Zur Problematik der Rechenautomaten*

Von Konrad Zuse

1972

Einführung

Wenn man die Entwicklung der Rechenmaschine verfolgt, so erkennt man, daß die entscheidenden Impulse im allgemeinen nicht von Fachleuten der Rechenmaschinentechnik gegeben wurden, sondern von Außenseitern. Mathematiker und Ingenieure suchten nach Möglichkeiten, sich durch Ausgestaltung der Rechenmaschine neue Werkzeuge zu schaffen. Darüber hinaus hat aber die Rechenmaschinenentwicklung auch ihrerseits weitgehend die Gedanken der Mathematiker beeinflußt, so daß man heute von einer engen Verflechtung zwischen Mathematik und Rechenmaschinentechnik sprechen kann.

Ohne an dieser Stelle einen geschichtlichen Rückblick im einzelnen geben zu wollen, seien die Namen Schickard, Pascal und Leibniz erwähnt, die als Astronomen, Mathematiker und Philosophen die ersten Rechengeräte für arithmetische Operationen bauten. Auch der Urvater der programmgesteuerten Rechenmaschine, Charles Babbage, war Mathematiker und versuchte im vorigen Jahrhundert vergeblich, seine weit vorausschauenden Pläne mit den damals noch primitiven technischen Möglichkeiten zu verwirklichen. [XIII 1]

Auch die etwa mit dem Jahre 1935 beginnende Entwicklung des „Computers“, wie wir heute sagen, lag vornehmlich in den Händen von Außenseitern. Das trifft sowohl für die Entwicklung in den USA als auch für diejenige, welche vom Verfasser in Deutschland durchgeführt wurde, zu.

Bekannt sind auch die Arbeiten des Mathematikers und Logikers Turing, der das theoretische Modell einer einfachen Rechenmaschine benutzte, um mit Hilfe des Prinzips der „Berechenbarkeit“ mathematische Sätze logisch zu beweisen. [XIII 10]

*ZIA 0022. ZuP 045/001. Version 1, Abbildungen fehlen. Durchgesehen von R. Rojas, G. Wagner, L. Scharf

Die weitere Entwicklung lag und liegt ebenfalls weitgehend in den Händen von Mathematikern. Sie waren und sind nicht nur als Anwender interessiert, sondern beeinflussen auch weitgehend die Struktur und die theoretischen Grundlagen der Rechengeräteentwicklung. Besonders hervorgehoben wird das durch die beiden in den USA geprägten Schlagworte Hardware und Software, die man auch mit materieller und immaterieller Ware übersetzen kann. Der Aufwand für Organisation und Programmierung der heutigen Großrechenanlagen übersteigt oft schon denjenigen für die eigentlichen Geräte. Die Bedeutung des Mathematikers in der Wirtschaft und in der Industrie ist dementsprechend stark gestiegen.

Zunächst unabhängig von der Rechengeräteentwicklung, jedoch in einem losen Zusammenhang mit ihr sind in den letzten Jahrzehnten noch einige Theorien und Wissenschaftszweige entstanden, wie die Informationstheorie, die Automatentheorie und die Kybernetik. Die Informationstheorie befaßte sich ursprünglich in erster Linie mit der Leistungsfähigkeit von Nachrichtenübertragungsanlagen, und man sollte in ihr nicht eine Theorie der Informationsverarbeitung sehen, obgleich sie zweifelsohne auch für dieses Gebiet einige Anregungen gegeben hat. Ebenso ist die Automatentheorie nicht eine Theorie der Automation, sondern sie hat sich verhältnismäßig abstrakt als selbständiger Zweig der Mathematik entwickelt.

Dementsprechend sind die Verbindungen zwischen diesen Theorien und der Praxis der Rechenmaschinenkonstruktion einschließlich Software-Entwicklung verhältnismäßig lose. Es ist noch nicht zu einer so fruchtbaren Zusammenarbeit gekommen, wie es vielleicht wünschenswert wäre. Die Kybernetik hat sich unter anderem zur Aufgabe gestellt, Brücken zwischen verschiedenen Wissenschaftszweigen herzustellen und auszubauen, wobei insbesondere biologische Betrachtungen mit einbezogen werden.

In diesem Rahmen haben einige Untersuchungen Bedeutung erlangt, die unter dem Namen „lernende Systeme“ [XIII 9], „sich selbst organisierende Systeme“ (XIII 2], „künstliche Intelligenz“ [XIII 5], „allgemeine Problemlösung“ (XIII 6], „sich selbst reproduzierende Systeme“ [XIII, 6, 11, 12] usw. bekannt geworden sind. Große Bedeutung haben ferner die formalen Sprachen, insbesondere die algorithmischen Sprachen, erlangt.

Heute setzt sich als Sammelname für die Informationsverarbeitung und ihrer angrenzenden Gebiete der Begriff „Informatik“ durch. Erfreulicherweise werden bereits an einigen Universitäten und Hochschulen Lehrstühle und Studienzweige für diese Richtung eingeführt.

Wir stehen noch mitten in dieser Entwicklung. Dadurch wird eine Fülle von Fragen aufgeworfen, die erst teilweise oder noch nicht in der richtigen Weise erkannt worden sind. Auf diese Problematik soll im folgenden eingegangen werden. Dabei kann nur ein kurzer Überblick versucht werden. Manche Frage kann nur gestellt,

aber noch nicht beantwortet werden.

Über Formeln und Algorithmen

Der Verfasser hat, als er mit seinen Arbeiten begann, versucht, den Begriff des Rechnens folgendermaßen zu formulieren: „Rechnen heißt, aus gegebenen Angaben nach einer Vorschrift neue Angaben bilden.“ Anstelle von Angaben sind heute die Worte „Daten“ bzw. „Information“ getreten. Für Vorschrift, insbesondere Rechenvorschrift, wird heute das etwas strengere Wort „Algorithmus“ gesetzt. Im englischen Sprachbereich pflegt man auch von „transformation of information“ zu sprechen, was wohl etwa auf dasselbe hinausläuft.

Bei dieser modernen Auffassung des Begriffes Rechnen ist es wesentlich, daß er über das reine Zahlenrechnen hinaus erweitert worden ist. Dabei hat die mathematische Logik Pate gestanden. Schon Leibniz sprach vom „Ausrechnen einer Streitfrage“ und entwickelte seine „ars combinatoria“, die als erster Versuch gelten kann, die Logik zu formalisieren. Unabhängig von der Informationstheorie ließ sich bald erkennen, daß sich alle Rechenprozesse letzten Endes in solche mit Ja-Nein-Werten, heute auch als „Bit“ bezeichnet, auflösen lassen.

Damit ist dem lnformatiker ein idealer Elementarbaustein in die Hand gegeben. Die Elementaroperationen stimmen mit den drei aussagenlogischen Operationen Konjunktion, Disjunktion und Negation überein. Allerdings hat man nicht immer folgerichtig die Konsequenz aus dieser Tatsache gezogen. Das zeigt sich z. B. an den heute eingeführten Programmiersprachen, die noch zu sehr in der alten Vorstellung verwurzelt sind, daß Rechnen in erster Linie aus numerischen Operationen besteht.

Rechnerische Prozesse können rein formal in algorithmischer Form als Rechenanweisungen beschrieben werden. Zu ihrer Ausführung mit Hilfe technischer Mittel sind dann Rechengeräte erforderlich, die früher aus Ziffernrädchen, Achsen und Verbindungsstangen bestanden und heute in erster Linie in Form von elektronischen Schaltungen aufgebaut sind. Nur letztere sind jetzt noch von Bedeutung und lohnen, theoretisch behandelt zu werden. Diesen materiell ausgeführten Schaltungen, der Hardware, können nun wieder theoretische Modelle, meist in Form von Zeichnungen, zugeordnet werden. Sowohl bei der formalen als auch bei der zeichnerischen Darstellung haben wir es mit Modellen rechnender Strukturen zu tun. Dabei sind in vielfältiger Art Brücken zwischen beiden Darstellungsformen möglich, die im folgenden etwas näher betrachtet werden sollen.

Mathematische Formeln sind Schriftsprachen, durch die Beziehungen zwischen gedachten Dingen, wie z. B. Variablen, in eindeutig formaler Weise und möglichst ohne Benutzung der Umgangssprache dargestellt werden sollen. Heute stellt man

dabei noch gern die Forderung, daß solche Formeln aus Zeichenketten bestehen, welche einem vorgegebenen Alphabet entnommen sind. Also z. B.

$$ab + a \cdot c = a(b + c).$$

In dieser Formel sind die Zeichen $a, b, c, +, \cdot, (,), =$ verwendet. Oft wird auch der „Zwischenraum“ als besonderes Trenn-Zeichen aufgefaßt. Derartige Zeichenketten haben den Vorteil, technisch leicht übertragen, gespeichert und verarbeitet werden zu können. Sie können z. B. bei der Wahl eines passenden Codes durch Fernschreiber übermittelt werden.

Aber nicht alle in der Mathematik eingeführten formalen Darstellungen genügen dieser Bedingung. Oft spielt die topologische Anordnung der Zeichen untereinander eine wesentliche Rolle.
Zum Beispiel:

$$\frac{a}{b}, a^2, \sqrt{u+v}, A_i, R', \bar{B}, \sum_{i=1}^{n}. \tag{1}$$

Solche Formeln können jedoch immer „gestreckt“, d.h. in eine Zeichenkette aufgelöst werden. Zum Beispiel

$$a : b, a \uparrow 2, \sqrt{}(u+v), A \downarrow i, R', -B, \sum 1, n, A \downarrow n. \tag{2}$$

Das bedarf selbstverständlich besonderer Vereinbarungen. Im allgemeinen leidet bei solchen Auflösungen in gestreckte Zeichenketten jedoch die leichte Lesbarkeit für den Mathematiker. Aus diesem Grund bemüht man sich auch schon vielfach, im Interesse eines guten Zusammenspiels zwischen Mensch und Maschine, beide Arten nebeneinander zu verwenden, wobei allerdings schon bei einfachen Formen schwierige Übersetzungsprobleme auftreten können. Die Ausgabe von formalen Darstellungen durch die Maschine in einer dem Mathematiker geläufigen Form erfordert erheblichen konstruktiven Aufwand (Sichtgeräte), so daß ein gutes Zusammenspiel zwischen Mensch und Maschine heute meistens an den Kosten scheitert.

Nicht jede formale Darstellung kann jedoch als Rechenanweisung benutzt werden. Man unterscheidet implizite und explizite Darstellungen. In der quadratischen Gleichung

$$x^2 + ax + b = 0 \tag{3}$$

ist x implizit gegeben. Erst die explizite Form

$$x = -\frac{a}{2} \pm \sqrt{\frac{a^2}{4} - b} \tag{4}$$

kann zum „Ausrechnen“ von x benutzt werden.

Allerdings kann man auch die in impliziter Form gegebene quadratische Gleichung als Rechenanweisung auffassen, wenn die drei Werte a, b und x gegeben sind und lediglich festgestellt werden soll, ob sie der obigen Gleichung genügen. Das Ergebnis dieser Kontrollrechnung ist dann ein Ja-Nein-Wert: „x ist richtig berechnet.“ Die umfangreiche Problemstellung, die sich aus dieser zur Ausrechnung erforderlichen Umwandlung von impliziten in explizite Formen ergibt, soll jedoch nicht das Thema dieser Abhandlung sein.

Wir wollen uns also im folgenden in erster Linie auf solche Formeln beschränken, die in expliziter Form als Zeichenketten gegeben sind. Auch dabei haben sich verschiedene Darstellungsformen eingebürgert. Da die Formeln im allgemeinen Verknüpfungen zwischen Größen darstellen, ergeben sich zunächst die beiden wichtigen Gruppen:

1. Variablenzeichen
2. Verknüpfungszeichen

Die Variablenzeichen dienen der Kennzeichnung von Größen, die keineswegs auf numerische Werte beschränkt sind. Sie können Stellvertreter für Ja-Nein-Werte, reelle Zahlen, komplexe Zahlen, Vektoren, Matrizen, Listen usw. sein. An die Stelle des Variablenzeichens kann dabei auch ein Konstantenzeichen, z. B. eine bestimmte Zahl treten.

Als Verknüpfungszeichen haben sich drei Formen eingebürgert: Operationszeichen, Operatoren- und Funktionszeichen. Die Operationszeichen werden zwischen zwei Variablenzeichen gesetzt. Operatoren- und Funktionszeichen kommen im allgemeinen vor den bzw. die Operanden (Argumente).
Zum Beispiel:

$$\begin{array}{ll} \text{Operationszeichen:} & a + b \\ \text{Operatorzeichen:} & \sqrt{}(a + b) \\ \text{Funktionszeichen:} & \sin \alpha \end{array} \tag{5}$$

Es ist bekannt, daß diese Unterscheidung nicht notwendig ist. Auch Operationszeichen können als Operatoren vor die Operanden gesetzt werden:

$$+(a, b) \tag{6}$$

Das kann z. B. bei einer mehrgliedrigen Summe sogar übersichtlicher sein:

$$+(a, b, c, d). \tag{7}$$

Die konsequente Anwendung dieses Operatorprinzips erfolgt in der sogenannten „polnischen Notation“:

$$k\sqrt{\frac{a^2+b^2}{c}} \equiv \times(k, \surd(:\,(+(\times(a,a), \times(b,b)), c))) \tag{8}$$

Man sieht leicht, daß die übliche linke Darstellung für das Auge übersichtlicher ist. Das gilt jedoch nicht für die Maschine. Die rechte Darstellung eignet sich gut zur Eingabe in einen Rechenautomaten und ist bereits „programmgerecht“ zugeschnitten.

