# Über die Vorbereitungsarbeit beim automatischen Rechnen

Autor(en): **Rutishauser, H.**

Objekttyp: **Article**

Zeitschrift: **Mitteilungen / Vereinigung Schweizerischer Versicherungsmathematiker = Bulletin / Association des Actuaires Suisses = Bulletin / Association of Swiss Actuaries**

Band (Jahr): **57 (1957)**

PDF erstellt am: **27.07.2024**

Persistenter Link: https://doi.org/10.5169/seals-551147

# Über die Vorbereitungsarbeit beim automatischen Rechnen

Von *H. Rutishauser*, Zürich

Für den Mathematiker, der eine grössere Berechnung durchzuführen hat, besteht heute die Möglichkeit, dieselbe mit einem Rechenautomaten (elektronische Rechenmaschine) durchführen zu lassen. Zu diesem Zweck muss er mit seinem Problem an eines der heute bereits bestehenden Rechenzentren gelangen, welches dann die Angelegenheit für ihn erledigt. Nun entsteht aber dem Rechenzentrum mit der Übernahme eines Rechenauftrages erhebliche Arbeit, indem das Problem erst sorgfältig präpariert werden muss, ehe es dem Rechenautomaten übergeben werden kann. Diese Vorbereitungsarbeit ist zur Zeit noch das Sorgenkind des automatischen Rechnens, deshalb soll hier über die damit zusammenhängenden Probleme berichtet werden.

## § 1 Die Sprache der Rechenautomaten

Wir gehen von der Annahme aus, dass der Auftraggeber die Lösung seines Problems bereits durch mathematische Formeln beschrieben dem Rechenzentrum übergeben habe. Dann besteht die Vorbereitungsarbeit lediglich noch aus dem Programmieren, d.h. in der Übersetzung der Formeln in die Sprache des Automaten, welche sich leider in verschiedener Hinsicht von der Sprache der Mathematik unterscheidet.

Die Sprache der ersten elektronischen Rechenmaschinen war noch ziemlich kompliziert und erschwerte das Programmieren erheblich, aber mit der Zeit haben die Konstrukteure eingesehen, dass dies ein ernstliches Hindernis für die erfolgreiche Benützung darstellt. Es wurden daher auch in dieser Hinsicht einige Fortschritte erzielt und heute hat sich im wesentlichen die in Abb. 1 dargestellte Organisation eingebürgert.

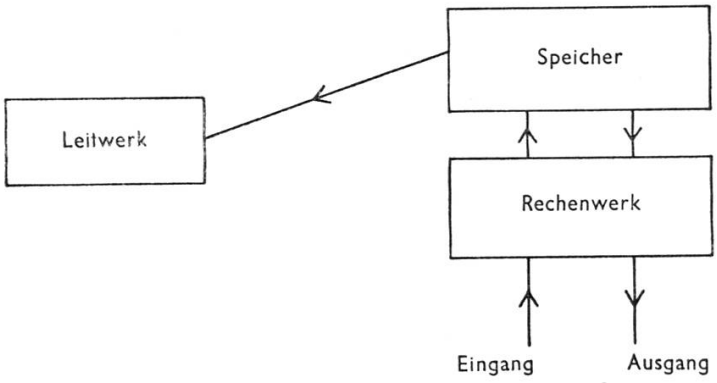


Abb. 1

Wesentlich sind für uns vorläufig nur das *Rechenwerk*, das die Rechengrössen verarbeitet und das *Speicherwerk*, welches sie aufbewahrt. Der Speicher besteht aus einer gewissen Anzahl (in der Regel zwischen 1000 und 50 000) diskreten Einheiten, Zellen genannt, von denen jede eine Zahl aufnehmen kann. Zur Wiederauffindung abgelegter Rechengrössen sind die Zellen des Speicherwerks fortlaufend numeriert; man spricht dann etwa vom «Inhalt der Zelle 117» und meint damit die in dieser Zelle gespeicherte Zahl.

