_2 benötigt wurden. Der folgende Takt liefert die Konfiguration III.

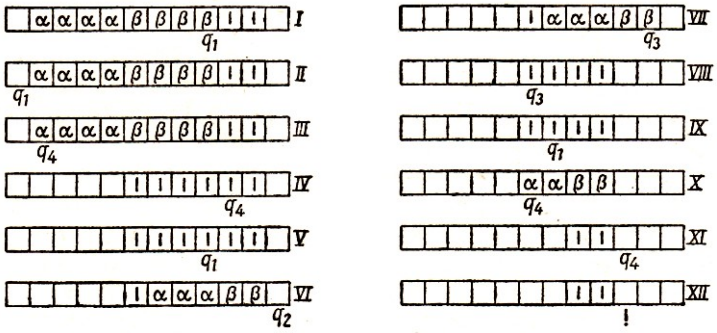


Abb. 18

Wie aus der Spalte q_4 zu ersehen ist (Abb. 8), werden jetzt bei den Rechtsverschiebungen alle α durch das Leerzeichen und alle β durch Striche ersetzt (d.h., alle α werden gelöscht). Nachdem das letzte β durch einen Strich ersetzt ist, befindet sich die Konfiguration IV der Abb. 18 auf dem Band.

Der folgende Takt liefert die Konfiguration V. Nach dem Vergleichszyklus kommt der Zyklus der Subtraktion der ersten von der zweiten Zahl. Hierbei wird die kleinere Zahl a gelöscht und die größere in a und $b-a$ zerlegt.
Die Zelle, in der der letzte Strich der ersten Zahl steht, wird aufgerufen, und die Maschine ist wiederum im Zustand q_1. Das bedeutet, dass die Aufgabe für die Zahlen a und b auf die gleiche Aufgabe für die Zahlen a und $b-a$ zurückgeführt ist. Hierauf basiert aber gerade, wie wir schon wissen, der Euklidische Algorithmus.

Nun kommt natürlich wiederum ein Vergleichszyklus. Diesmal ist jedoch die rechte Zahl die kleinere. Das stellt sich heraus, nachdem die Maschine drei Striche der ersten Zahl ersetzt hat und es in der zweiten keinen Strich mehr gibt, d.h., es liegt die Konfiguration VI vor.

Der folgende Takt erzeugt die Konfiguration VII, und mit ihr beginnt der Zyklus der Subtraktion der zweiten Zahl von der ersten, d.h. das Löschen aller β und das Ersetzen der α durch Striche. Nach dem Ersetzen des am weitesten links stehenden α durch einen Strich befindet

sich auf dem Band die Konfiguration VIII und danach IX, womit der Subtraktionszyklus abgeschlossen ist und der folgende Vergleichszyklus beginnt usw.

Der Prozess wird solange fortgesetzt, bis die Aufgabe auf zwei gleiche Zahlen (in unserem Falle ist das schon erreicht) zurückgeführt ist. Dann beginnt der letzte Vergleichszyklus, der auf das Resultat führen muss. In der Tat, nachdem die Konfiguration X entstanden ist, erzeugt die Subtraktion die Konfiguration XI und schließlich die Resultatkonfiguration XII.

VI. Komposition von Algorithmen

Zur Konstruktion neuer Funktionsschemata kann man oft nutzbringend schon bekannte Schemata verwenden. Das ist immer dann möglich, wenn der zu konstruierende Algorithmus sich aus schon bekannten zusammensetzen lässt.
Wie wollen das an einem Beispiel näher erläutern.

Es soll das Funktionsschema eines Algorithmus angegeben werden, der den größten gemeinsamen Teiler zweier durch Striche gegebener Zahlen a, b ermittelt und diesen im Dezimalsystem niederschreibt. Diesen Algorithmus kann man durch Komposition, d.h. aufeinanderfolgende Anwendung der beiden schon bekannten Algorithmen erhalten. Es wird zuerst der Algorithmus aus Abschnitt V und dann der aus Abschnitt II benötigt.
Das fertige Schema ist in Abb. 19 dargestellt, welches sich aus den Schemata der Abb. 8 und 12 zusammensetzt. Dazu werden erst die Zustände der Abb. 12 in p_0, p_1, p_2 umbenannt, damit man sie von den Zuständen der Abb. 8 unterscheiden kann.

Damit der Übergang von dem einen zum anderen Schema möglich ist, wurde das Stopzeichen "!" der Abb. 8 durch p_2 ersetzt. Damit ist der Aufbau des veränderten Schemas schon erledigt. (Die restlichen, von uns nicht ausgefüllten Fächer können mit beliebigen Ausgangstripeln ersetzt werden, weil die entsprechenden Eingangspaare in unserem Prozess nicht auftreten.)

Jetzt ist es nicht schwer zu verifizieren, dass das Schema der Abb. 19 den gewünschten Prozess beschreibt.

Abb. 19

	q_1	q_2	q_3	q_4	p_0	p_1	p_2
0					1 p_2	!	R
1					2 p_2	!	R
2					3 p_2	!	R
3					4 p_2	!	R
4					5 p_2	!	R
5					6 p_2	!	R
6					7 p_2	!	R
7					8 p_2	!	R
8					9 p_2	!	R
9					0 L	!	R
Λ	$\alpha R q_4$	$L q_3$	$R q_1$	p_2	1 p_2	!	$L p_1$
$\|$	αq_2	βq_1	$R q_1$	$L q_1$	L	$\Lambda L p_0$	R
α	L	R	$\|L$	ΛR			
β	L	R	ΛL	$\|R$			

Zuerst werden die beiden Sätze von Strichen bearbeitet, bis der größte gemeinsame Teiler der beiden Zahlen auf dem Bande steht. Jedoch tritt dann nicht das Stopzeichen "!" (siehe z.B. die Konfiguration XII der Abb. 18), sondern der Zustand p_2 ein, und der Prozess wird fortgesetzt. Im weiteren Arbeitsablauf wird der erarbeitete Satz von Strichen in eine Dezimalzahl

umgeschrieben.

Selbstverständlich kann man auf diese Weise Kompositionen mit endlich vielen Algorithmen aufstellen.