Zu den Variablen- und Operationszeichen kommen als dritte wichtige Gruppe die Zusammenfassungszeichen, die im allgemeinen die Form von Klammern haben. Dadurch werden gewissermaßen einzelne Formelteile zu „Paketen“ zusammengefaßt und als größere Einheiten wiederum durch Operationszeichen verknüpft. Auch dabei zeigt sich ein wesentlicher Unterschied zwischen menschlicher und maschineller Arbeitsweise. Der Mathematiker pflegt verschiedene Arten und Größen von Klammerzeichen zu verwenden, z. B.: $[, \langle, \{, (\ldots$ Logisch genügt aber eine einzige Art von Klammerzeichen, sofern man den verschiedenen Klammerzeichen nicht spezielle Bedeutungen zuordnet, wie z. B. in der Mengenlehre den Zeichen $\{\}$. Allerdings sind solche vereinfachten Darstellungen wiederum für das Auge schwer lesbar. Obiges Beispiel (8) wird z. B. wesentlich übersichtlicher, wenn man Klammern verschiedener Art und Größe verwendet:

$$\times[k, \surd\{:\,\langle+(\times(a,a), \times(b,b)), c\rangle\}]. \tag{9}$$

Wenn man das Bild der „Verpackung“ wörtlich anwendet, so läßt sich die Formel auch wie folgt schreiben bzw. zeichnen:

$$k \times \boxed{\surd\boxed{\boxed{a \times a} + \boxed{b \times b}} : c} \tag{10}$$

Läßt man die horizontalen Linien fort, so erhält man einen Ausdruck mit Klammern verschiedener Größe:

$$k \times \left[\surd\Big[\big[[a \times a] + [b \times b]\big] : c\Big]\right] \tag{11}$$

Die Zahl der Klammern kann durch Vereinbarungen über eine Rangfolge in der „Bindungsstärke“ verschiedener Operationszeichen vermindert werden. Zum Beispiel bindet das Multiplikationszeichen enger als das Additionszeichen, so daß wir anstatt $(a \cdot b) + (c \cdot d)$ schreiben können:

$$a \cdot b + c \cdot d. \tag{12}$$

Dabei wird auch noch häufig das Multiplikationszeichen fortgelassen und einfach geschrieben:

$$ab + cd. \tag{13}$$

Das setzt allerdings voraus, daß Variable stets nur durch ein einzelnes Zeichen dargestellt werden, was eine wesentliche Beschränkung bedeutet, die sich nur mangelhaft durch Anhängen von Indices usw. umgehen läßt. In modernen algorithmischen Sprachen sind deshalb auch derartige Vereinfachungen nicht erlaubt. Man darf jedoch einen Wert durch eine Kette von beliebig vielen Buchstaben darstellen, z. B.: Feld.

Die Klammerdarstellung ist nicht die einzige Möglichkeit, um Gruppierungen zu kennzeichnen. Betrachtet man das Bild der Formel (9), beobachtet man eine Verschachtelung von Größen. Man,kann auch von einer „Klammerbilanz“ sprechen. Ordnet man den nicht eingeklammerten Variablen den Index 0 zu, so kann jedem Klammerpaar ebenfalls ein Index zugeordnet werden; diese Indices seien zunächst unter die Zeichen gesetzt:

$$\underset{5}{\left[\times\right.}\, k, \underset{4}{\left\{\sqrt{}\right.}\, : \underset{3}{\left\langle} + \underset{2}{\left(} \times \underset{10}{(} a, a \underset{01}{)}, \times \underset{10}{(} b, b \underset{01}{)} \underset{2}{\right)} \underset{3}{\left.\right\rangle} \underset{4}{\left.\right\}} \underset{5}{\left.\right]} \tag{14}$$

Es ergibt sich hierbei ein Verschachtelungsgrad bzw. die Klammerbilanz 5. Diese Indices können auch den Operationszeichen zugeordnet werden, wobei dann die Klammerzeichen fortfallen können, ohne den Sinn der Formel zu verändern

$$\underset{5}{\times} k, \underset{4}{\sqrt{}} \underset{3}{:} \underset{2}{+} \underset{1}{\times} a, a, \underset{1}{\times} b, b, c. \tag{15}$$

Eine unmittelbar anschauliche Form erhält man, wenn man die Operatorzeichen ihrem Index entsprechend in verschiedene Höhen setzt, wodurch ihre „Reichweite“ symbolisiert wird:

$$\begin{array}{lccccccccc} 5 & \times & & & & & & & & \\ 4 & & k\sqrt{} & & & & & & & \\ 3 & & & : & & & & & & \\ 2 & & & & + & & & & c & \\ 1 & & & & & \times & & \times & & \\ 0 & & & & & & ab & & ab & \end{array} \tag{16}$$

Hierbei wird außerdem noch von der Möglichkeit Gebrauch gemacht, die Variablenzeichen in verschiedene Höhen zu setzen, wodurch die Kommazeichen fortfallen können. Diese Darstellungsart ist nicht an die polnische Notation gebunden.

Setzt man die Operationszeichen zwischen die Variablenzeichen, so erhält man folgende Form:

$$\begin{array}{lccccccccc}
5 & & \times & & & & & & & \\
4 & k & & \sqrt{} & & & & & & \\
3 & & & & & & & & : & \\
2 & & & & & + & & & & c \\
1 & & & & \times & & & \times & & \\
0 & & & a & & a & & b & & b
\end{array} \tag{17}$$

Nützt man die Vereinbarung über den Bindungsgrad der Operationszeichen aus, so ist folgende Vereinfachung möglich:

$$k \times \sqrt{} \; : \; c \qquad a \times a + b \times b \tag{18}$$

Man sieht, daß es zahlreiche formale Darstellungsformen gibt, die entweder mehr der Arbeitsweise des Menschen oder der der Maschine angepaßt sind.

Bei der algorithmischen Darstellung empfiehlt sich die Einführung des Ergibtzeichens bzw. des Zuweisungszeichens. Der Verfasser führte hierfür ursprünglich das Zeichen $\Rightarrow$ ein, wodurch die einseitige Wirkung des Gleichheitszeichens betont werden soll. Das erlaubt Ansätze wie folgt:

$$x + 1 \Rightarrow x \tag{19}$$

Gelesen „x plus eins ergibt x“. Mit der Bedeutung: Erhöhe den bisherigen Wert x um eins und rechne mit diesem neuen Wert x weiter: Dabei entspricht die Reihenfolge der Zeichen dem zeitlichen Ablauf der Rechnung. Links steht das Gegebene, rechts das Gesuchte. In Anlehnung an das in der Mathematik bereits weitgehend eingeführte Definitionszeichen hat sich jedoch folgende Form durchgesetzt:

$$x := x + 1. \tag{20}$$

Das hat den Vorteil, daß der neu zu bestimmende Wert am Anfang der Formel aufgeführt ist. Im folgenden wird im allgemeinen dieses Zeichen verwendet werden.

Die besprochenen Zeichenarten genügen jedoch noch nicht, um alle Rechenanweisungen klar zu formulieren; z. B. fehlen noch Darstellungsformen für rekursive und zyklische Rechenprozesse. In der Mathematik werden hierfür z. B. Summen- und Produkt-Zeichen

$$\sum \prod \tag{21}$$

verwendet, wobei die zu variierende Größe und die Variationsbreite angegeben werden muß.

Bei einer explizit vorliegenden formalen Darstellung unterscheidet man Eingabewerte, Zwischenwerte und Ergebniswerte. Das Wort „Ausgangswert“ soll ganz vermieden werden, da es zweideutig ist. Meistens versteht man darunter Werte, von denen man ausgeht, also die „Eingabewerte“. Bei Rechenmaschinen spricht man jedoch von Eingabe und Ausgabe, so daß „Ausgabewert“ mit „Ausgangswert“ verwechselt werden könnte. Leider hat sich diese Erkenntnis noch nicht allgemein durchgesetzt.

In einer einzelnen geschlossenen Formel werden die Zwischenwerte nicht durch besondere Zeichen gekennzeichnet. So gibt es in der Formel (8) folgende Zwischenwerte:

$$a^2, b^2, a^2 + b^2, \frac{a^2 + b^2}{c}, \sqrt{\frac{a^2 + b^2}{c}} \tag{22}$$

Man kann jedoch eine Formel aufteilen, indem man für einen oder mehrere Zwischenwerte besondere Bezeichnungen einführt, z. B.

$$\begin{array}{c} a^2 + b^2 = p, \\ k\sqrt{\frac{p}{c}} = R \end{array} \tag{23}$$

Die Einführung von Zwischenwerten dient aber nicht nur der Auflösung einer längeren Formel, um die Übersicht zu erleichtern, sondern kann auch ökonomische Gründe haben, falls ein bestimmter Ausdruck mehrmals in einer Formel vorkommt; z. B.

$$\frac{\sqrt{a^2 + b^2}}{1 + \sqrt{a^2 + b^2}} = C \tag{24}$$

Hierbei bedeutet die Einführung eines Zwischenwertes für die Wurzel die Vermeidung unnützen Doppelrechnens. Wir schreiben diese Formel gleich als Rechenanweisung:

$$\begin{array}{c} p := \sqrt{a^2 + b^2} \\ C := \frac{p}{1+p} \end{array} \tag{25}$$

Dieses Verfahren kann nun bis zum Extrem getrieben werden. Bei expliziten, also algorithmischen Formeln, erhält man dann eine Folge von Einzelanweisungen, welche je nur eine Rechenoperation mit der Aufzählung ihrer Operanden enthalten.

Die Formel (8) nimmt dann folgende Gestalt an:

$$\begin{array}{l} d := a \times a \\ e := b \times b \\ f := d + e \\ g := f : c \\ h := \sqrt{g} \\ R := k \times h \end{array} \tag{26}$$

Nunmehr ergeben die vier Eingabewerte a, b, c, k die Zwischenwerte d, e, f, g, h und den Resultatwert R.

Damit ist die Form des „Programms“ gefunden, die schon Babbage benutzte. Dabei hatte er allerdings noch einen weiteren Schritt vollzogen, er ersetzte nämlich die Buchstaben durch Variable mit fortlaufend numerierten Indices

$$\begin{array}{ll} V_1 := a & V_6 := V_1 \times V_1 \\ V_2 := b & V_7 := V_2 \times V_2 \\ V_3 := c & V_8 := V_6 + V_7 \\ V_4 := k & V_9 := V_8 : V_3 \\ V_5 := R & V_{10} := \sqrt{V_9} \\ & V_5 := V_4 \times V_{10} \end{array} \tag{27}$$

Die auf Seite 261 erklärte einseitige Wirkung des Zeichens := erlaubt es, Benennungen für Zwischenwerte, die später nicht mehr benötigt werden, mehrmals zu verwenden. Benutzt man hierfür das Zeichen V_0, so erhält man folgende Form des Programms:

$$\begin{array}{ll} V_6 := V_1 \times V_1 & V_0 := \sqrt{V_0} \\ V_0 := V_2 \times V_2 & V_5 := V_0 \times V_4 \\ V_0 := V_0 + V_0 & \\ V_0 := V_0 : V_3 & \end{array} \tag{28}$$

Durch Umkehrung der polnischen Notation (Formel 8), erhält man folgende Form: indem man die Operationszeichen nicht vor, sondern hinter die Variablen setzt.

$$\Big(\Big(\Big(\big((V_1, V_1)\times, (V_2, V_2) \times\big)+, V_3\Big) : \Big)\surd, V_4\Big)\times \Rightarrow V_5 \tag{29}$$

Ohne das Verständnis zu beeinträchtigen, kann man dabei noch die Klammern und Kommazeichen fortlassen, wenn man den in der Formel nicht besonders bezeichneten Zwischenwert V_6 entsprechend Programm (28) herauszieht:

$$V_1 V_1 \times \underset{V_6}{\downarrow} \quad V_2 V_2 \quad \underset{V_6}{\uparrow} + V_3 : \sqrt{V_4} \times \Rightarrow V_5 \tag{30}$$

Man erhält dann eine formale Darstellung, die unmittelbar als Programm für einen Rechenautomaten dienen kann. Bei dieser Schreibweise erkennt man auch den Vorteil des Ergibtzeichens⇒ gegenüber dem Zeichen : =.

Zunächst mußjedoch noch die Frage der Auflösung in Einzeloperationen näher untersucht werden. Beim oben besprochenen Beispiel bestehen die Variablen aus numerischen Werten und die Operationen aus arithmetischen Verknüpfungen dieser Werte. Wie bereits besprochen, ist die elementare Informationseinheit jedoch

das Bit, und die elementaren Rechenoperationen sind Verknüpfungen zwischen solchen Ja-Nein- Werten, also Konjunktion, Disjunktion und Negation. Man hat es also bei den arithmetischen Operationen gewissermaßen nur mit rechnerischen Molekülen zu tun, die sich wiederum in rechnerische Atome zerlegen lassen. Die Auflösung numerischer Werte in Ja-Nein-Werte ist ohne weiteres verständlich. Die einfachste Form stellen dabei die Binärzahlen dar, bei denen die einzelnen Ziffern bereits Ja-Nein-Werte sind. Bekanntlich lassen sich aber auch Dezimalzahlen in Bits auflösen, wobei man jede Dezimalziffer mit 4 Bit darstellt. Einige der oft verwendeten Codes seien angeführt:

	Direkter Binärcode	Stibitz-Code
0	*OOOO*	*OOLL*
1	*OOOL*	*OLOO*
2	*OOLO*	*OLOL*
3	*OOLL*	*OLLO*
4	*OLOO*	*OLLL*
5	*OLOL*	*LOOO*
6	*OLLO*	*LOOL*
7	*OLLL*	*LOLO*
8	*LOOO*	*LOLL*
9	*LOOL*	*LLOO*

(31)

Links stehen die Ziffern in direktem Binärcode verschlüsselt; rechts im sogenannten Stibitz-Code oder auch Drei-Exzeß-Code. Dabei sind sämtliche Ziffern gegenüber dem direkten Code um LL erhöht, was Vorteile bei der Komplementbildung zwecks Subtraktion bietet. Auf das sehr interessante Kapitel der verschiedenen Vercodungen soll hier jedoch nicht weiter eingegangen werden.