## § 2 Ausführung der arithmetischen Operationen

Auffallenderweise können die meisten Rechenautomaten nur gerade die 4 arithmetischen Grundoperationen direkt ausführen; höhere Operationen müssen auf diese zurückgeführt werden. Das Rechenwerk eines Rechenautomaten entspricht also in seinen Fähigkeiten etwa einer gewöhnlichen Bürorechenmaschine. Aber im Detail ist das Rechenwerk von Rechenautomat zu Rechenautomat sehr stark verschieden. Es gibt Maschinen, die im Dualsystem, andere, die dezimal arbeiten; ausserdem unterscheidet man zwischen festem und gleitendem Komma, und schliesslich arbeiten die verschiedenen Rechenautomaten mit unterschiedlicher Genauigkeit (meist zwischen 8 und 15 Dezimalen). Aber davon und von abweichenden Bezeichnungen abgesehen gilt für viele moderne Rechenautomaten das folgende Prinzip:

Es gibt im Rechenwerk ein ausgezeichnetes Register $MR$, welches immer eine Zahl enthält, nämlich das Resultat der vorangehenden Rechenoperation. Jede arithmetische Operation verbindet diese Zahl mit einer Zahl aus dem Speicher, das Resultat geht wieder nach $MR$.

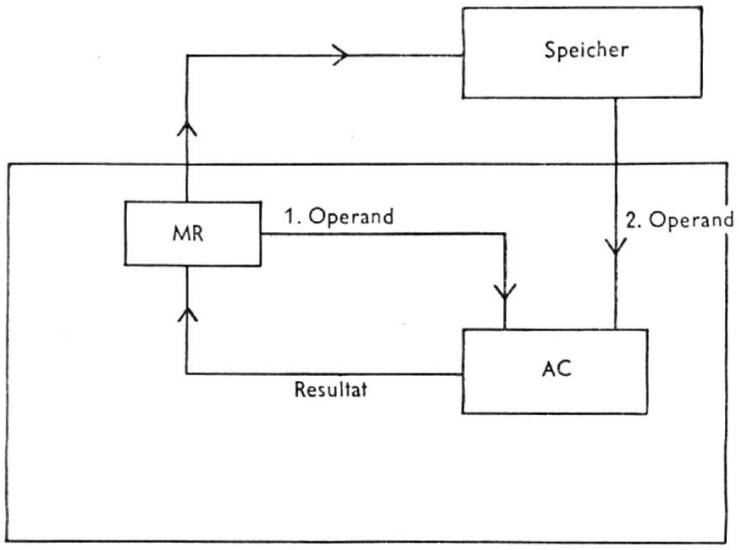


Abb. 2

Die arithmetischen Operationen werden durch Befehle ausgelöst:

$\times$ 758 «Multipliziere die Zahl in $MR$ mit dem Inhalt der Zelle 758».

$+$ 375 «Addiere den Inhalt der Zelle 375 zur Zahl in $MR$».

: 15 «Dividiere den Inhalt von $MR$ durch den Inhalt der Zelle 15», usw.

Ferner:

$A$ 37 «Hole den Inhalt der Zelle 37 nach $MR$». Diese Operation wird wie eine Addition gesteuert, aber der Zugang von $MR$ nach $AC$ unterbunden, so dass nachher der Inhalt von Zelle 37 in $MR$ steht. (Der Inhalt der Zelle 37 wird dabei nicht verändert.)

$S$ 37 «Speichere den Inhalt von $MR$ in der Zelle 37». In diesem Fall wird natürlich der frühere Inhalt der Zelle 37 gelöscht, aber $MR$ bleibt unverändert.

## § 3 Das Programm für eine einfache Formel

Aus einzelnen Operationsbefehlen kann man nun ganze Programme aufbauen, z. B. für die Formel

$$\frac{a-b}{c} + bc \Rightarrow x \text{ [1]}.$$

[1]) Das pfeilartige Gleichheitszeichen soll ausdrücken, dass die rechts stehende Grösse berechnet werden soll.

Wenn man annimmt, dass $a$, $b$, $c$ in den Zellen 10, 11, 12 gespeichert seien und verlangt, dass $x$ nach Zelle 13 kommen soll, so muss man der Maschine folgende Befehle erteilen:

$A$ 10
$-$ 11
$:$ 12
$S$ 20

} Hier wird erst $\frac{a-b}{c}$ berechnet und in Zelle 20 gespeichert.

$A$ 11
$\times$ 12
$+$ 20
$S$ 13

} Hier wird $bc$ berechnet und der Inhalt der Zelle 20 dazuaddiert.