Hieraus ergibt sich insbesondere, dass man bei der Realisierung numerischer Algorithmen annehmen kann, dass die natürlichen Zahlen in Form von Strichsätzen gegeben sind; denn hat man ein entsprechendes Schema $\mathfrak{A}$ unter diesen Voraussetzungen aufgestellt, so kann man es zu einem Schema $\mathfrak{B}$ erweitern, in welchem die Zahlen in dezimaler Schreibweise vorliegen.
Dazu hat man nach dem oben angegebenen Verfahren das Schema $\mathfrak{B}$ durch Komposition von drei Algorithmen zu konstruieren: dem Algorithmus zum Umschreiben aus dem Dezimalsystem, dem Algorithmus $\mathfrak{A}$ und dem Algorithmus zum Umschreiben in das Dezimalsystem.[20]

Ein anderes Verfahren, um Algorithmen miteinander zu koppeln, ist die mehrfache Anwendung des gleichen Algorithmus, bis eine gewisse vorgegebene Bedingung erfüllt ist. So führt beispielsweise der Algorithmus für das Umschreiben ins Dezimalsystem auf die wiederholte Anwendung des Algorithmus, der den Übergang von n zu $n+1$ beschreibt, bis alle Striche gelöscht sind.

Aus dem Funktionsschema eines gegebenen Algorithmus (wenn es einen solchen gibt) und entsprechenden Voraussetzungen kann man ein Schema für einen iterierten Algorithmus konstruieren. Diese Methode ist jedoch wesentlich komplizierter als die Komposition, und wir werden hier nicht weiter darauf eingehen.

Aus den analysierten Beispielen ist auch genügend klar zu ersehen, wie Funktionsschemata für andere, insbesondere nichtnumerische Algorithmen zu bilden sind. Wir wollen einen allgemeinen Plan angeben, nach dem ein Funktionsschema für einen Algorithmus zur Transformation von Wörtern aufgestellt werden kann (siehe Beispiel 4 des § 3).
Zuerst bildet man die Schemata $\mathfrak{A}_1$, $\mathfrak{A}_2$, $\mathfrak{A}_3$, $\mathfrak{A}_4$, die die orientierten Substitutionen

$$b \to acc, \quad ca \to accc, \quad aa \to \Lambda, \quad cccc \to \Lambda$$

realisieren. So formt beispielsweise die Maschine mit dem Schema $\mathfrak{A}_4$ das auf dem Band stehende Wort des Alphabets $\{a, b, c\}$ in ein äquivalentes Wort um, indem sie beim am weitesten links stehenden Buchstaben beginnend ein Quadrupel c auslöscht.
Enthält das Wort kein Quadrupel c, so bleibt es unverändert.

Weiter werden Schemata $\widetilde{\mathfrak{A}}_1$, $\widetilde{\mathfrak{A}}_2$, $\widetilde{\mathfrak{A}}_3$, $\widetilde{\mathfrak{A}}_4$ konstruiert, die die Algorithmen $\mathfrak{A}_1$, $\mathfrak{A}_2$, $\mathfrak{A}_3$, $\mathfrak{A}_4$ wiederholt anwenden.
So streicht beispielsweise die Maschine mit dem Schema $\widetilde{\mathfrak{A}}_4$ erst ein Quadrupel c, danach das nächste usw. bis das entstandene Wort kein Quadrupel aufeinanderfolgender c mehr enthält.
Das gesuchte Schema für den Reduktionsalgorithmus wird schließlich eine Komposition der Schemata $\widetilde{\mathfrak{A}}_1$, $\widetilde{\mathfrak{A}}_2$, $\widetilde{\mathfrak{A}}_3$, $\widetilde{\mathfrak{A}}_4$.

9 Die Grundhypothese der Theorie der Algorithmen

Die Analyse dieser Beispiele erweckt den Eindruck, als ob der Prozess in einer Turing-Maschine eine Zeitlupenaufnahme des Rechenprozesses ist, der von einem Menschen nach einem gewissen

[20] Hier drängt sich ein Vergleich mit den programmgesteuerten Rechenautomaten auf, die im Dualsystem arbeiten; in ihnen gibt es einen Eingabe-Umwandler vom Dezimal- in das Dualsystem und einen Ausgabe-Umwandler vom Dual- in das Dezimalsystem.

Algorithmus ausgeführt wird.
Gleichzeitig wird, ebenfalls auf Grund dieser Beispiele, verständlich, dass es einen Sinn hat, mittels Funktionsschemata von Turing-Maschinen andere uns bekannte Algorithmen wiederzugeben, die gewöhnlich in anderer Form vorliegen. Diese können durch Beschreibungen in Worten oder aber als spezielle Formeln gegeben sein.

In unserem augenblicklichen Untersuchungsstadium sieht es tatsächlich so aus, als ob das auch in allen Fällen gelingt. Ist das nun aber wirklich so?
Wie allgemein ist der Begriff der Turing-Maschine und des Turingschen Funktionsschemas? Kann man behaupten, dass die Vorgabe von Algorithmen mittels Funktionsschemata so universal ist, dass jeder Algorithmus auf diese Weise vorgegeben werden kann?

Diese Fragen beantwortet die Theorie der Algorithmen in Form folgender Hypothese:

Fundamentalhypothese der Theorie der Algorithmen.
Jeder Algorithmus kann mittels eines Turingschen Funktionsschemas vorgegeben und in einer entsprechenden Turing-Maschine realisiert werden.

Wir wollen hier zwei Fragen näher betrachten, die sich aus der Formulierung der Hypothese ergeben:

1. Worin besteht die Bedeutung dieser Hypothese für die Theorie der Algorithmen?
2. Worauf gründet sich diese Hypothese?

Die obige Formulierung der Hypothese enthält eine charakteristische Besonderheit, auf die wir ausdrücklich aufmerksam machen wollen. In dieser Formulierung ist einerseits die Rede von jedem Algorithmus, d.h. von dem allgemeinen Begriff des Algorithmus, der, wie schon mehrfach hervorgehoben, kein exakter mathematischer Begriff ist; andererseits wird von dem exakten mathematischen Begriff "Turingsches Funktionsschema" gesprochen.