Bekannt ist, daß sich Multiplikation und Division durch wiederholte Addition und Subtraktion aufbauen lassen. Auch die Addition ist nun wieder in Operationen zwischen einzelnen Bits auflösbar, wofür im folgenden ein Beispiel gegeben sei:

Es seien die einzelnen Ziffern einer natürlichen Binärzahl entsprechend ihrem Stellenwert mit den Indices $0 \rightarrow n$ bezeichnet. Bei der Addition ergeben sich dann zwei Summanden a, b mit den Ziffern $a_0 \ldots a_i \ldots a_n$, $b_0 \ldots b_i \ldots b_n$. Die Summe c hat die Ziffern $c_0 \ldots c_i \ldots c_{n+1}$.

Die Rechnung ist zyklisch für jede einzelne Binärstelle, wobei in jeder Stelle außer den beiden Ziffern a_i, b_i der beiden Summanden der Stellenübertragung zu berücksichtigen ist, der von der nächstniederen Stelle i – 1 auf die Stelle i übertragen wird. Aus diesen drei Eingabewerten a_i, b_i, u_1 sind zwei Ergebniswerte c_i

und u_{i+1} zu ermitteln. Wie man leicht erkennt, gelten hierfür folgende Formeln:

$$\begin{aligned} c_i := \quad & (a_i \wedge b_i \wedge u_i) \\ \vee \quad & (a_i \wedge \neg b_i \wedge \neg u_i) \\ \vee \quad & (\neg a_i \wedge b_i \wedge \neg u_i) \\ \vee \quad & (\neg a_i \wedge \neg b_i \wedge u_i) \\ u_{i+1} := \quad & (a_i \wedge b_i) \vee (a_i \wedge u_i) \vee (b_i \wedge u_i) \end{aligned} \tag{32}$$

Diese Formeln lassen sich vereinfachen bzw. entsprechend (27) und (28) in Programme auflösen, welche nur Anweisungen für einzelne aussagenlogische Operationen enthalten.

Allerdings ist bei digitalen Darstellungen numerischer Werte und ihrer Operationen eins zu bedenken: Alle digitalen Zahlensysteme arbeiten mit begrenzter Genauigkeit. Die Forderung, die z. B. durch das Peanosche Axiomensystem für reelle Zahlen an die Verknüpfungen gestellt werden, lassen sich durch kein reales System erfüllen. Sie können jedoch durch Erhöhung der Stellenzahl genügend genau angenähert werden. Den Axiomen lassen sich daher auch nicht einzelne konkrete Automaten bzw. Rechengeräte zuordnen, sondern Reihen von Automaten, deren Aufbau rekursiv festgelegt ist. Das heißt, es muß die Beschreibung für einen Automaten mit der Stellenzahl n vorliegen und eine Anweisung, wie man von diesem Automaten durch Erweiterung seiner Glieder zu einem entsprechenden der Stellenzahl n + 1 kommt. Erst durch den praktisch nicht ausführbaren Grenzübergang $n \to \infty$ werden die Axiome exakt erfüllt.

Die Erkenntnis, daß die arithmetischen Operationen in solche mit Ja- Nein-Werten auflösbar sind, läßt sich verallgemeinern. Diesen Weg ist der Verfasser bei der Aufstellung des Plankalküls konsequent gegangen [XIII 13, 14, 15]. Bei den bisher behandelten Formeln wurden solche verwendet, die Verknüpfungen zwischen numerischen Werten und solcher welche Verknüpfungen zwischen Ja-Nein-Werten darstellen. Bei der Verallgemeinerung des Begriffes „Rechnen“ im Sinne der auf Seite 255 gegebenen Definition können jedoch innerhalb der gleichen Formeln Daten sehr verschiedener Struktur auftreten. Ein wirklich universeller Formalismus muß in dieser Hinsicht völlig frei sein. Das zeigt ein Blick auf das Schachspiel: Etwa ein Dutzend Angaben verschiedener Struktur, wie „Feldbesetzung“, „Zugangabe“, „Spielsituation“, „Liste der Steine“, gehen als Informationseinheiten in die Rechnung ein oder werden als Resultate bestimmt. So besteht die „Feldbesetzung“ aus der Aufzählung der vierundsechzig Felder und ihrer Besetzung. Bei der „Spielsituation“ ist diese Angabe durch den Ja-Nein-Wert „Weiß oder Schwarz am Zuge“ und Angaben über bereits vollzogene Rochaden ergänzt, ferner noch durch eine zusätzliche Information, welche in speziellen Fällen angibt, ob „en passant“ geschlagen werden darf.

Vor die Aufgabe, mit Werten verschiedener Struktur und Bedeutung zu arbeiten, ist der Mathematiker schon immer gestellt. Man behilft sich mit unterschiedlichen Zeichenklassen, wie römischen, griechischen oder deutschen Buchstaben.

So stellt man zum Beispiel Vektoren mit deutschen Buchstaben dar, oder man benutzt Unter- und Überstreichungen, Indices in verschiedenen Positionen und dergleichen. Erinnert sei an Zeichen wie

$$\alpha_p \, \mathfrak{A}_{i-1}^{P} \, '\mathfrak{B}_{\beta.k}$$

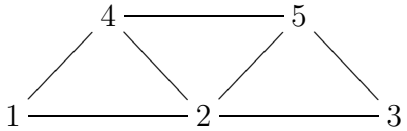
Oft kann man den Sinn solcher Zeichen nur aus dem Zusammenhang erraten. Wie werden im Plankalkül durch einen rein schematischen „Strukturkalkül“ ersetzt, dessen wichtigste Punkte hier nur angedeutet werden können.

Aufbau zusammengesetzter Datenstrukturen:

$S0$	Ja-Nein-Wert
$S1 \cdot n = n \times S0$	n-stellige Folge von Ja-Nein-Werten
$S1 \cdot 4$	Tetrade
σ	allgemeines Zeichen für Angabe beliebiger Struktur
2σ	Paar von Werten
$m \times 2\sigma$	Paarliste

Auf diese Weise lassen sich z.B. beliebige technische Strukturen, wie Netzwerke, Stabwerke, Konstruktionen usw., darstellen.

Beispiel: Stabwerk



Paarliste

$1 - -2$
$2 - -3$
$1 - -4$
$2 - -4$
$2 - -5$
$4 - -5$
$3 - -5$

Die Strukturen sind aber damit nicht ausreichend gekennzeichnet, denn der Begriff „Zahl“ kann durch sehr unterschiedliche Strukturenarten dargestellt werden, wie Binärzahl, Dezimalzahl, Gleitkommadarstellung usw. Man braucht dann den übergeordneten Begriff der Datenart.

Darüber hinaus wird eine einfache und doch übersichtliche Darstellungsart der verschiedenen Variationen von Strukturen und Zeichenarten benötigt. Eine Möglichkeit besteht darin, die Zeichen in Komponenten, die in verschiedene Zeilen gesetzt werden, zu zerlegen.

Neben der Hauptzeile, weiche die Formel im wesentlichen in der traditionellen Form enthält, wird eine zweite Zeile (V) für den Variablen-Index, eine dritte für den Komponenten-Index (K) und eine vierte für den Struktur-Index (S) eingeführt. Die letztere braucht, strenggenommen, nicht immer ausgefüllt zu werden, dient aber wesentlich zur Erleichterung des Verständnisses einer Formel. Die

Zeilen werden durch Vorsetzen der zugeordneten Buchstaben (V, K, S) gekennzeichnet.

Beispiele:

V V 3 K S $m \times 2 \times 1.n$		Die Variable V_3, ist eine Paarliste von m Paaren der Struktur 2 x 1.n und soll als Ganzes in die Rechnung eingehen.
V V 3 K i S $2 \times 1.n$		Von der Paarliste V_3 soll das i-te Paar genommen werden (Struktur 2 x 1.n). (i kann dabei ein laufender Index sein.)
V V 3 K $i.0$ S $1.n$		Von dem i-ten Paar der Paarliste V_3 soll das Vorderglied (erstes Element des Paares) genommen werden (Struktur 1.n).
V V 3 K $i.0.7$ S 0		Von dem Vorderglied des i-ten Paares der Paarliste V_3 soll der Ja-Nein-Wert Nr. 7 genommen werden (Struktur S0 = Ja-Nein-Wert).

Besonders wichtig ist die Möglichkeit, durch den Komponenten-Index (K) beliebige Teile aus einer großen Informationseinheit herauslesen zu können. So könnte die Aufgabe darin bestehen, beim Schachspiel aus einer kompliziert aufgebauten „Spielsituation“ lediglich die Teilangabe „Schwarz darf die große Rochade ausführen“ Für eine weitere Rechnung zu isolieren.

Umgekehrt kann man die Angaben zusammenfassen, um Komponenten zu bilden. Hierdurch entstehen höhere lnformationseinheiten.

Bei den heute praktisch eingeführten Programmiersprachen ist man diesen Weg allerdings nicht gegangen. Da man zunächst nur das Ziel hatte, numerische Rechnungen zu programmieren, unterschied man nur zwischen verschiedenen Zahlenarten, z. B. ganzen Zahlen (INT) und reellen Zahlen (REAL). Dazu kommen eventuell Unterscheidungen zwischen Festkommadarstellung und Gleitkommadarstellung. Erst später ergänzte man diese Strukturen durch Boolesche Variable (Boolean), Vektoren (Array) und Listen von Zahlen (STRING). Dies gilt in erster Linie für die Sprachen FORTRAN und ALGOL. Die mehr für kommerzielle Belange zugeschnittene Sprache COBOL erlaubt allerdings eine größere Variabilität der Datenstrukturen. Sie paßt sich dabei der freien Aufteilungs- und Kombinationsmöglichkeit der Lochkarte in bezug auf die Spalten an.

In den seit der ersten Einführung dieser Sprachen vergangenen 10 Jahren hat

man allerdings erkannt, daß das bisherige System zu starr ist. Es ist jedoch kaum möglich, jetzt noch auf das dem Plankalkül zugrunde liegende Prinzip konsequent umzuschalten. Hier liegt eine große Problematik der heutigen Situation auf dem Computergebiet. Die Dinge sind noch voll im Fluß, und wirklich durchgreifende Lösungen liegen noch vor uns.

Übergang von Formeln zu Schaltungen

Unter Schaltungen versteht man zunächst konkrete materielle Einrichtungen, bei denen verschiedene Elemente zu einem System zusammengeschaltet sind. Sie haben in erster Linie bei elektrischen Netzen aller Art Bedeutung. Jedoch kann man auch Rohrleitungsnetze und ähnliche Einrichtungen als Schaltungen auffassen. Die Beschreibung solcher Netze erfolgt seit langem durch Schaltzeichnungen oder Schaltpläne. Ein Schaltplan verhält sich zu einer Konstruktionszeichnung etwa wie die Topologie zur Geometrie. Bei Schaltzeichnungen kommt es lediglich auf die gegenseitigem Beziehungen der Elemente untereinander an. Die Möglichkeit, eine elektrische Leitung auch in der praktischen Ausführung räumlich fast beliebig verlegen zu können, erlaubt es, die zu verbindenden Elemente in der Schaltzeichnung in eine logische bzw. organische Ordnung zu bringen, ohne auf ihre tatsächliche räumliche Anordnung Rücksicht nehmen zu müssen.

Schon von Anfang an befaßte man sich auch mit den theoretischen Grundlagen solcher Schaltungen. Zunächst wurden mehr die Gesetze der Stromverteilung in komplizierten Netzen behandelt, wie sie z. B. durch die Kirchhoffschen Gesetze festgelegt sind. Das führte zu einigen grundsätzlichen Erkenntnissen über den Zusammenhang zwischen Schaltungen und mathematischen Systemen. Da die Stromverteilung in einem Leitungsnetz mit beliebig verteilten Maschen und Ohmschen Widerständen sich mit Hilfe linearer Gleichungssysteme bestimmen läßt, kann man auch umgekehrt ein solches Leitungsnetz bei passendem Aufbau als Modell zur Lösung eines linearen Gleichungssystems verwenden. Derartige Geräte sind unter der Bezeichnung „Netzmodell“ in verschiedenen Ausführungen gebaut worden. Sie fallen in die Klasse der Analoggeräte, da bei ihnen numerische Größen durch physikalische Größen „analog“ repräsentiert werden (Widerstände, Spannungen, Stromstärken). Derartige Netzdurchrechnungen werden heute jedoch meistens mit Digitalrechnern durchgeführt.

Erst verhältnismäßig spät hat man versucht, auch die unstetigen Schaltfunktionen von Schaltern, Kontakten usw. formal zu erfassen. Als erster hat hierüber Claude Shannon [XIII 8] geschrieben, etwas später (1939) folgte eine Arbeit von Hansi Piesch [XIII 7]. Die etwa gleichzeitigen Arbeiten des Verfassers sind nicht veröffentlicht worden. Heute ist die Schaltungsmathematik bzw. die Schaltalgebra ein weit ausgebautes Spezialgebiet der Informatik bzw. der Automatentheorie.

Man weiß, daß Formeln und abstrakte Schaltmodelle nur verschiedene Ausdrucksformen derselben funktionalen Beziehungen darstellen. Um das zu zeigen, sei an die Formel (8) angeknüpft.

$$R := k\sqrt{\frac{a^2 + b^2}{c}}$$

Wir gehen über zur Darstellungsforin (17), die sich besonders leicht in, eine Schaltung überführen läßt:

Abbildung 1:

Man erhält so das Bild eines „Baumes“, den wir noch einmal in etwas anderer Form zeichnen wollen, ohne die topologische Anordnung der Elemente zu ändern.

Diese Schaltung hat die Eingänge a, b, r, k und den Ausgang R. Die Verknüpfungselemente sind Additionsglieder, Multiplikationsglieder und ein Glied zum Quadratwurzelziehen.