An dieser Stelle muss wohl darauf hingewiesen werden, dass bei älteren Maschinen das Programm in Form eines Lochstreifens der Maschine zugeführt wurde, welche es dann Befehl für Befehl verarbeitete (Bandgesteuerte Rechenautomaten). Etwa ab 1949 ist man dazu übergegangen, die Befehle durch Zahlen darzustellen und wie die Rechengrössen (aber von diesen getrennt!) im Speicherwerk der Maschine aufzubewahren. Die Maschine sucht dann die «Befehlszellen» der Reihe nach ab und führt die darin enthaltenen Befehle aus (Maschinen mit gespeichertem Programm).

Im obigen Beispiel könnte man die 8 Befehle etwa in den Zellen 0–7 aufbewahren und in Zelle 8 noch einen *Stop-Befehl* (Fin) unterbringen. Dies ist schon deshalb notwendig, damit die Maschine nach Ausführung der Befehle 0–7 nicht in die Rechengrössen (Zelle 10 ff.) «hineinläuft» und die dort befindlichen Zahlen als Befehle interpretiert.

## § 4 Die Struktur eines Problems

Wie man an dem obigen Beispiel erkennt, ist die Übersetzung einer Formel in die Automatensprache durchaus nicht schwierig [1]). In der Tat liegen die Schwierigkeiten des Programmierens nicht so sehr in der Übersetzung der einzelnen Formeln als in der *Struktur* des mathematischen Problems; diese ist es, die beim Programmieren Mühe und Schwierigkeiten verursacht.

Würde die ganze Aufgabe in der einmaligen Auswertung der in § 3 angegebenen Formel bestehen, so wäre die Struktur *linear:*

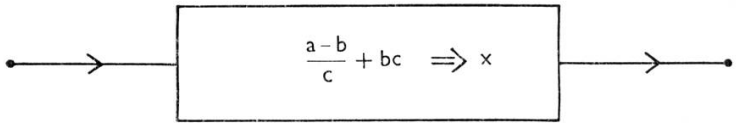


Abb. 3

[1]) Eine Einschränkung muss allerdings erwähnt werden: Wenn die betreffende Maschine mit *festem Komma* rechnet, dann kann das Programm nicht einfach so hingeschrieben werden, weil man sich auch noch um die Grössenordnung der auftretenden Zahlen kümmern muss.

Etwas realistischer ist schon die Annahme, dass die Formel für mehrere Zahlensätze $ab_1c_1$, $ab_2c_2$, $ab_3c_3$, usw. auszurechnen sei:

$$\frac{a-b_1}{c_1} + b_1c_1 \Rightarrow x_1,$$

$$\frac{a-b_2}{c_2} + b_2c_2 \Rightarrow x_2,$$

usw. bis

$$\frac{a-b_{100}}{c_{100}} + b_{100}c_{100} = x_{100}.$$

Wir haben zwar im wesentlichen immer noch dieselbe Formel, aber diese ist wiederholt auszurechnen, und diese Wiederholung drückt sich durch die *zyklische* Struktur aus, die wir schematisch darstellen (siehe Abb. 4).