Der Wert dieser Hypothese besteht nun darin, dass sie den allgemeineren aber verschwommenen Begriff "jeder Algorithmus" durch den spezielleren aber modernen exakten mathematischen Begriff "Turingsches Funktionsschema" (und seine Realisierung in einer Turing-Maschine) präzisiert.
Auf diese Weise erfasst die Theorie der Algorithmen alle möglichen Turingschen Funktionsschemata (Turing-Maschinen) als Untersuchungsgegenstände.
Gleichzeitig erhalten viele Fragestellungen einen vernünftigen Sinn, wie z.B. die Frage, ob es für irgendeinen Aufgabentypus einen Algorithmus gibt oder nicht. Jetzt ist diese Frage wie folgt zu verstehen:

Existiert eine Turing-Maschine (ein Funktionsschema), die die gewünschten Eigenschaften besitzt, oder existiert keine?

Unsere Hypothese rechtfertigt die Verwendung der grundlegenden Definition der modernen Theorie der Algorithmen, nach welcher der verschwommene Begriff des Algorithmus mit dem exakten Begriff des Funktionsschemas einer Turing-Maschine identifiziert wird.

Worauf beruht nun diese fundamentale Hypothese?

Zunächst wollen wir ausdrücklich darauf hinweisen, dass es sich nicht darum handeln kann, diese Hypothese so zu beweisen, wie man gewöhnlich in der Mathematik Sätze beweist. Der Wortlaut der Hypothese hat nämlich nicht den Charakter eines Satzes, weil er eine Behauptung bezüglich des allgemeinen Begriffs des Algorithmus enthält, der kein exakter mathematischer

Begriff ist, folglich auch kein Objekt strenger mathematischer Überlegungen sein kann.

Unsere Überzeugung von der Richtigkeit dieser Hypothese beruht hauptsächlich auf der Erfahrung. Alle bekannten Algorithmen, die im Verlaufe der vieltausendjährigen Geschichte der Mathematik erdacht wurden, lassen sich mittels Turingscher Funktionsschemata beschreiben. Natürlich bezieht sich der Inhalt dieser Hypothese nicht nur auf die Vergangenheit und stellt nicht nur fest, dass man alle bekannten Algorithmen durch Funktionsschemata vorgehen kann, sondern er macht auch eine ganz bestimmte Aussage für die Zukunft:

Wenn eine Vorschrift als Algorithmus erkannt ist, dann kann sie unabhängig davon, in welcher Form und mit welchen Mitteln sie formuliert ist, mittels eines Funktionsschemas einer Turing-Maschine wiedergegeben werden.

In diesem Sinne kann man die Fundamentalhypothese mit einem physikalischen Gesetz vergleichen, z.B. mit dem Gesetz von der Erhaltung der Energie, auf Grund dessen ebenfalls Aussagen über die Zukunft gemacht werden. Die vielfältigen Erfahrungen, die in der Vergangenheit gesammelt wurden, lassen solche Prognosen als hinreichend begründet erscheinen.

Es gibt aber auch andere Überlegungen, die uns die Überzeugung von der Richtigkeit der Fundamentalhypothese vermitteln.

Im vorhergehenden Paragraphen waren zwei Beispiele angegeben worden, wie man aus bekannten Algorithmen kompliziertere aufbauen kann: durch Komposition und Iteration.
Man kann noch viele solcher Verfahren angeben. Jedoch sind alle bekannten Verfahren und alle diejenigen, die man entsprechend dem Entwicklungsstand der Wissenschaften voraussehen kann, so beschaffen, dass man kompliziertere Algorithmen stets wieder durch Funktionsschemata darstellen kann, wenn die Teil-Algorithmen mittels Funktionsschemata vorgebbar sind.

Insbesondere war ja gezeigt worden, wie man aus vorgegebenen Funktionsschemata (Algorithmen) durch Komposition neue aufbauen kann.
Wir erinnern außerdem noch an eine Tatsache, die schon im § 6 erwähnt wurde. Als in der Wissenschaft das dringende Bedürfnis nach einem exakten Begriff des Algorithmus entstand, bemühten sich viele Mathematiker, eine gewisse Standardform zur Vorgabe von Algorithmen zu finden, die hinreichend exakt sein sollte, um als Objekt mathematischer Untersuchungen dienen zu können, gleichzeitig aber auch hinreichend allgemein, um alle denkbaren Algorithmen zu erfassen.

Neben den Funktionsschemata von Turing-Maschinen wurden andere Methoden zur Präzisierung dieses Begriffs angegeben. So kamen beispielsweise A.A. Markow zu dem Begriff des normalen Algorithmus (der in diesem Buche nur kurz am Reduktionsalgorithmus für die Halbgruppe des Beispiels 4 im § 3 erläutert wurde) und Gödel und Kleene zum Begriff des rekursiven Algorithmus (der rekursiven Funktion) usw.
Alle diese Präzisierungen erwiesen sich jedoch letzten Endes als äquivalent. Das ist bestimmt kein Zufall. Es ist ein zusätzliches Argument für die oben formulierte Hypothese.

Wir wollen zum Abschluss noch bemerken, dass die Fundamentalhypothese innerhalb der Theorie der Algorithmen gar nicht angewandt wird, d.h., beim Beweis irgendwelcher Sätze in dieser Theorie wird von der Hypothese kein Gebrauch gemacht. Wenn also jemand die Hypothese nicht kennt, oder aber unsere Argumente nicht anerkennt, wird er in der modernen Theorie der Algorithmen nicht auf formale Schwierigkeiten stoßen.

Für diese Personen wird das, was wir die Theorie der Algorithmen nennen, nur eine Theorie

der Funktionsschemata von Turing-Maschinen, d.h. im wesentlichen nur eine Theorie gewisser Algorithmen spezieller Gestalt sein.

Der Autor dieser Zeilen teilt die Überzeugung von der Richtigkeit der Fundamentalhypothese und der sich daraus ergebenden Einschätzung der modernen Theorie der Algorithmen als einer allgemeinen Theorie, die die Natur der Dinge selbst beschreibt; er glaubt nicht, dass es sich dabei nur um eine Theorie einer künstlich eingeschränkten Klasse spezieller "Turingscher Algorithmen" handelt.

10 Die universelle Turing-Maschine

Bis jetzt nahmen wir den Standpunkt ein, dass verschiedene Algorithmen von verschiedenen Turing-Maschinen realisiert werden, die sich durch ihre Funktionsschemata voneinander unterscheiden. Man kann jedoch eine universelle Turing-Maschine konstruieren, die in gewissem Sinne jeden Algorithmus verwirklichen und damit die Arbeit jeder speziellen Turing-Maschine übernehmen kann.