Abbildung 2:

Will man eine derartige Schaltung vom Papier in die Wirklichkeit übertragen, so braucht man entsprechende konstruktive Elemente. Nach diesem Prinzip sind z. B. Analoggeräte gebaut. Die ersten Ausführungen waren meist mechanisch. Heute verwendet man elektronische Bauelemente. Das Additionsglied kann bei mechanischen Ausführungen z. B. leicht durch einen Waagebalken bzw. durch ein

Abbildung 3:

Ausgleichsgetriebe, wie es in die Hinterachse eines Autos eingebaut ist, verwirklicht werden (Bild 3). Das Getriebe hat zwei Eingänge und einen Ausgang, so wie es auch der abstrakten Darstellung in der Schaltung entspricht. Andere Getriebe, z. B. zum Multiplizieren, sind in mechanischer bzw. elektronischer Ausführung etwas schwieriger. Es sei hier nur am Rande erwähnt, daß solche Analoggeräte auch in der Lage sind, direkt zu integrieren. Eine mechanische Ausführung stellt das Reibscheibengetriebe (Bild 4) dar. Derartige Bauelemente lassen sich einer Schaltzeichnung entsprechend in verschiedener Weise zusammenschalten, wobei selbstverständlich gewisse technische Grenzen zu beachten sind. Ein wesentlicher Unterschied zu digitalen Geräten besteht darin, daß man mit ihnen kontinuierliche Abläufe simulieren kann, während man bei digitalen Geräten im allgemeinen nur eine Formel mit bestimmten Eingabewerten durchrechnet. Deshalb eignen sich Analoggeräte sehr gut für Schwingungsrechnungen. Man arbeitet heute vielfach mit sogenannten Hybridrechnern. Diese bestehen aus digitalen und analogen Teilen, wobei jeder jeweils die Aufgaben übernimmt, für die er besser geeignet

Abbildung 4:

ist. An den Übergangsstellen sind dann „Digital-Analog-Wandler“ und „Analog-Digital-Wandler“ erforderlich (Bild 5).

Abbildung 5:

Dieses Prinzip kann man nun auch auf nichtnumerische Größen übertragen. Dem Ja-Nein-Wert entspricht z. B. ein einfacher Schalter bzw. Kontakt. Es sei zunächst in gleicher Weise wie beim Bild 2 die für das Addieren gefundene Formel (32) als Schaltung dargestellt. Um Schaltglieder einzusparen, muß jedoch die Anweisung 32 noch etwas den Regeln des Aussagenkalküls entsprechend umgeformt werden.

Der Ansatz für c_i besteht aus vier Disjunktionsgliedern. Wir fassen zunächst das erste und das vierte zusammen, wobei wir der Einfachheit halber die Indices i fortlassen.

$$(a \wedge b \wedge u) \vee (\neg a \wedge \neg b \wedge u) \equiv ((a \wedge b) \vee (\neg a \wedge \neg b)) \wedge u \equiv (a \Leftrightarrow b) \wedge u$$

Dasselbe führen wir mit dem zweiten und dem dritten Glied durch:

$$(a \wedge \neg b \wedge \neg u) \vee (\neg a \wedge b \wedge \neg u) \equiv ((a \wedge \neg b) \vee (\neg a \wedge b)) \wedge \neg u \equiv \neg(a \Leftrightarrow b) \wedge \neg u$$

Die beiden so erhaltenen Teilansätze können nun noch einmal zusammengefaßt werden:

$$((a \Leftrightarrow b) \wedge u) \vee (\neg(a \Leftrightarrow b) \wedge \neg u) \equiv (a \Leftrightarrow b) \Leftrightarrow u \tag{33}$$

In ähnlicher Weise können wir den Ansatz rür u_i+1 (Formel 32) zusammenfassen:

$$(a \wedge b) \vee (a \wedge u) \vee (b \wedge u) \equiv (a \wedge b) \vee ((a \vee b) \wedge u) \tag{34}$$

Diese beiden Ansätze lassen sich leicht als Schaltung in Baumform darstellen:

Abbildung 6:

Folgende Symbolik hat sich in letzter Zeit für diese Schaltelemente $\vee$, $\wedge$, eingebürgert:

Bild 6 und Bild 7 nehmen dann folgende Form an; wobei die Äquivalenz ($\Leftrightarrow$) durch die 3 elementaren Operationen dargestellt ist:

Die konstruktive Verwirklichung kann durch „Bausteine“ erfolgen, die diese Grundaufgaben lösen. Wesentlich ist dabei, daß es sich um ein unstetiges Verhalten

Abbildung 7:

Abbildung 8:

handelt. Aus diesem Grund sind alle physikalischen Verhaltensweisen von materiellen Anordnungen, die solche Unstetigkeiten aufweisen, im Prinzip für derartige Bausteine geeignet. Solche Unstetigkeiten treten z. B. an Grenzflächen zwischen verschiedenen Medien bzw. Körpern auf. Das kann bereits rein mechanisch zu einer einseitigen Wirkung der Körper a und b auf den Körper c ausgenutzt werden. Bild 9 zeigt ein einfaches Schema zur Lösung der Aufgabe der Disjunktion $a \vee b = c$, wobei die Ja-Nein-Werte a, b und c durch einfache Schieber, die je zwei Stellungen O und L einnehmen können, repräsentiert werden.

Abbildung 9:

Um das rechte Glied c von der Grundstellung O auf die Stellung L zu bringen, genügt es, eine der beiden Scheiben a oder b auf die Stellung L zu schalten. Das elektrische Analogon dazu ist eine Schaltung mit Gleichrichtern bzw. Dioden:

Es genügt, an einen der beiden Pole a bzw. b Spannung zu legen, um c auf das gleiche Spannungsniveau anzuheben. Dabei werden die Werte O bzw. L durch verschiedene Spannungsniveaus repräsentiert. Der Widerstand W sorgt dafür, daß c an der dem Wert O entsprechenden Spannung liegt, falls a bzw. b sich nicht auf dem dem Wert L entsprechenden Spannungsniveau befinden. Auch hierbei wird die physikalische Unstetigkeit einer Grenzfläche zwischen zwei Materialien ausgenutzt, die bei den modernen Halbleiterdioden durch verschiedene Dotierungen von Germanium oder Silizium erfolgt. (n-leitend, p-leitend). Auf diesem Prinzip baut die moderne „Diodenlogik“ auf. Allerdings stellen diese Dioden passive Bauelemente dar. Man braucht zum Aufbau einer Schaltung außerdem Verstärker, wofür sich die Röhre bzw. der Transistor eignet. Die zahlreichen Vorteile des Transistors haben ihm zu seinem Siegeszug verholfen. Logisch gesehen vereinigt er in sich die Dioden und den Verstärker. Konstruktiv ist er außerordentlich zuverlässig. Er eignet sich auch gut für moderne Herstellungsverfahren, bei denen auf sehr kleinem Raum durch eine integrierte Technik komplette Schaltungen hergestellt werden.

Geschichtlich gesehen ging der Weg allerdings zunächst von der Mechanik über die Elektromechanik zur Elektronik mit den Stufen Röhrentechnik, Transistortechnik und integrierte Schaltkreistechnik.

Besonders leicht verständlich sind Reiaisschaltungen. Auch sie beruhen auf der durch eine Grenzfläche, nämlich der Kontaktfläche, gegebenen Unstetigkeit, wodurch die galvanische Verbindung zwischen zwei Leitern einem strengen Ja-Nein-

Abbildung 10:

Prinzip unterworfen wird.

Abbildung 11:

Bild 11 zeigt die drei Grundschaltungen in Reiaistechnik für die Konjunktion, die Disjunktion und die Negation. Die Kontakte a, b werden dabei durch die Relaismagnete A, B betätigt. G ist ein Grundspannungspol.

Die Konjunktion wird durch Hintereinanderschalten, die Disjunktion durch Parallelschalten und die Negation durch Ruhekontakt gelöst. Bild 12 zeigt die Schaltung von Bild 6 als Relaisschaltung. Dabei wird noch von der Möglichkeit Gebrauch gemacht, einen Arbeits- und einen

Abbildung 12:

Ruhekontakt zu einem Umschaltkontakt zu vereinigen. Dadurch ist die Aufgabe der Äquivalenz besonders leicht lösbar. Es wird ein Zwischenrelais D_i verwendet. Seine Elimination ist eine besondere Aufgabe der Schaltalgebra, auf die hier jedoch nicht weiter eingegangen werden soll.

Reiaisschaltungen für logische Aufgaben lassen sich daher sehr einfach durch passende Hintereinander- bzw. Parallelschaltung von Arbeits- und Ruhekontakten aufbauen. Ihre Funktion ist unmittelbar anschaulich. Logische und Verstärker-Funktionen sind in demselben einfachen Bauelement vereinigt. Leider hat das Relais den Nachteil, daß die Magnetspule und die mechanische Bewegung der Anker bzw. der Kontakte der Trägheit unterworfen sind. Man kann im allgemeinen nur 30 Schaltungen pro Sekunde erreichen, während moderne Transistorschaltungen bereits im Nanosekundenbereich arbeiten (eine Nanosekunde = 10^{-9} sec).

Wegen der klaren Übersichtlichkeit der Reiaissehaltungen hatte sich die Schaltalgebra auch zunächst vornehmlich mit Reiaisschaltungen befaßt. Es lassen sich verhältnismäßig leicht die formalen Regeln des Aussagenkalküls auf die entsprechenden Regeln für Schaltungen übertragen. Das gilt natürlich auch für die Schaltungen in Halbleitertechnik, sofern die Elemente für die logischen Grundoperationen klar definiert sind. Prinzipiell erlaubt die Schaltalgebra, Schaltungen rein formal durch logische Formeln zu lösen, und diese dann den Regeln der betreffenden Technik entsprechend auf die Konstruktion zu übertragen. Diesen Weg ist seinerzeit (1937-1945) der Verfasser zusammen mit Dr. Schreyer gegangen, der zunächst lediglich elektronische Schaltungen zur Lösung der logischen Grundoperationen entwarf (entsprechend Bild 7 und 11). Die bereits ausgearbeiteten und

erprobten Relaisschaltungen brauchten dann nur auf dem Wege über die formale logische Darstellung in die elektronische Technik übertragen zu werden.

Nunmehr können einige prinzipielle Vergleiche zwischen Formeln und Schaltungen erfolgen. Wir haben gesehen, daß Strukturen, die in der Verknüpfung von Werten bzw. Variablen durch Operationen bestehen, sowohl als Formeln als auch als Schaltungen dargestellt werden können. Die Regeln einer eineindeutigen Zuordnung zwischen diesen beiden Formen sind leicht darstellbar. Auch das in einzelne Anweisungen aufgelöste Programm (25, 26) ist eine spezielle formale Darstellungsform. Logisch gesehen besteht sie aus einer Folge von viergliedrigen Termen, nämlich der Adressen (Nummern der Speicherzellen) der beiden Operanden, der Operationsart und der Adresse des Resultats.

Ähnliche Verhältnisse finden sich bei der Darstellung von Relationen, die auch als zweistellige Prädikate bezeichnet werden. Sie sind über einer Menge von Elementen definiert, zwischen denen eine gegebene Beziehung besteht. So kann zum Beispiel ein Straßennetz durch die Relation Ort A ist durch eine Straße mit Ort B verbunden oder kurz: Str(A, B) dargestellt werden. Diese Relation kann symmetrisch oder auch gerichtet sein. Bei einem Straßennetz ist die Beziehung normalerweise symmetrisch. Bei Einbahnstraßen liegt eine gerichtete Beziehung vor. Die mathematische Logik benutzt drei Darstellungsarten solcher Relationen: Die Paarliste, die Matrix und die Pfeilfigur. Bild 13a, b, c zeigt die drei Formen für ein Straßennetz mit den Orten A, B, C, D, E.

Abbildung 13:

Dabei kann man die Pfeilfigur als topologische Darstellung mit der Schaltung vergleichen. Sie zeigt, wie die Orte vermittelst der Straßenverbindungen zueinander „geschaltet“ sind. Die Paarliste entspricht dem in einzelne Anweisungen aufgelösten Programm. Für die Matrix wurde bisher keine entsprechende Form für Formeln mit Operationszeichen gezeigt. Das liegt daran, daß die besprochenen formalen Verknüpfungen einmal dreigliedrige Relationen darstellen (z. B. a + b = c) und zum anderen nicht homogen für die ganze Formel gelten (verschiedene Operationen).

Andererseits entspricht keine der in Bild 13a, b, c gezeigten Formen einer als Zeichenfolge geschriebenen Formel. Dies liegt daran. daß Formeln in der Form, wie wir sie kennen, nur baumartige Strukturen zulassen. Eine baumartige Relation wäre ebenfalls als Formel darstellbar, wobei wir lediglich die Zusammenfassungen durch Klammern benötigen, da die Operationszeichen fortfallen. (Es gibt nur eine Verknüpfungsart.) Auf diese Weise lassen sich z. B. kompliziert aufgebaute Informationsstrukturen darstellen, die verschiedene Zusammenfassungen und Verzweigungen aufweisen. Als Beispiel sei die Gleitkommadarstellung einer numerischen Zahl in einem Rechner gewählt. Wir haben zunächst den Exponenten

E und die Mantisse M, die zusammengefaßt den numerischen Betrag ergeben. Durch Hinzuziehen des Vorzeichens V erhält man die dargestellte reelle Zahl. Ein Sonderzeichen S möge noch in besonderen Fällen Ausnahmewerte wie „Null“, „Unendlich“ oder dergleichen bedeuten.

Bild 14 zeigt die verschiedenen Formen, *a* als Blockschema, *b* als Pfeilfigur und c als Formel.

Abbildung 14:

Das Blockschema *a* ist eine andere Form der bildlichen Darstellung, wie sie bei linearer Anordnung einzelner Elemente einer Gesamtinformation sinnvoll ist. Sie kann in direkte Beziehung zu den Registern einer Rechenmaschine gebracht werden.