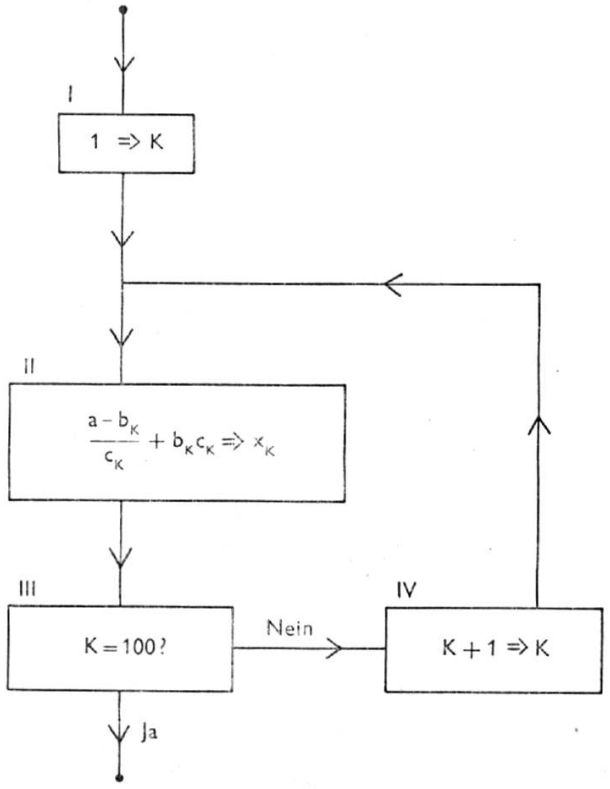


Abb. 4

Ein solches Schema, genannt *Flussdiagramm*, ist noch kein Programm und hat zunächst auch nichts mit Rechenautomaten zu tun, sondern ist vorerst nur als Mittel zur Darstellung des Rechenablaufs zu verstehen. Es enthält alle Angaben über die durchzuführende Berechnung in übersichtlicher Weise und wird daher die Aufstellung des Rechenprogramms stark erleichtern und Programmierungsfehler vermeiden helfen.

## § 5 Das Rechnen mit Befehlen

Die mannigfachen Schwierigkeiten, die sich beim Programmieren von Problemen mit komplizierter Struktur ergeben, sollen an dem soeben behandelten einfachen Beispiel

$$\frac{a-b_k}{c_k} + b_k c_k \Rightarrow x_k, \quad (k = 1, 2, \ldots, 100)$$

erläutert werden. Für die Aufstellung des Programms müssen wir zuerst die Zellenverteilung für die Rechengrössen festlegen: Wir haben einen $a$-Wert und je 100 $b$-, $c$-, $x$-Werte. Dazu kommen noch 100 Zwischenresultate, die wir aber nicht weiter benötigen und daher in derselben Zelle speichern können. Dies erlaubt uns die folgende Zellenverteilung:

$a$ in Zelle 100
$b_k$ in Zelle $100+k$
$c_k$ in Zelle $200+k$
$x_k$ in Zelle $300+k$
Zwischenresultat in Zelle 51

und damit folgendes Programm für $\frac{a-b_k}{c_k} + b_k c_k \Rightarrow x_k$:

| | | | |
|---|---|---|---|
| $A$ | 100 | $A$ | $100+k$ |
| $-$ | $100+k$ | $\times$ | $200+k$ |
| $:$ | $200+k$ | $+$ | 51 |
| $S$ | 51 | $S$ | $300+k$. |

Dieses Programm muss 100mal durchgerechnet werden, nämlich für $k = 1, 2, \ldots, 100$. Aber dieses Programm ist noch rudimentär; es enthält nur die «Rechenbefehle», die dem Feld II im Flussdiagramm Abb. 4 entsprechen. Tatsächlich tragen aber auch die übrigen Felder des Flussdiagramms zum Programm bei, sie liefern die sogenannten «organisatorischen Befehle»:

Zunächst verlangt Feld IV, dass die Grösse $k$ um 1 vergrössert wird. Da aber die Adressen einiger Rechenbefehle ebenfalls von $k$ abhängig sind, müssen diese bei dieser Gelegenheit ebenfalls verändert (hier um 1 erhöht) werden. Da die Befehle ebenfalls durch Zahlen dargestellt und wie diese gespeichert sind, ist dies leicht möglich: Beispielsweise wird der Befehl $\times$ 175 in der ERMETH durch die Zahl 0400175 dargestellt; addiert man 1 dazu, so entsteht 0400176, also der Befehl $\times$ 176.

Ferner wird in Feld I verlangt, dass die Berechnung mit dem Indexwert $k = 1$ begonnen wird; damit müssen gleichzeitig die Anfangswerte der von $k$ abhängigen Adressen eingesetzt werden.