Damit wir uns besser vorstellen können, wie das vor sich gehen soll, machen wir folgendes Experiment.
Auf das Band der Maschine sei eine Anfangsinformation $\mathfrak{A}$ geschrieben. Außerdem sei vorausgesetzt, dass ein Mensch angeben soll, wie diese Information von der Maschine behandelt wird und wie das Endresultat aussieht.
Wenn dieser Mensch mit den Arbeitsprinzipien von Turing-Maschinen vertraut ist, braucht man ihm außer der Anfangsinformation $\mathfrak{A}$ nur noch das entsprechende Funktionsschema der Maschine mitzuteilen. Dann kann er, indem er die Arbeit der Maschine nachahmt, zu demselben Ergebnis kommen wie die Maschine. Dabei hat er die nötigen Konfigurationen zu notieren, wie wir es bei der Analyse des Euklidischen Algorithmus machten.

Das bedeutet aber gerade, dass dieser Mensch die Arbeit einer Turing-Maschine ausführen kann, wenn er nur ihr Funktionsschema kennt. Der Prozess der Nachahmung der Maschine kann aber gemäß ihrem Funktionsschema ganz genau beschrieben werden.
Diese Beschreibung kann man aber einem Menschen mit- teilen, auch wenn er nicht die geringste Ahnung davon hat, was eine Turing-Maschine ist.

Eine solche Beschreibung soll als Nachahmungsalgorithmus bezeichnet werden. Dieser enthält außer dem Funktionsschema einer gewissen Turing-Maschine eine Anfangskonfiguration, die auf dem Bande gespeichert ist. Ein Mensch, der über eine solche eingehende Beschreibung verfügt, kann ohne weiteres die Arbeit der entsprechenden Maschine nachahmen und am Schluss das gleiche Resultat liefern.
Einen solchen Nachahmungsalgorithmus könnte man auch in Form eines Systems von Anweisungen beschreiben:

Anweisung 1. Rufe die (einzige) Zelle auf dem Bande auf, unter der ein Buchstabe geschrieben steht.

Anweisung 2. Ermittle in der Tabelle[21] die Spalte, die mit dem unter der aufgerufenen Zelle stehenden Buchstaben bezeichnet ist.

Anweisung 3. Gehe in der Zeile, die mit dem in der Zelle stehenden Buchstaben bezeichnet

[21] D.h. im Funktionsschema.

ist, bis zur oben ermittelten Spalte und notiere das dort stehende Tripel.

Anweisung 4. Ersetze den Buchstaben aus der aufgerufenen Zelle durch den ersten Buchstaben des notierten Tripels.

Anweisung 5. Wenn in dem notierten Tripel der zweite Buchstabe ein "!" ist, so halte an. Der Prozess ist beendet.

Anweisung 6. Wenn in dem notierten Tripel der zweite Buchstabe ein M ist, so ersetze den Buchstaben, der unter der aufgerufenen Zelle steht, durch den dritten Buchstaben des notierten Tripels.

Anweisung 7. Wenn in dem notierten Tripel der zweite Buchstabe ein L ist, so lösche den unter der aufgerufenen Zelle stehenden Buchstaben und setze unter die links benachbarte Zelle den dritten Buchstaben des notierten Tripels.

Anweisung 8. Wenn in dem notierten Tripel der zweite Buchstabe ein R ist, so lösche den unter der aufgerufenen Zelle stehenden Buchstaben und setze unter die rechts benachbarte Zelle den dritten Buchstaben des notierten Tripels.

Anweisung 9. Gehe über zur Anweisung 1.

Man kann aber anstelle eines Menschen, der nach diesem Algorithmus arbeitet, eine Turing-Maschine setzen. Das ist dann die universelle Turing-Maschine, die die Arbeit jeder anderen Turing-Maschine nachahmen kann.
Mit anderen Worten, der Nachahmungsalgorithmus, den wir oben durch ein System von 9 Anweisungen festgelegt haben, kann in passender Weise in Form eines gewissen Turingschen Funktionsschemas (universellen Schemas) vorgegeben werden.

Der vollständige und strenge Beweis dieser Tatsache, die eine weitere Bestätigung der Fundamentalhypothese der Theorie der Algorithmen ist, ist in seinen Einzelheiten viel zu kompliziert, als dass wir ihn in diesem kleinen Bändchen bringen könnten. Wir beschränken uns auf einige allgemeine Bemerkungen, die wohl zum Verständnis des Wesens der Sache ausreichen dürften.

Wir weisen zuerst darauf hin, dass in dem von uns beschriebenen Nachahmungsalgorithmus als Ausgangsdaten (Ausgangsinformation) das Funktionsschema der nachzuahmenden Maschine und die entsprechende Anfangsinformation auftreten. Diese Ausgangsinformation wird durch den Algorithmus zu einer Abschlusskonfiguration verarbeitet, die das Resultat herstellt, welches durch die nachzuahmende Maschine geliefert wird.
Die universelle Maschine muss dasselbe machen. Es müssen hier jedoch noch zwei Umstände berücksichtigt werden.

1. Die unmittelbare Eingabe des Funktionsschemas der nachzuahmenden Maschine und der entsprechenden Konfiguration auf das Band der universellen Maschine als Ausgangsinformation ist unmöglich.
In der universellen Maschine setzt sich nämlich, wie in jeder Turing-Maschine, eine Information aus Buchstaben zusammen, die auf einem eindimensionalen Band gespeichert sind, d.h. in einer Zeile, wobei diese Information aus einem oder mehreren Wörtern des äußeren Alphabets der Maschine besteht.
Wir gaben jedoch die Funktionsschemata stets durch "zweidimensionale" Tabellen vor, in denen die Buchstaben in mehreren Zeilen angeordnet waren. Ähnlich verhält es sich mit den Konfigurationen, in denen die Zeichen für die Zustände unter die Buchstaben des äußeren Alphabets (unter das Band) geschrieben wurden.

2. Die universelle Maschine (wie überhaupt jede Turing-Maschine) kann nur ein festes endliches äußeres Alphabet besitzen.
Trotzdem muss sie aber auf alle möglichen Schemata und Konfigurationen anwendbar sein, wobei Buchstaben der verschiedensten Alphabete mit beliebiger Anzahl verschiedener Buchstaben auftreten können.