Man sieht, daß die einzelnen Darstellungsformen verschiedene Vorteile und Mängel haben. Als eine Erweiterung der Pfeilfigur ist der Begriff des „Graphen“ entstanden. Schaltung und Pfeilfigur kann man als spezielle Graphen auffassen. Die Definition des Begriffs „Graph“ ist in der Literatur nicht einheitlich. Im allgemeinen versteht man darunter eine zeichnerische Darstellung, bei der, ähnlich wie bei der Pfeilfigur, Punkte durch Linien, welche als „Kanten“ bezeichnet werden, verbunden sind. Diese Kanten können, ebenso wie bei der Pfeilfigur, gerichtet oder doppelsinnig sein. Im Gegensatz zur Pfeilfigur einer zweistelligen Relation erlaubt der Graph jedoch zusätzliche Angaben, welche den Kanten bzw. Punkten zugeordnet werden können, wodurch eine größere Variabilität erreicht ist. Die Schaltschemen von Bild 1, 2 und 6 können in diesem Sinne als Graphen aufgefaßt werden.

Es gibt also eine Reihe von Darstellungsformen für denselben Sachverhalt, die ineinander übergehen bzw. ineinander überführt werden können.

Die Formel besteht aus einer Zeichenkette. Benutzt man hierfür Morsezeichen, so hat man sogar eine echte lineare Form, gewissermaßen ein eindimensionales Bild. Bei einer Kette von Schriftzeichen stellt das einzelne Zeichen zwar ein zweidimensionales Bild dar, da diese aber hintereeinander angeordnet sind, kann man von einer quasilinearen Darstellung sprechen. Darstellungsformen entsprechend (1) stellen den Übergang zur topologischen Darstellung in zwei Dimensionen dar. Das ist aber auch entsprechend (16) und (17) möglich. Im Grunde genommen, handelt es sich dabei um eine quasi eindimensionale Form.

Auch die Zerlegung einer algorithmischen Formel in eine Kette von Einzelanweisungen und ihre Zusammensetzung zu einem Programm entsprechend (26), (27), (28) ist eine quasi-lineare Darstellung. Erst die Entwicklung eines Graphen entsprechend Bild 1 und 2 führt zur echten zweidimensionalen Form, wobei man allerdings auch diesen Graphen so strecken känn, daß eine quasilineare Form

entsteht:

Abbildung 15:

Mit derartigen Graphen ist der Übergang zu Schaltzeichnungen gegeben. Bei allen Formen handelt es sich um Verknüpfungen, und es ergibt sich die Aufgabe, die zu verknüpfenden Elemente und ihre Verknüpfungen zu kennzeichnen. Dies kann durch Zeichen und durch topologische Anordnung erfolgen. Eine rein formale Darstellung ist ohne die Bezeichnung der Variablen sinnlos. In der Formel (8) treten z. B. die Variablen a und b je zweimal auf. Dagegen wäre beispielsweise Bild 15 in sich auch ohne die Buchstabenbezeichnung der Anschlüsse verständlich.

Eine Schaltzeichnung kann im Prinzip immer so aufgebaut werden, daß interne Bezeichnungen von Anschlüssen überflüssig sind. Das gilt, z. B.,für die Schaltung entsprechend Bild 8. Dagegen sind in der Schaltung von Bild 12 die Bezeichnungen D_i und d_i logisch erforderlich, da sie die Verbindung zwischen dem Relais D_i und seinem Kontakt d_i kennzeichnen. Die Verwendung dieser Bezeichnungen erfolgt jedoch nur aus Gründen der Übersichtlichkeit. Man könnte z. B. die Relaiswicklung direkt neben den Kontakt setzen bzw. die Wirkung der Wicklung auf die Kontakte durch besondere Wirklinien kennzeichnen. Erst dann hat man es mit einer rein schaltungsmäßigen Darstellung zu tun. Dem betreffenden Teil der Schaltung von Bild 12 könnte man dann z. B. folgende Form geben (Bild 16):

Abbildung 16:

Bild 16 ist ohne weiteres verständlich. Allerdings könnten bei kormplizierten Schaltungen die Darstellungen sehr unübersichtlich werden. Schaltzeichnungen werden daher nur aus organisatorischen, nicht aber aus Gründen der logischen Notwendigkeit, in Teilschaltungen aufgeteilt. Die Verbindungen zwischen ihnen müssen dann durch Bezeichnungen gekennzeichnet werden.

Wird dann an Hand eines derartigen Satzes von Schaltzeichnungen das Gerät praktisch gebaut, so verlieren die Bezeichnungen wieder an Bedeutung. Für die Funktion sind sie nicht erforderlich, denn jede Verbindung ist jetzt materiell verwirklicht. Die Bezeichnungen haben lediglich eine Bedeutung für den Bau des Gerätes und für die Fehlersuche. Auch eine ganze Rechenmaschine ist im Grunde genommen eine solche topologische Anordnung und Verknüpfung verschiedener Elemente, deren Teile-Bezeichnungen nicht relevant sind. Sogar die Eingabetasten und die Ausgabelampen brauchten im Grunde genommen keine Bezeichnung, wenn durch ihre topologische Anordnung ihre Bedeutung erkennbar ist.

Der Auflösung einer algorithmischen Formel in Einzelanweisungen entspricht die Aufstellung einer Verdrahtungsliste für die Fertigung eines elektrischen Gerätes.

Hier werden Verbindungslinien durch Bezeichnungen ersetzt. Dieser Vergleich eines Programms mit einer Verdrahtungsanweisung wird unmittelbar anschaulich, wenn man die bei Lochkartengeräten üblichen Steckpanne aus betrachtet; sie können in gewisser Hinsicht als Vorläufer der Programmsteuerung angesehen werden. Die Verbindungsmöglichkeiten zwischen einzelnen Teilen des Gerätes (Lochkartenabfühlstation, Zähler, Register usw.) können dabei der Aufgabe entsprechend variabel über ein Steckpanneau hergestellt werden. Derartige gesteckte Programme haben auch bei analogen Geräten eine große Bedeutung, da ja bei ihnen nicht eine Befehlsfolge nacheinander abgearbeitet, sondern der gegebenen Formel entsprechend eine Schaltung aufgebaut wird.

Schaltnetze und Schaltwerke

Man spricht von Schaltnetzen, wenn bei ihnen der zeitliche Ablauf von untergeordneter Bedeutung ist. Die besprochenen Schaltungen stellen in diesem Sinne Schaltnetze dar, auch wenn zu ihrer Funktion das zeitliche Nacheinanderwirken einzelner Glieder z. B. der Relais, erforderlich ist. Wesentlich ist, daß jedes Element im Verlaufe der zu lösenden Aufgabe nur einmal in Funktion tritt. Man nennt derartige Schaltnetze auch Zuordner. Sie haben eine Reihe von Eingängen und eine Reihe von Ausgängen. Jeder Eingabekombination wird eine Ausgabekombination zugeordnet.

Fast jede umfangreiche Rechnung enthält jedoch Teilaufgaben, die mehrmals hintereinander mit verschiedenen Eingabe- und Ausgabewerten durchgerechnet werden. Baut man die Multiplikation aus wiederholter Addition auf, so wird man im allgemeinen nur ein Addierwerk verwenden und dieses mehrmals hintereinander benutzen. Aber auch die Addition selbst ist ein zyklischer Prozeß, der sich von Stelle zu Stelle wiederholt. Mechanische Rechner und Relaisrechner haben fast durchweg parallele Addierwerke, bei denen jeder Stelle eigene Addierglieder zugeordnet sind. Elektronische Geräte arbeiten dagegen wie der Mensch im allgemeinen nach dem Serienprinzip, wobei die einzelnen Ziffern einer Zahl nacheinander über die gleiche Leitung in das Addierwerk geleitet werden. Die Summe entsteht dann ebenfalls nacheinander Ziffer für Ziffer. Wegen des Stellenübertrages muß bei diesen Durchläufen mit der untersten Stelle begonnen werden. Bei Bild 17 handelt es sich um ein 6stelliges Paralleladdierwerk mit 6 Eingängen $a_5 \ldots a_0$.

Abbildung 17:

Die Summe wird auf einen Zwischenspeicher geleitet und von dort nach einer zeitlichen Verzögerung wieder auf das Addierwerk gegeben. Das Ganze hat dann die Funktion eines Registers, in das man immer wieder neue Zahlen hereinaddieren kann, wobei lediglich zu beachten ist, daß der Stellenbereich nicht überschritten

wird, was durch eine Leitung U, (Stellenübertrag auf die nicht eingebaute Stelle 6) gemeldet wird.

Bild 18 zeigt die gleiche Einrichtung, nach dem Serienprinzip gebaut. Man sieht sofort, daß der schaltungsmäßige Aufwand erheblich geringer ist. Das wirkt sich erst recht aus, wenn man, wie üblich, mit Wortlängen von 40 bis 60 Bit arbeitet. Anstelle des Speichers S tritt jetzt eine Verzögerungslinie V, welche die einzelnen Ziffern hintereinander durchlaufen. Die Verzögerung beträgt mindestens eine volle Wortzeit, das ist die Zeit, die eine Zahl (ein Wort) braucht, um mit allen Ziffern durch das Addierwerk zu laufen.

Abbildung 18:

Fast alle praktisch ausgeführten Schaltungen bestehen aus derartigen Schaltwerken. Die Zuordnung zwischen Schaltung und Formel ist dabei immer nur für die durch die Schaltung repräsentierten Teilaufgaben möglich. Besonders typisch ist das für zyklische Programme, die zunächst besprochen seien, um das Verständnis zu erleichtern.

Als einfaches Beispiel sei die Summenbildung von n Summanden besprochen: Ist n gegeben und nicht zu groß, so läßt sich die Rechlenanwei- sung voll ausschreiben. Beispiel: n = 4

$$S := a_l + a_2 + a_3 + a_4 \tag{35}$$

Dem entspricht als Schaltung folgendes Schema mit drei Addiergliedern:

Abbildung 19:

Nun sind aber gerade Anweisungen für Summenbildung mit beliebig vielen Gliedem besonders wichtig. Die Mathematiker kennen hierfür schon lange das Summenzeichen:

$$S := \sum_{i=1}^{n} a_i.$$

Mitunter wird dafür auch folgende Form gewählt:

$$a_1 \ldots a_i + \ldots a_n = S \tag{36}$$

Es ist nicht ohne weiteres möglich, dafür mit normalen formalen Mitteln eine Anweisung für Rechenautomaten abzuleiten. Wir nehmen daher zunächst die Wortsprache zu Hilfe.

Die Anweisung lautet dann:

> Nimm die ersten beiden Summanden, addiere sie und addiere zu dieser Teilsumme den nächsten Summanden. Fahre in diesem Sinne fort, bis alle Summanden erfaßt sind. Die letzte Summe ist gleich der gesuchten Gesamtsumme.

Bei dieser Formulierung tritt der Index i nicht auf. Es wird jedoch vorausgesetzt, daß die Feststellung, daß es sich um den letzten Summanden handelt, möglich ist. Bei der Aufstellung einer Liste von Summanden kann das durch ein Abschlußzeichen oder anderweitig erfolgen, wobei auch eine Folge von Zwischenraumzeichen die Rolle des Abschlußzeichens übernehmen kann. Damit stoßen wir schon auf eine Problematik der Datenverarbeitung: Diese organisatorischen Informationen sind mit Bestandteil der Rechnung. Es kann jedoch auch n von vornherein bekannt sein. Dann lautet die Vorschrift zur Summenbildung genau genommen wie folgt:

> Bilde einen laufenden Index i. Setze ihn zunächst gleich 1. Suche aus der Liste der gegebenen Summanden denjenigen mit dem Index i, addiere ihn zur bereits gebildeten Teilsumme, erhöhe i um eins, suche wiederum den Summanden mit dem Index, addiere ihn ... usw. Sobald i = n ist, brich den Prozeß ab.

Wir haben hier einen zyklischen Vorgang, der n-mal hintereinander abläuft. Außer der möglichen Summenbildung läuft eine „lndexrechnung“. Wichtig ist jedoch, daß außer diesem zyklischen Teil der Anweisung eine Startanweisung gegeben wird. Sie enthält einmal die Anweisung, mit i = 1 zu beginnen. In obiger Anweisung ist aber noch stillschweigend vorausgesetzt, daß beim Erfassen des ersten Summanden „die bereits gebildete Teilsumme“ gleich Null ist. Diese für den menschlichen Rechner selbstverständliche Voraussetzung muß beim maschinellen Rechnen gesondert formuliert werden. Die Vorschrift lautet dann:

Startteil:	„setze i = 1“
	„setze S = 0“
zyklischer Teil:	„addiere a_i zum bisherigen S, dies gibt das neue S.“
	Erhöhe i um eins.
	Ist i kleiner oder gleich n, so wiederhole den Zyklus, andernfalls brich die Rechnung ab. S ist die gesuchte Gesamtsumme.

(37)

Rein formal ist dieses Programm wie folgt darstellbar:

$$
\begin{array}{ll}
 & i := 1 \\
 & S := 0 \\
\text{Zyklus:} & S := S + a_i \\
 & i := i + 1 \\
 & i \leq n \Rightarrow \text{ wiederhole Zyklus}
\end{array}
\tag{38}
$$

Man braucht also zur Ergänzung des traditionellen mathematischen Formalisinus ein Zeichen für die Anweisung „wiederhole Zyklus“. Dafür gibt es verschiedene Möglichkeiten. Eingeführte algorithmische Sprachen lehnen sich hierbei an die englische Sprache an und verwenden den „Goto“ Befehl. Zum Beispiel in der Form:

$$\text{IF } i \leq n \text{ go to Zyklus.}$$

Im Beispiel (38) werden im Gegensatz zur Form (35) nicht n – 1, sondern n Additionen ausgeführt wobei die erste Addition eine Scheinaddition ist.