Schliesslich muss in Feld III geprüft werden, ob $k$ den Endwert 100 erreicht hat. Solange dies nicht der Fall ist, muss die Rechnung fortgesetzt werden, andernfalls wird sie abgebrochen. Insgesamt ergibt sich dann folgendes Programm:

| Adresse | Befehl | | Adresse | Befehl | |
|---|---|---|---|---|---|
| 0 | $A$ 43 | Feld I | 23 | $A$ 13 | Feld IV |
| 1 | $S$ 13 | | 24 | + 48 | |
| 2 | $A$ 44 | | 25 | $S$ 13 | |
| 3 | $S$ 14 | | 26 | $A$ 14 | |
| 4 | $A$ 45 | | 27 | + 48 | |
| 5 | $S$ 16 | | 28 | $S$ 14 | |
| 6 | $A$ 46 | | 29 | $A$ 16 | |
| 7 | $S$ 17 | | 30 | + 48 | |
| 8 | $A$ 47 | | 31 | $S$ 16 | |
| 9 | $S$ 19 | | 32 | $A$ 17 | |
| 10 | $A$ 48 | | 33 | + 48 | |
| 11 | $S$ 50 | | 34 | $S$ 17 | |
| | | | 35 | $A$ 19 | |
| 12 | $A$ 100 | Feld II | 36 | + 48 | |
| 13 | (— 100+$k$) | | 37 | $S$ 19 | |
| 14 | (: 200+$k$) | | 38 | $A$ 50 | |
| 15 | $S$ 51 | | 39 | + 48 | |
| 16 | ($A$ 100+$k$) | | 40 | $S$ 50 | |
| 17 | (× 200+$k$) | | 41 | $C$ 12 | |
| 18 | + 51 | | | | |
| 19 | ($S$ 300+$k$) | | 42 | Fin | Stop |
| 20 | $A$ 50 | Feld III | 43 | — 101 | Konstanten |
| 21 | — 49 | | 44 | : 201 | |
| 22 | $Co$ 42 | | 45 | $A$ 101 | |
| | | | 46 | × 201 | |
| | | | 47 | $S$ 301 | |
| | | | 48 | 1 | |
| | | | 49 | 100 | |

Das Programm befindet sich also in den Zellen 0–42, einige Konstanten sind in den Zellen 43–49; die Zellen 50 und 51 enthalten Rechengrössen, nämlich den Index $k$ und die Hilfsgrösse $\frac{a-b_k}{c_k}$. Die eingeklammerten Befehle 13, 14, 16, 17, 19 werden erst während der Rechnung (durch die Befehle 0–11) eingesetzt; die betreffenden Stellen im Programm können deshalb leer gelassen werden.

## § 6 Der Ablauf der Berechnung

Nach Durchlauf der Befehle 0–11 kommt die Maschine zu den eigentlichen Rechenbefehlen, die jetzt mit $k = 1$ eingesetzt sind, so dass $x_1$ berechnet wird. Nachher kommt sie, da sie die Befehle der Reihe nach aus dem Speicher nimmt, zu 20 und 21, wo sie die Differenz $k-100$ bildet und schliesslich zu 22 *Co* 42.

Dies ist eine Operation besonderer Art, die mit anderem Namen bei allen Rechenautomaten vorhanden ist. Wir betrachten zuerst die Wirkung von

*C* 42 (Unbedingter Sprung).

Dies bewirkt, dass der normale Befehlsablauf unterbrochen wird und die Maschine beim Befehl in Zelle 42 weiterrechnet.

*Co* 42 (Bedingter Sprung)

hat dieselbe Wirkung, falls in *MR* (Resultat der letzten Rechenoperation) die Zahl 0 steht, andernfalls bleibt der Befehl wirkungslos. Es gibt noch weitere Sprungbefehle, z. B.

*C+* (wird nur wirksam, wenn in *MR* eine positive Zahl steht) und

*Cz* (wird nur wirksam, falls der am Schaltpult einstellbare Schalter *Z* auf «Ein» steht).

In unserm Fall steht die Differenz $k-100$ in *MR*, wenn die Maschine zum Befehl *Co* 42 kommt, und dies ist beim ersten Durchlauf $-99 \neq 0$, so dass der Sprungbefehl definitionsgemäss wirkungslos bleibt, die Maschine geht daher zur Ausführung der Befehle 23–40, die dafür sorgen, dass $k$ und alle von $k$ abhängigen Adressen um je 1 erhöht werden. Anschliessend kommt der Befehl *C* 12 (in Zelle 41) zur Ausführung, infolgedessen springt die Maschine zurück zum Befehl 12 und rechnet von da aus weiter. Es wird dann $x_2$ berechnet, weil in den Rechenbefehlen jetzt die Adressen von $b_2$, $c_2$, $x_2$ stehen. Danach werden die Adressen weiter vergrössert und wieder beim Befehl in Zelle 12 begonnen, usw.