Daher muss in erster Linie dafür gesorgt werden, dass die Funktionsschemata und die Konfigurationen in entsprechender Weise umgearbeitet werden. Sie müssen so vorgegeben werden, dass die oben angegebenen Besonderheiten einzelner Turing-Maschinen nicht stören.
Die Eindimensionalität der Information und die Endlichkeit des Alphabets müssen stets gewahrt bleiben. Der Beschreibung eines solchen Verfahrens wollen wir uns jetzt zuwenden.

1. Statt das Schema als zweidimensionale Tabelle mit k Zeilen und m Spalten zu schreiben, setzen wir es in $m \cdot k$ Quintupeln von Buchstaben an, derart, dass das erste Symbol eines Quintupels die Spalte der Tabelle, das zweite die Zeile und die restlichen drei jenes Tripel bezeichnen, welches in der Tabelle im Schnittpunkt der angegebenen Zeile mit der angegebenen Spalte zu finden ist.

So wäre beispielsweise statt des Schemas der Abb. 6 die folgende Zeile zu schreiben:

$$q_1 \mid \alpha M q_2 q_2 \mid \beta M q_1 q_3 \mid\mid R q_1 ... \qquad (\Omega)$$

Offenbar kann man aus dieser Zeile auf Wunsch wieder die ursprüngliche Tabelle rekonstruieren. Analog lässt sich bei den Konfigurationen verfahren. Wir setzen die Zustandszeichen nicht unter die Buchstaben der aufzurufenden Zelle, sondern unmittelbar links neben sie. Die Konfiguration IV der Abb. 18 hat dann das folgende Aussehen:

$$||||q_4||$$

Offenbar kann man auch aus dieser Zeile die Konfiguration auf eindeutige Weise zurückgewinnen.

2. Zur Charakterisierung der Funktionsschemata und der Konfigurationen sind keineswegs spezielle Zeichen für die Buchstaben des äußeren und des inneren Alphabets (der Zustände) wesentlich. Wenn beispielsweise in Abb. 6 der Buchstabe β überall durch den Buchstaben b ersetzt wird, so ändert das gar nichts an unseren Betrachtungen. Wichtig ist nur, dass verschiedene Objekte durch verschiedene Symbole wiedergegeben werden und dass man die Buchstaben des inneren Alphabets von denen des äußeren unterscheiden kann.

Man kann natürlich auch statt L, R und M andere Zeichen für die Verschiebungen (nach links, nach rechts und keine Verschiebung) wählen, doch muss hier klar festgelegt werden, welcher Buchstabe welche Verschiebung bedeutet. Hier muss die Tatsache zum Ausdruck kommen, dass jeder der drei Buchstaben eine wohlbestimmte Operation bezeichnet, die durch keine andere ersetzt werden kann.

Unter Berücksichtigung dieser Umstände ersetzen wir in der Zeile Ω jeden einzelnen Buchstaben durch eine Folge von Einsen und Nullen (Kodegruppe) derart, dass verschiedene Buchstaben durch verschiedene Kodegruppen ersetzt werden. Selbstverständlich sollen gleiche Buchstaben immer durch die gleiche Kodegruppe ersetzt werden.
Damit geht die Zeile Ω in eine Zeile Ω' über. Damit man aber Ω aus Ω' wieder zurückgewinnen kann, muss das Kodierungsverfahren (Verschlüsselung der Buchstaben durch Kodegruppen) folgende Bedingungen erfüllen:

1. Man muss die Zeile Ω' eindeutig in einzelne Kodegruppen zerlegen können.
2. Man muss erkennen können, welche Kodegruppe man jedem einzelnen der Buchstaben L, R, M zuschreiben muss. Ferner müssen die Kodegruppen für die Buchstaben des äußeren Alphabets sich deutlich von den Kodegruppen für die Zustandszeichen unterscheiden.

Das folgende Kodierverfahren erfüllt diese beiden Bedingungen sicher.

1. Als Kodegruppe nimmt man $3 + k + m$ verschiedene Wörter der Form

$$100...01$$

(zwischen den Einsen stehen lauter Nullen).
Dann kann die Zeile Ω' leicht auf eindeutige Weise in Kodegruppen zerlegt werden, die sich durch die Anzahl der aufeinanderfolgenden Nullen voneinander unterscheiden, welche zwischen zwei Einsen stehen (wo 11 steht, endet und beginnt je eine Kodegruppe).

2. Die Buchstaben werden gemäß folgender Kodierungstabelle verschlüsselt:

	Buchstabe	Kodegruppe		
	L	101		
	M	1001		
	R	10001		
äußeres	s_1	100001	4 Nullen	gerade
Alphabet	s_2	10000001	6 Nullen	Anzahl
	...	...	...	Nullen,
	s_k	100...001	2(k+1) Nullen	mindestens 4
inneres	q_1	1000001	5 Nullen	ungerade
(Zustands-)	q_2	100000001	7 Nullen	Anzahl
Alphabet	...	...	...	Nullen,
	q_m	1000...001	2(m+1)+1 Nullen	mindestens 5

In diesem Kodierungssystem ergibt sich in unserem Fall für Ω':

$$10000011000011000000110011000000011000000011000011000000000110011000001...$$

Eine Zeile aus Einsen und Nullen für das Funktionsschema oder für die Konfiguration nennen wir Chiffre des Funktionsschemas bzw. Chiffre der Konfiguration.
Nach einer vorgegebenen Chiffre ist das Schema bzw. die Konfiguration in der ursprünglichen Form leicht reproduzierbar. Daher kann man die Vorgabe eines Schemas oder einer Konfiguration stets durch Chiffren bewerkstelligen. Anstelle der Einsen und Nullen kann man selbstverständlich auch beliebige andere Zeichen nehmen, z.B. a, b.

Jetzt ist es nicht schwer, die Formulierung der Anweisungen 1-9 so abzuändern, dass man einen Nachahmungsalgorithmus erhält, der die Chiffre der nachzuahmenden Maschine verarbeitet und die Chiffre der Anfangskonfiguration in die Chiffre der Resultatkonfiguration umformt. Wir beschränken uns auf einige Illustrationen:

Anweisung 1. Rufe in der Konfigurations-Chiffre die (eindeutig bestimmte) Kodegruppe auf, die unmittelbar rechts neben der Kodegruppe mit ungerader Anzahl Nullen steht.