Dadurch vereinfacht sich jedoch das Programm. Denn sonst müßte für die erste Addition eine nur einmal zu durchlaufende Startanweisung gegeben werden. Außerdem kommt noch ein Punkt hinzu: Man würde voraussetzen, daß mindestens drei Summanden vorhanden sind, nämlich zwei für die erste befohlene Addition und mindestens einer für den zyklischen Teil. Die Form (38) funktioniert jedoch auch, wenn nur ein einziger Summand gegeben ist, also für n = 1. Auch das wird in vielen Fällen noch nicht genügen, denn es könnte ja sein, daß die Menge der Summanden leer ist, d. h. daß n = 0 ist. Das Programm (38) würde dann vergeblich im ersten Zyklus den Summanden a_1 suchen. Hier zeigt sich der Unterschied zwischen der Maschine und dem menschlichen Rechner. Die Maschine wäre in dieser Situation hilflos, da sie keine Anweisung hat, was zu tun ist. Man kann diesen Umstand nun durch einen kleinen Trick berücksichtigen: Wir setzen den Vergleich zwischen n und i an den Anfang des zyklischen Teils und führen das bedingte Schlußzeichen „FIN“ ein:

$$\begin{array}{ll} & i := 1 \\ & S := 0 \\ \text{Zyklus:} & n < i \Rightarrow FIN \\ & S := S + a_i \\ & i := i + 1 \\ & \text{Go-to-Zyklus} \end{array} \tag{39}$$

Für den Fall, daß n = 0 ist, wird der zyklische Teil gar nicht erst durchlaufen, sondern es wird sofort das Schlußzeichen gegeben. S ist dann vorschriftsmäßig = 0.

An diesem bewußt sehr elementar gewählten Beispiel zeigt sich bereits, welche Problematik das Programmieren bietet. Gerade diese nicht zur eigentlichen Rechnung gehörenden Nebenoperationen entscheiden über den einwandfreien Ablauf eines Programms.

Goldstine und John von Neumann haben eine sehr anschauliche Darstellungsform gefunden, um die oft recht komplizierten Abläufe und ihre Verschachtelung darzustellen. Die Flußdiagramme für die Programme (38), (39) sehen wie folgt aus (Bild 20, 21).

Zyklische Teile sind dabei durch Führungslinien gekennzeichnet, die vom Ausgang des zyklischen Teiles zu seinem Eingang laufen. Entscheidungen, welche im allgemeinen von Ungleichungen abhängen, werden durch sechseckig ausgeführte Kästchen angedeutet, die zwei Ausgänge für „ja“ und „nein“ haben.

Die Möglichkeit, durch bedingte Befehle beliebige Verschachtelungen eines Programms vorzusehen, gibt den Abläufen eine außerordentlich große Beweglichkeit. Darin liegt jedoch auch eine große Gefahr. Nicht immer sind die Konsequenzen solcher vielfach verflochtenen Abläufe klar zu überblicken. Wir sahen bereits an dem einfachen Beispiel der Addition, welche Rolle Kleinigkeiten spielen können. Ihre Nichtbeachtung kann dazu führen, daß Zyklen endlos durchlaufen werden, oder was noch schlimmer ist, daß Verzweigungen auftreten, die vom Programmierer nicht beabsichtigt sind. Der Go-to-Befehl gilt deshalb auch mit Recht als einer der gefährlichsten Befehle, und es gibt kritische Stimmen, die behaupten, daß die Zahl der Fehler in einem größeren Programm etwa proportional der Zahl der verwendeten Go-to-Befehle ist.

Abbildung 20:

Abbildung 21:

Äußerlich haben Flußdiagramme eine gewisse Ähnlichkeit mit Schaltungen. Eine Analogie läßt sich jedoch erst durch Einführung der Operation Streckung und Faltung durchführen.

Nun zurück zu Bild 19, das als Schaltung für die Bildung einer Summe von 4 Werten aufgefaßt werden kann. Bei einer tatsächlich ausgeführten Schaltung wird man aber sicher nicht mehrere Addierwerke, sondern eine Anordnung entsprechend Bild 17, 18 verwenden. Man kann nun Bild 19 alsSS-treckung“ von Bild 18 auffassen. Um das zu veranschaulichen, seien die einzelnen Addierwerke von Bild 19 in axonometrischer Darstellung übereinander gezeichnet, wie es Bild 22a zeigt. Durch Projektion aller Addierwerke in der senkrechten Richtung erhält man dann durch Überlagerung bzw. „Faltung“ den Aufbau eines zyklisch arbeitenden Rechners (Bild 22b). Dasselbe Verfahren läßt sich ohne weiteres auf ein Flußdiagramm anwenden. Man kann dieses „strecken“, wobei die so erhaltene Anordnung einen speziellen Fall der Durchrechnung für eine bestimmte Aufgabe darstellt, also den tatsächlichen Ablauf für eine bestimmte Realisierung des durch das Flußdiagramm gegebenen Programms. Bild 23a, b zeigt das Beispiel für Flußdiagramm von Bild 21 für den Fall n = 2.

Abbildung 22:

Abbildung 23:

Nunmehr kann man von Schaltwerken zu programmgesteuerten Rechengeräten übergehen. Betrachten wir die bisher entwickelten Programme, so leuchtet es ein, daß eine nach dem Adressenprinzip organisierte Maschine am günstigsten zu verwenden ist. Das Programm wird entsprechend (26) in einzelne Anweisungen aufgelöst, und die auftretenden Werte werden entsprechend (27) fortlaufend numeriert. Dies entspricht der von Babbage gewählten Form. Die auftretenden Variablen sind bei numerischen Rechnungen alle von gleicher Struktur, nämlich reelle Zahlen, die allerdings auf sehr verschiedene Weise dargestellt werden können (Dezimalzahlen, Binärzahlen, Gleitkomma usw.). Das Schema der Babbage-Maschine zeigt Bild 24.

Abbildung 24:

Das Gerät besteht im wesentlichen aus drei Teilen: Dem zentralen Rechenwerk zur Ausführung der arithmetischen Operationen, dem Speicher zur Aufnahme der Zahlen und dem Programmwerk. Das Programm selbst wurde bei Babbage auf Lochkarten festgehalten. Er unterschied zwischen Operations- und Variablenkarten. Die Operationskarten enthielten 4 Lochfelder, von denen jeweils eins gelocht wurde, die Variablenkarten dagegen enthielten 1000 Lochfelder, von denen auch jeweils nur eins gelocht wurde. Babbage kannte also noch nicht die Vercodung von Rechenoperation und Adressen. Heute würde man von einem 1-aus-4-Code für die Operationen und von einem 1-aus-1000-Code für ‚die Variablen sprechen. Das ergab sich aus der Übertragung des Steuerungsprinzips der Jacquard-Webstühle auf die Rechenmaschine. Dem entsprechend bestanden die Auswahleinrichtungen bei Babbage auch aus direkt zugeordneten Fühlstiften zu den einzelnen Speicherzellen. Dieses Prinzip ist heute längst überwunden und wäre bei der Speicherkapazität der heutigen Geräte nicht mehr verwendbar.

Das erste Gerät mit vercodeten Befehlen sowohl für Operationen als auch für Adressen war das vom Verfasser gebaute Gerät Z 3 (1941). Es hatte einen aus 8 Bit bestehenden Befehlscode. Die ersten beiden Bit ‚gaben an, ob es sich um einen Rechenbefehl, um einen Lesebefehl aus dem Speicher oder einen Speicherbefehl handelt. Die weiteren 6 Bit dienten der Vercodung der Rechenoperation und der Adressen (64 Speicherzellen). Diese vercodeten.Befehle ergeben eine wesentlich konzentriertere Form der Befehlsträger(zunächst Lochstreifen). Dem Speicher wird dabei eine Auswahleinrichtung votgeschaltet, eine sogenannte Wahlpyramide, welche die vercodete Adresse in den 1-aus-n-Code übersetzt, Bild 25.

Abbildung 25:

Mit einer derartigen Einrichtung ist es möglich, gestreckte Programme also solche ohne Variationsmöglichkeit durch bedingte Befehle, durchzurechnen. Man kann nun nach dem gleichen Schema programmgesteuerte Rechengeräte bauen, die nicht numerisch rechnen. Auf Seite 264 wurde bereits gezeigt, daß, sich die Addition im binären Zahlensystem in Operationen mit einzelnen Bits auflösen läßt (32). Anweisung (32) kann man, wie leicht einzusehen ist, auch als in einzelne Operationen aufgelöstes Programm entsprechend (27), (28) darstellen. Ein entsprechendes programmgesteuertes Rechenwerk würde das Schema von Bild 25 ergeben, jedoch mit dem Unterschied, daß das Rechenwerk R lediglich die drei Grundoperationen des Aussagenkalküls durchzuführen hätte. Wir erhalten also ein denkbar einfaches Rechenwerk. Da sich alle Rechenoperationen in solche zwischen Bits auflösen lassen, wäre eine derartige Form im Prinzip in der Lage, nicht nur numerische Programme durchzurechnen, sondern generell alle gestreckten Programme der allgemeinen Informationsverarbeitung. Der Gedanke ist zunächst bestechend. Ein derartig logisches Rechenwerk hat der Verfasser während des Krieges einmal zu Studienzwecken gebaut. Für den praktischen Einsatz kommt es jedoch kaum in Frage. Der minimale Aufwand für das Rechenwerk wird durch erheblich längere Programme – eine einzelne Multiplikation erfordert bereits Tausende von Befehlen – und umfangreichere Auswahleinrichtungen erkauft.

Zwischen diesen beiden Extremen, arithmetisch voll ausgebaute Rechenwerke einerseits und Rechenwerke für Ja-Nein-Werte andererseits, lassen sich nun beliebige Zwischenstufen einschalten, und das gilt für die meisten heute auf dem Markt befindlichen Geräte. Als Speichereinheit wählt man dabei das Wort, welches im allgemeinen eine komplette reelle Zahl repräsentieren kann, also etwa 40 – 60@ Bit enthält. Das Rechenwerk ist jedoch nicht voll ausgebaut. Es kann addieren, subtrahieren, linksverschieben, rechtsverschieben, das ist zum Aufbau der arithmetischen Operationen Multiplikation und Division erforderlich. Eine wichtige Elementaroperation ist noch die Intersektion ($\barwedge$) Darunter versteht man die paarweise Bildung der Konjunktion zwischen je zwei Ziffern der gleichen Stelle zweier gegebener Bit-Ketten (Wörter) a und b. Als Ergebnis erhält man eine Bit-Kette c.

$$\begin{array}{cc} a & \boxed{LLOLLO} \\ b & \boxed{OLLLOO} \\ c & \boxed{OLOLOO} \end{array} \tag{40}$$

Neuerdings setzen sich auch mehr und mehr sogenannte Kurzwortmaschinen durch. Sie haben im allgemeinen eine Wortlänge von 12 bis 24 Bit. Numerische Werte können also entweder nur mit geringer Genauigkeit dargestellt werden oder müssen auf mehrere Wörter verteilt werden. Man wählt zum Teil auch noch kleinere Informationseinheiten, wie z. B. das Byte, eine Kette von 8 Bits oder eine Kette von 4 bzw. 5 Bits, womit einzelne alphanumerische Zeichen (Dezimalziffern,

Buchstaben) getrennt behandelt werden können. Eine zwölfstellige Dezimalzahl muß bei der letzten Form natürlich auf mindestens 12 Zeichen verteilt werden. Dies erlaubt auch die Möglichkeit, Geräte mit variablen Wortlängen zu bauen. Bei Aufruf einer Adresse wird zunächst das unter dieser Adresse gespeicherte Zeichen herausgegeben und anschließend alle folgenden, bis ein Schlußsignal erfolgt.

Die verschiedenen Möglichkeiten seien noch einmal zusammengestellt.

Typ	Informationseinheit	Rechenoperationen
voll ausgebaute Arithmetik	Langwort (40 – 60 Bit)	$+, -, \times, :, \sqrt{\dots}$
teilweise ausgebaute Arithmetik	Langwort	$+, -, \leftarrow, \rightarrow, \twoheadrightarrow$
Kurzwortmaschine	Kurzwort (12 – 24 Bit)	$+, -, \leftarrow, \rightarrow, \twoheadrightarrow$
zeichenorganisierte Geräte	Byte (8 Bit) Tetrade (4 Bit)	$+, -,$
logisches Gerät	Bit	$\wedge, \vee, \neg$

Alle modernen Maschinen, auch die Langwortgeräte, ermöglichen es, mit beliebig erhöhter Stellenzahl zu rechnen, wobei eine Variable auf mehrere Worte verteilt wird.

Die Variabilität der Programme wird durch bedingte Befehle und Adressenumrechnung erreicht. In diesem Zusammenhang sei zunächst die Turing-Maschine kurz erwähnt.

Turing schuf sein theoretisches Modell einer Rechenmaschine, um den Begriff der Berechenbarkeit einer Funktion auf mathematischlogische Ableitungen anzuwenden. Er hatte keinerlei Ambitionen, eine praktisch einsetzbare Rechenmaschine zu konstruieren. Seine Maschine ist wohl kaum je in reiner Form gebaut worden. Sie besteht im wesentlichen aus einem Kasten K mit einer Schreib- und Leseeinrichtung für ein als Informationsträger dienendes Band B, das relativ zum Kasten K in beiden Richtungen verstellbar ist und prinzipiell von unendlicher Länge gedacht ist (Bild 26). Im Extremfall enthält das Band nur eine Kette von Bits.