Anweisung 2 und 3. Suche in der Chiffre des Funktionsschemas das Paar benachbarter Kodegruppen, welches mit dem Kodegruppenpaar in der Konfigurations-Chiffre übereinstimmt, in dem die zweite Kodegruppe die aufgerufene ist.

Anweisung 6. Wenn in dem aufgerufenen Kodegruppentripel der Funktionsschema-Chiffre die zweite Gruppe die Gruppe 1001 ist, so ersetze in der Konfigurations-Chiffre die Kodegruppe mit ungerader Anzahl Nullen durch die dritte Gruppe des aufgerufenen Tripels.

Bei einer weitergehenden Untersuchung dieses Algorithmus gelangt man dahin, dass sich jede Operation über Kodegruppen auf eine Kette von Standardoperationen zurückführen lässt, die in der Turing-Maschine ausführbar sind (Ersetzen eines Zeichens durch ein anderes, Verschiebung um einen Schritt usw.).
Dabei treten außer den Zeichen 0 und 1, aus denen die Chiffren bestehen, noch andere Buchstaben auf, z.B. ein Buchstabe, der eine Chiffre von einer anderen trennt, Buchstaben, die als provisorische Kennzeichen bei der Durchmusterung der Einsen und Nullen gebraucht werden (vergleiche mit dem Euklidischen Algorithmus) und andere.

Als Ergebnis einer solchen Aufgliederung des Nachahmungsalgorithmus ergibt sich schließlich die Beschreibung eines gewissen Turingschen Funktionsschemas. Das ist dann gerade das Schema der universellen Maschine. Wenn irgendeine Maschine A eine gewisse Aufgabe löst, so kann auch die universelle Maschine diese Aufgabe lösen. Es ist dazu nur notwendig, dass außer der Anfangsdaten-Chiffre auch die Chiffre der Maschine A auf ihrem Bande gespeichert wird.

Im Zusammenhang mit der Existenz der universellen Turing-Maschine kann man alle möglichen Funktionsschemata (oder ihre Chiffren) auf zweifache Weise deuten:

1. Das Schema beschreibt den logischen Block einer speziellen Turing-Maschine, die den entsprechenden Algorithmus realisiert (das ist der Standpunkt, den wir ursprünglich einnehmen).

2. Das Schema beschreibt ein Programm, welches auf das Band der Turing-Maschine aufgetragen ist, um den entsprechenden Algorithmus zu realisieren.

Zum Abschluss wollen wir noch darauf hinweisen, dass die modernen programmgesteuerten Rechenautomaten gerade wie die universelle Turing-Maschine aufgebaut sind. In dem Speicher der Automaten befindet sich neben den Anfangswerten der gestellten Aufgabe auch das Programm zu ihrer Lösung.

Die Unterteilung des Speichers in einen inneren und einen äußeren ist auch für die Rechenautomaten charakteristisch. Der äußere Speicher ist meist ein Magnettrommelspeicher oder ein Tonband. In diesem Speicher wird genau wie beim gewöhnlichen Magnettongerät die Information durch Magnetisierung aufbewahrt.
Zum Unterschied von der Turing-Maschine, in der der äußere Speicher (das Band) unendlich ist, ist jedoch der äußere Speicher in jedem realen Rechenautomaten endlich (Magnettonband oder Trommel).

Selbstverständlich kann man diesen prinzipiellen Unterschied zwischen einem realen Rechenautomaten und der Turing-Maschine, die ja eine abstrakte idealisierte Maschine ist, nicht beseitigen.
Außerdem sei darauf hingewiesen, dass man den äußeren Speicher eines realen Rechenautomaten unbegrenzt vergrößern kann, ohne die Konstruktion der Maschine zu verändern. Man braucht nur an das schon vorhandene Tonbandstück neue ergänzende Stücke "anzukleben".

11 Algorithmisch unlösbare Probleme

Der Übergang vom verschwommenen Begriff des Algorithmus zum exakten Begriff der Turing-Maschine, die durch ihre Chiffre vorgegeben sein kann, gestattet auch, die Frage nach der algorithmischen (oder maschinellen) Unlösbarkeit irgendeiner Aufgabenschar zu präzisieren. Die Fragestellung lautet jetzt:

Gibt es eine TURING-Maschine, die eine vorgegebene Schar von Aufgaben löst, oder gibt es diese Maschine nicht? (Was "eine Turing-Maschine löst eine gewisse Schar von Aufgaben" bedeutet, wurde in § 7 erwähnt.)

Auf diese Frage gibt die Theorie der Algorithmen in einigen Fällen eine verneinende Antwort. Eines der ersten Resultate in dieser Beziehung wurde im Jahre 1936 von dem amerikanischen Mathematiker A. Church angegeben.[22] Es bezieht sich auf das Entscheidungsproblem der mathematischen Logik (siehe § 6).

Satz von Church. Das Entscheidungsproblem ist algorithmisch unlösbar.

Das erklärt nicht nur, warum alle vorangegangenen Versuche, einen Algorithmus aufzustellen, erfolglos waren, sondern auch die Vergeblichkeit aller dieser Versuche.

Dieser Unmöglichkeitsbeweis, der in der Theorie der Algorithmen geführt wurde, weist dieselbe mathematische Strenge wie ein in anderen Gebieten der Mathematik geführter Unmöglichkeitsbeweis auf. (z.B., dass die Winkeldreiteilung mit Hilfe von Zirkel und Lineal unmöglich ist oder dass es unmöglich ist, für die Seite und die Diagonale eines Quadrates ein gemeinsames Maß anzugeben.)
Wir skizzieren einen solchen Beweis für das Selbstanwendbarkeitsproblem.

Auf das Band einer Turing-Maschine sei ihre eigene Chiffre (d.h. die Chiffre des Funktionsschemas der Maschine) geschrieben. Es sind zwei Fälle möglich:

1. Die Maschine ist auf ihre Chiffre anwendbar, d.h., sie verarbeitet diese Chiffre und bleibt nach endlich vielen Takten stehen, weil das Stopsignal gekommen ist.

2. Die Maschine ist nicht auf ihre Chiffre anwendbar, d.h., das Stopsignal tritt niemals auf.
In diesem Zusammenhang teilt man die Maschinen (Chiffren) in zwei Klassen ein: Die Klasse der auf sich selbst anwendbaren und die Klasse der nicht auf sich selbst anwendbaren Maschinen (Chiffren).