Abbildung 26:

Die Maschine kann folgende Operationen ausführen: Lesen des gerade in der Schreib- und Lesestation befindlichen Zeichens, Löschen dieses Zeichens, Schreiben eines Zeichens, Linksverschieben und Rechtsverschieben des Bandes. Im Gegensatz zu den oben besprochenen Geräten arbeitet es also nicht mit der Einteilupg in Programmwerk, Rechenwerk und Speicherwerk mit Rechen- und Adressenorganisation. Das Band stellt einen unendlichen Speicher dar. Der Kasten ist so organisiert, daß er mehrere Zustände einnehmen kann. Dabei ist der Folgezustand jeweils eine Funktion des gegebenen Zustandes und des gerade gelesenen

Zeichens. Dieses Prinzip, mit Zuständen zu arbeiten, ist dann später auch von der Automatentheorie übernommen worden, um allgemein das Verhalten beispielsweise von Rechengeräten zu beschreiben. Rechengeräte sind „finite“ „sequentielle“ Automaten, d. h. ihr Verhalten ist deterministisch, die Anzahl der Zustände, die sie einnehmen können, ist begrenzt, und sie arbeiten mit einer zeitlichen Folge diskreter Zustände. Der logisch sehr einfache Aufbau der Turing-Maschine muß allerdings durch sehr komplizierte Programme erkauft werden, die als Zeichenketten auf das Band aufgebracht werden. Das Gerät braucht eine bestimmte Startinformation, um überhaupt arbeiten zu können. Das Aufsuchen einer gesuchten Information muß jedoch stets durch eine Folge von einfachen Verschiebungsbefehlen erfolgen, das ist im Vergleich zum Adressenprinzip sehr umständlich. Vom Standpunkt des mathematischen Logikers gilt ein Problem aber als gelöst, wenn es, überhaupt einen Weg gibt, um in endlich vielen Schritten zum Ziel zu kommen. Da es sich bei all diesen Betrachtungen um rein theoretische Erwägungen handelt, stört es den Mathematiker nicht, wenn die Zahl der Durchläufe außerordentlich hoch ist, so daß die praktische Durchführung der Rechnung sinnlos wird.

Die Turing-Maschine hat jedoch den Vorteil, daß sie in ihrer Organisation so aufgebaut ist, daß besondere bedingte Befehle nicht als solche in Erscheinung treten, da eine strenge Befehlsorganisation überhaupt nicht vorhanden ist. Bedingungen, wie z. B. $n < i$ des Flußdiagramms Bild 21 erscheinen als Ja-Nein-Wert und bewirken aufgrund der Zustandstabelle des Automaten die Fortsetzung des bisherigen Zyklus oder den Abbruch der Rechnung. Das gibt der Turing-Maschine, ihre Universalität sie ist jeder überhaupt berechenbaren Funktion gewachsen.

In der Praxis haben sich streng auf dem Adressen- und Befehlsprinzip aufgebaute Geräte durchgesetzt. Wir wollen daher noch einmal zu Bild 25 zurückkehren (Gerät Z 3) und von dort aus die geschichtliche Weiterentwicklung verfolgen. Bild 27 gibt einen Überblick über die einzelnen Phasen. Selbst so einfache Programme wie (39) können auf diesem Gerät nicht vollautomatisch gerechnet werden. Man kann sich so behelfen, daß man zunächst einen „Vorspann“ eingibt, durch welchen die Register i und S auf null gesetzt und die Summanden in den Speicher gegeben werden. Anschließend kann die zyklische Durchrechnung, gesteuert durch einen als Schleife ausgebildeten Lochstreifen, erfolgen, wobei allerdings keine automatische Abbruchmöglichkeit gegeben ist.

Das erste, was wir brauchen, ist daher ein Stopzeichen, das vom Rechenwerk her das Programmwerk stillsetzt. Dieser eine Draht, der rückwirkend vom Rechenwerk her das Programmwerk beeinflußt, ist nun der Schlüssel, um Programme, die in Form von Flußdiagrammen gegeben sind, durchzurechnen. Die Verzweigungen der Flußdiagramme können, jedenfalls bei numerischen Rechnungen, fast immer auf die Form einer Ungleichung gebracht werden. Der Wahrheitswert dieser Ungleichung gibt dann an, ob ein bestimmter Befehl durchgeführt wird oder

nicht. Go-To-Befehie enthalten also eine Bedingung und ein „Rufzeichen“ für den Start eines Programmes bzw. Programmteiles. Konstruktiv wurde das bei den ersten Geräten durch Einbau mehrerer Abtaster mit verschiedenen Programmteilen oder durch Unterbringen mehrerer Programmteile auf dem gleichen Lochstreifen

Abbildung 27:

durchgeführt. Die konstruktiven Nachteile dieses Verfahrens zeigten sich bald. Man hatte inzwischen längst erkannt, daß sich auch Befehle aus Ketten von Ja-Nein-Werten aufbauen lassen. Sie können daher ebenso gespeichert werden wie Zahlen oder andere Informationen. Damit laufen alle Vorgänge, wie verschiedene Ablaufvariationen, zyklische Wiederholungen, bedingte Sprünge usw., innerhalb des Programmwerkes und des Programmspeichers ab. Das hat nicht nur konstruktive Vorteile, sondern die Beweglichkeit des Gerätes wird erheblich gesteigert, da die Umschaltungen jetzt fast zeitlos erfolgen und somit technische Grenzen für den Grad der Verschachtelung kaum gegeben sind.

Zunächst wurde dieser Programmspeicher getrennt neben den Zahlenspeicher eingebaut. Man erkannte jedoch bald, daß es nicht erforderlich ist, da beide Speicher konstruktiv gleich aufgebaut werden können und damit die Auswahleinrichtungen nur einmal eingebaut zu werden brauchen. Damit ist aber nicht nur ein konstruktiver Vorteil gewonnen. Die Vereinigung beider Speicher bedeutet einen wesentlichen Schritt in Richtung allgemeiner Informationsverarbeitung. Um die volle Beweglichkeit der heutigen Geräte zu erreichen, war noch ein weiterer wesentlicher Schritt erforderlich: Die Adressenumrechnung. Das Programm (39) gibt die Anweisung

$$S := S + a_i.$$

Es muß hier einmal der Wert S aufgerufen werden, der schon im Register stehen kann, und zum anderen der Wert a_i. Die Anweisung enthält also die variable Adresse i. Man muß also die Möglichkeit haben, Adressen umzurechnen. Zu diesem Zweck wurden dem Programmwerk zunächst sogenannte „Index-Register“ zugeordnet, in denen Adressen fortlaufend weitergezählt werden konnten. Die weitere Entwicklung zeigt dann, daß man derartige Adressenumrechnungen über das normale Rechenwerk laufen lassen kann, wodurch nicht nur einfache Zählungen, sondern beliebig komplizierte Adressenumrechnungen möglich werden. Damit ist aber nur der eigentliche rechnende Teil eines Computers erfaßt. Hinzu kommen Ein- und Ausgabeeinrichtungen, die einen erheblichen Aufwand erfordern. Es gehört zur Problematik der Rechengeräteentwicklung, daß diese „Peripherie-Geräte“ mitunter einen erheblich größeren Umfang annehmen als das eigentliche Rechengerät einschließlich Speicher. Neben den großartigen Leistungen der Elektronik darf man nicht vergessen, daß die Peripheriegeräte fast alle noch weitgehend mechanisch arbeiten. Es mußten wahre Wunderwerke der Feinwerktechnik

und Feinmechanik geschaffen werden, die einigermaßen mit der Geschwindigkeit der elektronischen Rechenwerke Schritt halten konnten.

Diese Fragen sollen jedoch nicht das Thema dieser Abhandlung sein. Verfolgt man die Entwicklung des logischen Aufbaus der Rechengeräte weiter, so kann man feststellen, daß der Unterschied zwischen Programmwerk und Rechenwerk immer mehr verschwindet und beide Teile ineinander übergehen. Es ergibt sich dann ein informationsverarbeitender und ein informationsspeichernder Teil. Befehle, Zahlen und sonstige lnformationen laufen über dieselben Register. Man kommt immer mehr zur allgemeinen Informationsverarbeitung.

Ist nun die Trennung zwischen informationsverarbeitenden und informationsspeicherndem Teil logisch notwendig? Sie erfolgt in erster Linie aus ökonomischen Gründen. Bauelemente, die nur speichern, sind erheblich, eventuell um Zehnerpotenzen, billiger als solche, die Informationen verarbeiten. Mit der modernen integrierten Schaltkreistechnik gleichen sich diese Unterschiede aber immer mehr einander an. Die Frage ist daher berechtigt, ob wir im Zuge der Entwicklung dahin kommen, daß man in Zukunft vielleicht nur noch informationsverarbeitende Teile in den Geräten hat.

Ein Blick auf biologische Systeme zeigt, daß dies durchaus sinnvoll sein kann. Das menschliche Gehirn enthält schätzungsweise 10 Milliarden Ganglienzellen, die sehr kompliziert geschaltet sind. Eine strenge Trennung zwischen Speicherung und Verarbeitung läßt sich nicht feststellen. Es sprechen im Gegenteil alle Anzeichen dafür, daß in allen Teilen des Gehirns Informationen sowohl gespeichert als auch verarbeitet werden. Damit haben wir einen wesentlichen Unterschied gegenüber der Rechenmaschine gefunden. Hinzu kommt die besondere Organisation des Speicherinhalts in Form eines assoziativen Gedächtnisses. Die bisher besprochenen Geräte arbeiten mit Adressen; um eine Information aufzusuchen. Es muß vorher feststehen, in weicher Speicherzelle die benötigte Information steht. Die Adresse kann zwar nach gegebenen Anweisungen beliebig umgerechnet und abgewandelt werden, jedoch steht sie als solche in keinerlei Beziehung zum Inhalt der gesuchten Information selbst. Im Gegensatz dazu wird bei einem assoziativen Speicher ein Stichwort aufgerufen, und es meldet sich diejenige Stelle des Speichers, in der dieses Stichwort registriert ist. Die betreffende Speicherzelte spricht also durch Aufrufen ihres Inhaltes an. Das ist aber ein Informationsverarbeitungsprozeß, denn es ist erforderlich, daß ständig in sämtlichen Speicherzellen der Vergleich mit dem aufgerufenen Stichwort und dem Inhalt der Zellen durchgeführt wird [XIII 3].

Durch dieses Prinzip ist es dem Gehirn möglich, Assoziationen durchzuführen, wodurch die logische Beweglichkeit der Informationsverarbeitung des Menschen gegenüber der Rechenmaschine ein wesentlich höheres Niveau hat. Hinzu kommt noch der Vorteil der Quantität. Die im Vergleich selbst zu den größten Rechenanlagen ungeheuer hohe Anzahl von Schaltelementen gibt dem Menschen eine

erhebliche Überlegenheit gegenüber der Maschine bei allen Aufgaben, bei denen es auf Kombinationsfähigkeit innerhalb eines umfangreichen Gedächtnisinhalts ankommt.

Es ist heute noch nicht abzusehen, ob, wann und in welchem Umfang die Ingenieure in der Lage sein werden, uns Rechengeräte ähnlicher Leistungsfähigkeit zu liefern. Die Vorarbeiten sind im Gange. Die integrierte Schaltkreistechnik macht von Jahr zu Jahr größere Fortschritte. Dadurch können auf kleinem Raum informationsverarbeitende Elemente konzentriert werden. Die Hersteltungsverfahren werden laufend verbessert, und die Produkte werden immer preiswerter. Die Kosten müßten allerdings noch um Größenordnungen sinken, wollten wir, im Wettbewerb mit dem menschlichen Gehirn, Geräte mit genügend großen assoziativen Speichern bauen.

Beim assoziativen Speicher hat man es in gewissem Sinne mit einer Auflösung des Adressenprinzips zu tun. Allerdings läßt sich ein solcher Speicher durch einen adressierbaren Speicher simulieren, indem man systematisch alle Speicherzellen nach dem gegebenen Stichwort abfragt, das ist jedoch sehr zeitraubend.

Eine Zwischenlösung zwischen einem rein assoziativen Speicher und einem adressenorientierten Speicher stellt das „Blattprinzip“ dar (page organization). Hierbei werden große Informationseinheiten, z. B. Blöcke von mehreren hundert Worten unter einem frei wählbaren Rufzeichen zusammengefaßt und können“somit als ganze Blöcke assoziativ aufgerufen werden.

Ein anderer Kompromiß mit den technischen Gegebenheiten hat unter dem Namen „Listenprogrammierung“ (list processing) Bedeutung erlangt. Man arbeitet dabei mit einer Kette von Informationseinheiten, die außer ihrem eigentlichen Inhalt Angaben über Adresse der vorhergehenden bzw. der folgenden Informationseinheit enthalten. Man ist jetzt in der Lage, die Glieder von Listen mit veränderlicher Gliederzahl unter beliebigen, gerade freien Adressen eines Speichers abzusetzen. Die Ordnung der Kettenglieder innerhalb des Speichers entspricht also nicht ihrer logischen Ordnung, jedoch können von einem gegebenen Glied stets sofort die Nachbarglieder der Kette gefunden werden.

Der echte assoziative Speicher arbeitet mit einer Mannigfaltigkeit von Informationen, die man sich beliebig vieldimensional angeordnet denken kann.

Bei der bisherigen Betrachtung wurde immer angenommen, daß in einem solchen Informationsfeld zwar laufend in allen Teilen nach aufgerufenen Stichworten gesucht wird, jedoch die aktuelle Information sich immer nur an einer diskreten Stelle befindet und somit die zu verarbeitende Information auch lokal begrenzt ist. Man kann das Bild der feldartig angeordneten Information aber auch in dem Sinne verwenden, daß ständig in allen Teilen eine nach dem gleichen Programm ablaufende Informationsverarbeitung durchgeführt wird. Die extreme Konsequenz dieser Idee ist der zellulare Automat. Er besteht,beispielsweise bei zweidimensio-

naler Ausführung aus einem Gitternetz, bei dem jedem Gitterpunkt ein eigenes Rechengerät zugeordnet ist.