Es ergibt sich das folgende Selbstanwendbarkeitsproblem. Von einer beliebig vorgegebenen Chiffre ist auszusagen, zu welcher Klasse die entsprechende Maschine gehört, zur Klasse der auf sich selbst anwendbaren oder zu der Klasse der nicht auf sich selbst anwendbaren.

Das ist eine typische Aufgabe zur Konstruktion eines Algorithmus, weil man zu ihrer Lösung einer allgemeinen Methode (Algorithmus oder Maschine) bedarf, die automatisch von jeder beliebigen Chiffre entscheidet, ob sie selbstanwendbar ist oder nicht.

Theorem. Das Selbstanwendbarkeitsproblem ist algorithmisch unlösbar.

Beweis (indirekt). Wir wollen annehmen, eine solche Maschine A existiere; dann würde in A jede selbstanwendbare Chiffre in ein Symbol σ (welches die bejahende Antwort auf die Frage nach der Selbstanwendbarkeit ausdrückt) und jede nichtselbstanwendbare in ein anderes Sym-

[22] A. Church, A note on the Entscheidungsproblem, J. symbolic Logic 1, 40-41 (1936).

bol τ (was eine negative Antwort auf die gestellte Frage bedeuten soll) umgeformt.

In diesem Fall könnte man auch eine Maschine B konstruieren, die ebenfalls die nichtselbstanwendbaren Chiffren in τ umformte, aber auf die selbstanwendbaren Chiffren nicht anwendbar ist. Das könnte dadurch erreicht werden, dass nach dem Auftreten des Symbols σ nicht das Stopsignal zur Auswirkung kommt, sondern das Symbol σ unbeschränkt oft wiederholt wird.

Somit wäre die Maschine B auf jede nichtselbstanwendbare Chiffre anwendbar (es wird das Symbol τ erzeugt), aber auf jede selbstanwendbare Chiffre nichtanwendbar. Das führt jedoch zu einem Widerspruch.

In der Tat:
1. diese Maschine B sei selbstanwendbar, dann ist sie auf ihre Chiffre B anwendbar und formt sie in das Symbol τ um. Das bedeutet aber gerade, dass B nicht auf sich selbst anwendbar ist.
2. Sei B nicht auf sich selbst anwendbar; dann muss B selbstanwendbar sein. Dieser Widerspruch beweist den Satz.

Die ersten Resultate über algorithmische Unlösbarkeit wurden für Probleme aus der mathematischen Logik (Entscheidungsproblem) und der Theorie der Algorithmen (z.B. das Selbstanwendbarkeitsproblem) erzielt. Später zeigte sich, dass auch Probleme aus scheinbar viel weniger umfassenden Problemkreisen der verschiedensten Gebiete der Mathematik algorithmisch unlösbar sind.

In erster Linie muss man hier auf eine Reihe algebraischer Probleme hinweisen, die von sowjetischen Mathematikern untersucht worden sind und die zu den verschiedenen Varianten des Wortproblems gehören.

Das Äquivalenzproblem für Halbgruppen (siehe § 3) wurde schon 1914 von dem norwegischen Mathematiker A. Thue formuliert.
Er behandelte Algorithmen zur Lösung des Äquivalenzproblems in gewissen speziellen Halbgruppen. Man hat bis heute viele Versuche unternommen, um einen allgemeinen Algorithmus zu konstruieren, der dieses Problem in beliebigen Halbgruppen und für beliebige Wortpaare entscheiden könnte (ob Äquivalenz vorliegt oder nicht).

In den Jahren 1946 und 1947 konstruierten der sowjetische Mathematiker Andre Andrejewitsch Markow und der amerikanische Mathematiker Emil Post POST unabhängig voneinander konkrete Beispiele von Halbgruppen, in denen das Wortproblem algorithmisch unlösbar ist. Also kann es auch keinen Algorithmus für beliebige Halbgruppen geben.

Auf Grund dieses Ergebnisses stellten A.A. Markow und seine Schüler ganze Klassen von Halbgruppen auf, für welche die Konstruktion eines Algorithmus unmöglich ist.

Einen großen Eindruck machte auf die Mathematiker der Welt das Ergebnis von Pjotr Sergejewitsch Nowikow über die algorithmische Unlösbarkeit des Identitätsproblems (Wortproblems) in der Gruppentheorie. Die Arbeit erschien im Jahre 1955[23].
Formal ist dieses Problem ein Spezialfall des Äquivalenzproblems in Halbgruppen.[24]

[23] Für diese Arbeit erhielt P.S. Nowikow 1957 den Leninpreis.

[24] D.h., wenn es einen Algorithmus gäbe, mit dessen Hilfe sich die Identität von Wörtern in Halbgruppen ermitteln ließe, so müsste dieser auch das Identitätsproblem in Gruppen lösen. Aus der algorithmischen Unlösbarkeit dieses Problems in Halbgruppen folgt jedoch nicht, dass das Identitätsproblem in der Gruppentheorie unlösbar ist.

Es werden nämlich nur solche Halbgruppen betrachtet, in denen zu jedem Buchstaben a des Alphabets eine zulässige Substitution der Form

$$a\alpha - \Lambda$$

existiert, wobei α irgendein Buchstabe desselben Alphabets ist (der mit a übereinstimmen kann). Dann ist die Halbgruppe eine Gruppe.

Der inhaltliche Sinn dieser Forderung wird verständlich, wenn man die Wörter einer Halbgruppe (analog dem Beispiel 4 aus § 3) als Transformationen auffasst, die durch Multiplikation elementarer Transformationen entstanden sind. Jedem Buchstaben eines Wortes entspricht eine elementare Transformation.
Das leere Wort ist die identische Transformation (die alles unverändert lässt; vgl. mit § 3).

Die zulässige Substitution $a\alpha - \Lambda$ bedeutet dann, dass es zu jeder elementaren Transformation (die durch den Buchstaben a gegeben ist) eine elementare Transformation (die durch α beschrieben wird) gibt derart, dass ihre Hintereinanderausführung gerade die identische Transformation ergibt. Ohne uns auf weitere Einzelheiten einzulassen, wollen wir erwähnen, dass die Untersuchung solcher Transformationsgruppen von großem theoretischen und praktischen Interesse ist. Der Begriff der Gruppe selbst ist einer der Grundbegriffe der modernen Mathematik.

Wir müssen uns noch darüber klar werden, dass das äußerst wichtige Resultat von Markow und Post keine Aussage über das Wortproblem in der Gruppentheorie zulässt.
Da nämlich die speziellen Halbgruppen, für die A.A. Markow und E. Post die algorithmische Unlösbarkeit des Äquivalenzproblems bewiesen haben, keine Gruppen sind, d.h. die oben angegebene wesentliche Forderung nicht erfüllen, wird durch das Resultat von Markow und Post die Existenz eines Algorithmus für das Identitätsproblem der Gruppentheorie nicht ausgeschlossen.

Daraufhin wurde weiter nach einem solchen Algorithmus gesucht. Alle Hoffnungen mussten aufgegeben werden, als die Arbeit von P.S. Nowikow bekannt wurde.
Darin wurde bewiesen, dass es einen solchen Algorithmus nicht geben kann. Nowikow konstruierte ein Beispiel einer Halbgruppe, die die angegebene Forderung erfüllt (also ein Beispiel einer Gruppe), und bewies, dass in dieser Halbgruppe das Äquivalenzproblem algorithmisch unlösbar ist. Also kann es erst recht keinen Algorithmus geben, der für alle endlich erzeugbaren Gruppen brauchbar ist.

Die Beispiele, die von Markow und Nowikow zum Nachweis der algorithmischen Unlösbarkeit der untersuchten Probleme konstruiert wurden, waren sehr kompliziert und wiesen hunderte zulässiger Substitutionen auf.
Natürlich fragte man nach möglichst einfachen Beispielen. Diese Aufgabe wurde unlängst von dem jungen Leningrader Mathematiker G.S. Zeitin glänzend gelöst. Für die im § 3 angegebene von Zeitin konstruierte Halbgruppe, die nur sieben zulässige Substitutionen enthält, ist das Äquivalenzproblem ebenfalls algorithmisch unlösbar.

Im § 1 hatten wir das Hilbertsche Problem über diophantische Gleichungen formuliert. Ein halbes Jahrhundert lang führte man Untersuchungen durch, um den gewünschten Algorithmus zu konstruieren, die aber erfolglos waren.
In letzter Zeit wurden auch schon Untersuchungen in entgegengesetzter Richtung geführt, d.h., man versuchte die algorithmische Unlösbarkeit zu beweisen.

Obwohl noch kein abschließendes Resultat vorliegt, kann man angesichts der bisher vorliegen-

den Teilergebnisse als fast sicher annehmen, dass das zehnte Hilbertsche Problem algorithmisch unlösbar ist.

12 Schlussbemerkungen

Wir wollen zum Schluss noch einige allgemeine Bemerkungen machen.

1. Der Satz über die algorithmische Unlösbarkeit gewisser Scharen von Aufgaben darf nicht zum Agnostizismus verleiten.
Jeder solche Satz betrifft zwar eine ganze Klasse von Aufgaben, von denen nachgewiesen wird, dass sie nicht durch eine einzige effektive Methode (Algorithmus) lösbar sind.

Das bedeutet aber nicht, dass man unter den vielen einzelnen Aufgaben dieser Scharen nicht welche finden kann, die trotzdem lösbar sind. Auch der oben bewiesene Satz ist keinesfalls so zu verstehen, als ob es Chiffren gäbe, von denen man im Prinzip nicht feststellen kann, ob sie selbstanwendbar sind oder nicht.

Er besagt nur, dass der betrachtete Aufgabentyp so allgemein und umfassend ist, dass ein einziger Algorithmus zur Lösung aller Aufgaben dieses Typs nicht existiert.
In diesem Fall wird es das Ziel der mathematischen Untersuchungen sein, schrittweise immer allgemeinere Algorithmen zu konstruieren, die stets umfangreichem Unterklassen des gegebenen Aufgabentyps der automatischen Berechnung zugänglich machen.

2. Die Sätze über die algorithmische Unlösbarkeit zeigen, dass die Mathematik nicht auf die Konstruktion von Algorithmen zurückgeführt werden kann.
Der Erkenntnisprozess in der Mathematik lässt sich nicht bis zum Ende automatisieren.

Schon in einzelnen, relativ engen Gebieten der Mathematik (wie der Theorie der Gruppen mit endlich vielen Erzeugenden usw.) treten Probleme auf, die kein Automat (d.h. keine Turing-Maschine mit endlich vielen Zuständen und endlichem Speicher) lösen kann. Um so absurder ist die Behauptung, die schöpferische Arbeit des Wissenschaftlers könne vollständig durch Maschinen ersetzt werden.

3. Gleichzeitig ist festzustellen, dass das Anwendungsgebiet der algorithmischen Prozesse immer umfangreicher wird. Das bezieht sich nicht nur auf Rechenprozesse, die in der Mathematik auftreten.
Für viele Prozesse, die gewöhnlich als sehr schwierig und kompliziert angesehen werden, kann man theoretisch Algorithmen konstruieren, die ihrer Idee nach einfach sind.

Die praktischen Schwierigkeiten jedoch, die bei der Realisierung dieser Prozesse auftreten, hängen mit dem großen Umfang der angegebenen Algorithmen zusammen. Zu ihrer Ausführung sind außerordentlich viele Operationen (obwohl diese Operationen selbst einfach sind) notwendig. Diese Bemerkung bezieht sich insbesondere auf Spielprozesse (Schach usw.), bei denen sehr viele Varianten auftreten, von denen die optimale zu ermitteln ist.

Mit der Entwicklung der elektronischen Rechenautomaten hat sich jedoch die Anzahl der praktisch anwendbaren Algorithmen wesentlich vergrößert.

4. Schließlich richten wir unsere Aufmerksamkeit noch einmal darauf, dass jede reale Rechenmaschine nur als Näherungsmodell einer Turing-Maschine angesehen werden kann.
In realen Maschinen ist nämlich das Volumen des äußeren Speichers beschränkt, während im Schema der Turing-Maschine ein unendliches Band auftritt.

Die technische Verwirklichung eines unbeschränkten Speichers ist selbstverständlich unmöglich. Eine bedeutende Vergrößerung des Speichervolumens im Vergleich mit dem erreichten Niveau ist nicht nur wünschenswert, sondern auch ausführbar. Durch Vergrößerung des Speichervolumens und Erhöhung der Rechengeschwindigkeit sind weitere große Fortschritte in der Entwicklung der Rechenautomaten zu erwarten.