Derartige Anordnungen eignen sich gut zur Lösung partieller Differentialgleichungen. Allerdings sind sie sehr aufwendig. Zellulare Automaten können darüber hinaus aber auch als Modelle für theoretische Betrachtungen verschiedener Art dienen. Der Verfasser hat in seiner Arbeit „Rechnender Raum“ [XIII 1] versucht, automatentheoretische Überlegungen auf die Physik anzuwenden. Das Gebiet ist allerdings noch viel zu unerschlossen, als daß heute schon konkrete Ergebnisse zu erwarten wären.

Bei der bisher besprochenen Form des bedingten Befehls wurde durch einen Ja-Nein-Wert bestimmt, ob ein Befehl äusgeführt werden soll oder nicht. Im letzten Fall wird er übergangen und der nächste Befehl herangezogen. Durch mehrmaliges Hintereinandersetzen solcher bedingter Befehle, lassen sich darin auch komplizierte Verzweigungen programmieren.

Logisch leistungsfähige Geräte kennen aber im allgemeinen nicht nur die einfache Ausführungsbedingung, sondern darüber hinaus die Befehlsmodulation. Es sind dabei im Befehlswort mehrere Bits vorgesehen, durch die dann die Art der Ausführung des Befehls verschieden gestaltet werden kann. Im Extremfall haben wir keine eigentliche Befehlsverordnung, sondern ein Rechengerät mit steuerbarer Schaltung. Man spricht dann von einem analytischen Code. Beispiele für derartige Geräte sind die aus dem „Minima“-Konzept von Fromme hervorgegangenen Geräte Z 22 und Z 23. Das Gerät enthält eine Reihe von Übertragungsteitungen zwischen einzelnen Registern usw., die durch sogenannte „Tore“ eingeschaltet werden können. Diesen Toren sind nun Befehlsbits im Befehlswort zugeordnet. Die logische Beweglichkeit solcher Geräte ist außerordentlich groß, da jeder Programmierer sich seine eigenen Befehlslisten aufbauen kann. Diese Geräte waren daher auch in der Pionierzeit der Elektronenrechner bei Mathematikern besonders beliebt, da sie ihren individuellen Wünschen weit entgegenkamen. Heute sind sie allerdings durch die schnelle Kurzwortmaschine abgelöst.

Schlußbetrachtung

Nachdem die verschiedenen rechnenden Strukturen analysiert wurden, sei jetzt noch etwas über die grundsätzlichen Möglichkeiten solcher Geräte gesagt. Solche Betrachtungen werden dadurch schwierig, daß einerseits das von Journalisten geprägte Schlagwort vom Elektronengehirn bereits Allgemeingut geworden ist, und andererseits die Fachleute in einer Art Abwehrstellung immer wieder betonen, daß ein Computer nur das tun könne, was man ihm bis in alle Einzelheiten genau vorschreibe und er somit keinerlei selbständige oder gar schöpferische Denkfähigkeit besitze.

Wer sich mit den Dingen ernsthaft beschäftigt, weiß, daß hier wohl die größte Problematik der Rechenautomaten liegt. Die beiden genannten extremen Standpunkte sind falsch. Die Kybernetik ist bemüht, Klarheit in die Situation zu bringen, was ihr nur zum Teil gelingt. Dem Verfasser ist seit über dreißig Jahren klar, daß die Entwicklung unaufhaltsam in Richtung auf das künstliche Gehirn geht, wobei es lediglich ein Streit um Worte ist, wie man den Begriff Denken definieren will und ob das, was logisch leistungsfähige Rechengeräte tun, als Denken bezeichnet werden kann. Gern wird die Tätigkeit des Denkens mit dem Bewußtsein verknüpft, wodurch der ganze Fragenkomplex aber nur noch komplizierter wird. Man kann diesen Konsequenzen bis zu einem gewissen Grade ausweichen, wenn man im Sinne von Leibnizärs combinatoria“ vom universellen, mathematisch-logischen Rechnen oder dergleichen spricht. Ohne Zweifel fallen unter diesen Begriff auch alle schematischen Denkoperationen, die nach streng gegebenen Vorschriften ablaufen. Wo liegt die Grenze? Gibt es überhaupt eine? Oder können wir einen stetigen Übergang von schematischem zu schöpferischem Denken annehmen?

Die heute auf dem Markt befindlichen Rechengeräte und die hierfür ausgearbeiteten Programme befassen sich zweifelsohne mit ganz wenigen Ausnahmen nit Informationsverarbeitungsprozessen, die schematisch ablaufen und allenfalls in das Gebiet der schematischen Denkvorgänge hineinreichen. Diese Geräte können zwar „logische Befehle“ ausführen und „Entscheidungen“ fällen. Was sich hinter diesen Worten verbirgt, ist jedoch zunächst weiter nichts als ein Rechnen mit Ja-Nein-Werten, was – wie wir gesehen haben – die primitivste Stufe des Rechnens überhaupt darstellt. Der Begriff „Entscheidung“ ist relativ. Im täglichen Leben sprechen wir erst dann von einer Entscheidung, wenn es um einen Selektionsprozeß aus verschiedenen Möglichkeiten des Handelns geht, der erhebliche Konsequenzen nach sich ziehen kann; Man entscheidet sich zu studieren oder nicht zu studieren. Selbstverständlich kann ich mich auch entscheiden, ob ich ins Kino gehe öder nicht. Aber bei einer solchen belanglosen Angelegenheit verblaßt bereits der Begriff Entscheidung. Die Auswahl zwischen zwei Möglichkeiten läuft oft routinemäßig ab, z. B. die Bestimmung des Fußes, den ich zuerst bewege, wenn ich spazierengehe. Man könnte also sagen, eine Entscheidung liegt vor, wenn die zugehörigen lnformationsverarbeitungsvorgänge bewußt empfunden werden. Dann könnten wir aber Rechengeräten nicht die Fähigkeit zu Entscheidungen zusprechen. Es sei denn, wir muten auch ihnen ein Bewußtsein zu; auch das wurde zum Teil schon versucht [XIII 4].

Rein rechnerisch gesehen, ist eine Entscheidung wiederum lediglich die Bestimmung eines Ja-Nein-Wertes. Da sämtliche in unseren Rechenautomaten ablaufenden Operationen in solche mit Ja-Nein-Werten aufgelöst werden können, werden also bei einem Computer pro Sekunde viele Millionen von Entscheidungen gefällt. So gesehen ist z. B. auch die Frage, ob bei der Addition zweier Binärzahlen von der Stelle i auf die Stelle i + 1 ein Stellenübertrag stattfindet ebenfalls eine Entscheidung.

Um den Begriff „Entscheidung“ im Sinne der Umgangssprache zu gebrauchen, müßten wir also eine Bewertung einführen, die jedoch mit großen Schwierigkeiten verbunden und letzten Endes nutzlos ist.

Das Schlagwort also von der Entscheidungsfähigkeit der Rechenautomaten bringt uns nicht weiter. Aussichtsreicher erscheint es schon, den Unterschied zwischen schematischem und schöpferischem Denken zu analysieren. Man könnte zunächst fragen, ob es überhaupt statthaft ist, auch das schöpferische Denken als einen Vorgang der Informationsverarbeitung zu betrachten. Es müßte dann möglich sein, bei schöpferischen Leistungen die Ermittlung der „Lösung“ eines schwierigen Problems bis in alle Einzelheiten zu analysieren und nachzuweisen, daß eine lückenlose Kette von einzelnen Überlegungen, eventuell auf vielen Umwegen zu dem Endergebnis geführt hat. Bei schöpferischen Vorgängen im menschlichen Gehirn läuft jedoch nur ein kleiner Teil über das Bewußtsein. Oft ist es die „plötzliche Eingebung“ oder der „göttliche Funke“, der nach langem vergeblichem Suchen die Lösung bringt. Es bedurfte erst einiger Überlegungen, um plausibel zu machen, daß auch diese Vorgänge als Inforrnationsverarbeitungsprozesse komplizierter Art gedeutet werden können.

Bei der Besprechung des assoziativen Gedächtnisses haben wir bereits den grundsätzlichen Unterschied zwischen dem Arbeiten eines Computers heutiger Bauart und dem menschlichen Gehirn kennengelernt. Für den Mathematiker ist der Unterschied vielleicht am besten durch den Vergleich des Differenzierens und des Integrierens zu veranschaulichen. Gemeint ist das „symbolische“ Operieren, das heißt nicht die zahlenmäßige Differentiation bzw. Integration, sondern die formelmäßige. Beim Differenzieren haben wir für jeden vorliegenden Fall eine Folge von eindeutig gegebenen Vorschriften auszuführen. Es handelt sich also um einen schematischen Ablauf, der heute schon von Rechenmaschinen ausgeführt werden kann. Beim Integrieren hingegen können solche systematischen Vorschriften nur in wenigen einfachen Fällen angewandt werden. Oft führt der Weg zur Lösung über ein „Einsetzen“, für das jedoch keine Regel angegeben werden kann. Es muß vom Mathematiker durch einen „schöpferischen Denkvorgang“ gefunden werden. Tatsächlich spielt sich das so ab, daß im Gehirn des Mathematikers unbewußt ein großes Erfahrungsmaterial auf die Möglichkeit einer brauchbaren Einsetzung durchsucht wird. Hierzu ist eben das assoziative Gedächtnis erforderlich.

Man ist nun seit langem bemüht, auch solche Informationsverarbeitungsprozesse für den Computer aufzubereiten. Diese Versuche laufen unter dem Namen „künstliche Intelligenz“ oder auch „generelle Problemlösung“. Darunter fällt z. B. auch die Programmierung des Schachspieles. Da es nicht möglich ist, selbst mit den schnellsten Rechenmaschinen in erträglicher Zeit den absolut besten Zug nach einer schematischen Vorschrift zu errechnen, muß ein den Umständen entsprechend aussichtsreich erscheinender Zug ausgewählt werden. Auch das erfolgt nach einer Vorschrift. Wo liegt also der Übergang vom schematischen zum schöpferischen

Denken? Solange der Rechenautomat den nächsten Schachzug nach einer vom Programmierer vorgegebenen Vorschrift ausführt, ist er also nur der Arbeits-Sklave des Menschen. Aber schon ist eben dieser Mensch dabei, die strengen Vorschriften variabel zu gestalten. Unter der Bezeichnung „lernende Systeme“ oder auch „sich selbst organisierende Systeme“ sind Geräte und Programme entstanden, welche ihr Verhalten den gemachten Erfahrungen entsprechend laufend ändern, so wie es der Mensch auch tut. Geräte, die mit solchen „Lernprogrammen“ arbeiten, machen also beim Schachspiel in der gleichen Situation nicht immer denselben Zug, sondern verbessern ihre Fähigkeit zu spielen bei jedem Spiel, so wie der Mensch. Man könnte natürlich auch jetzt noch sagen, daß dieser ganze Vorgang eine Folge des Lernprogramms ist, welches der Mathematiker vorher in das Gerät eingegeben hat; aber wo liegt dann die Grenze zum schöpferischen Denken? Kann man dann nicht genau so gut sagen, daß der gesamte Lebenslauf eines Mathematikers einschließlich aller Lernvorgänge, durch die er sein Erfahrungsmaterial bildet, letzten Endes eine Folge der ihm angeborenen Grundprogramme ist, wobei die Informationen, die er von der Außenwelt im Laufe seines Lebens bekommt, lediglich als „Eingabewerte“ des Programms „Bildung der mathematischen Erfahrung“ dienen? So gesehen kann auch schöpferisches mathematisches Denken unter den Begriff der Turing-Berechenbarkeit fallen. Man sollte also vorsichtig sein mit allen Formulierungen, die bestimmte Möglichkeiten der Informationsverarbeitung der Maschine grundsätzlich absprechen.

Diese Dinge sind noch voll im Fluß. Die praktische Auswirkung der Bestrebungen um die generelle Problemlösung ist noch verhältnismäßig gering. Daraus darf man aber nicht den voreiligen Schluß ziehen, daß hier etwa im Sinne des perpetuum mobile grundsätzlich unlösbare Aufgaben vor uns liegen. Es muß daran erinnert werden, daß die qualitative Überlegenheit des menschlichen Gehirns weitgehend auf eine quantitative zurückzuführen ist. Wir sind heute und wahrscheinlich auf lange Zeit hin noch nicht in der Lage, viele Milliarden informationsverarbeitender Elemente auf engem Raum zu vereinigen. Quantität ist aber eine technische Frage. Man halte sich den Stand der Computertechnik vor 30 Jahren vor Augen, um zu erkennen, daß die nächsten 30 Jahre vielleicht ebenso entscheidende Überraschungen bringen können.

Daß hier gewaltige Aufgaben und Probleme vor uns liegen, leuchtet ein. Sind wir dieser Situation gewachsen?

Leider muß man sagen, daß unsere gesamte „Software“-Entwicklung zu sehr auf die Aufgaben des Tages zugeschnitten ist. Die eingeführten algorithmischen Sprachen wurden vor 15 Jahren in erster Linie zur Formulierung numerischer Rechnungen geschaffen. Heute sind wir längst zum allgemeinen Rechnen übergegangen, aber unsere Programmiersprachen sind an den Grenzen ihrer Möglichkeiten angekommen. Vielleicht müssen wir das nachholen, was seinerzeit versäumt wurde, nämlich das Problem an der Wurzel zu fassen, wie es der Verfasser sich im

„Plankalkül“ zur Aufgabe gestellt hatte. Es fehlt nicht an gründlichen theoretischen Untersuchungen. Die Informationstheorie und die Automatentheorie sind zu Wissenszweigen mit einer umfangreichen Literatur angeschwollen. Aber beide Gebiete sind sehr abstrakt und führen weitgehend ein Eigenleben unabhängig von der Computerentwicklung.

Leider haben wir diese Dinge nicht im Griff. Was not tut, ist eine Grundlagenforschung, die sich zum Ziele setzt, Theorie und Praxis zu vereinigen und uns auf die Aufgaben der kommenden Generation von Wissenschaftlern und Ingenieuren vorzubereiten. Hierin sieht der Verfasser die größte Problematik in der augenblicklichen Situation der Rechenautomaten.