Erst wenn $k = 100$ ist und mit den Befehlen 12–19 $x_{100}$ berechnet wurde, wird die mit den Befehlen 20–21 berechnete Differenz $k - 100 = 0$, so dass der Sprungbefehl *Co* 42 erstmalig wirksam wird; die Maschine geht also nach 42 und hält dort an (Fin-Befehl).

Wie man erkennt, erfordern die organisatorischen Befehle 0–11 und 20–42 mehr Raum und auch mehr Rechenzeit als die eigentliche Berechnung (Befehle 12–19); und zwar ist dies beim automatischen Rechnen eine allgemeine Erscheinung [1]).

Die Angabe, dass ein Rechenautomat pro Sekunde $n$ Rechenoperationen ausführen könne, ist daher insofern irreführend, als der Auftraggeber für die Abschätzung des Rechenaufwandes an Hand seiner Formeln natürlich nur die eigentlichen Rechenoperationen zählt. Die Erfahrung zeigt, dass solche Schätzungen zweckmässig mit 3 multipliziert werden.

## § 7 Rechenkontrollen

Es wird nicht oft und nicht sehr gern von den Fehlern gesprochen, die Rechenautomaten gelegentlich machen. Sie sind jedoch nicht absolut zu vermeiden, und es ist auch nicht sehr schlimm, sofern diese Fehler nicht allzuhäufig auftreten.

Es ist aber unbedingt notwendig, dass man sich vor den Folgen solcher Fehler schützt, indem man mathematische Kontrollen in das Programm einbaut. Diese beruhen darauf, dass ein Rechenautomat mit Hilfe der bedingten Sprungbefehle auf Grund der bisherigen Rechenergebnisse über den weiteren Verlauf der Rechnung entscheiden kann. Ebensogut kann er auf Grund mathematischer Identitäten über die Richtigkeit der errechneten Resultate entscheiden und bei Bedarf Teile der Rechnung wiederholen.

Bei vielen Maschinen, darunter auch der ERMETH, gibt es Rechenkontrollen, die *in die Maschine eingebaut* sind. Dieser Schutz ist zwar nicht absolut, gestattet aber eine Reduktion der programmierten Rechenkontrollen, ohne sie ganz überflüssig zu machen.

---

[1]) Bei einer Maschine mit *Indexregistern* lassen sich sowohl die Anzahl der organisatorischen Befehle als auch die Rechenzeit erheblich reduzieren. Beispielsweise könnte dieselbe Aufgabe für die ERMETH unter Benützung der Indexregister mit 18 Befehlen programmiert werden.

## § 8 Programmierungsfehler

Leider befinden sich in einem Programm – sei es durch Flüchtigkeit, sei es infolge Fehlüberlegungen beim Aufstellen des Programms – meist noch einige Fehler. Wird ein fehlerhaftes Programm in die Maschine eingegeben und die Rechnung eingeleitet, so folgt die Maschine getreu den falschen Anweisungen. Im günstigsten Fall steht die Maschine in einem solchen Fall nach einiger Zeit einfach still.

Wenn beispielsweise im Programm für die Formel

$$\frac{a - b_k}{c_k} + b_k c_k \Rightarrow x_k$$

die Erhöhung der Adresse von $x$ (Befehle 35–37) vergessen wird, so werden alle berechneten $x$-Werte in dieselbe Zelle hineingeschrieben. Jeder neue $x$-Wert löscht daher den vorangehenden aus, mit dem Ergebnis, dass am Schluss nur noch $x_{100}$ (in der Zelle 301) greifbar ist. Die Ursache wird nicht einmal leicht zu erkennen sein, denn da von der vorangehenden Berechnung noch irgendwelche Zahlen in den Zellen 302–400 standen, sind diese jetzt noch da und man wird zunächst nur feststellen, dass alles falsch ist. Die in die Maschine eingebauten Rechenkontrollen werden auch nicht helfen, da die Maschine ja keinen Fehler gemacht hat. Dagegen würde eine programmierte Kontrolle, die auf mathematischen Identitäten beruht, den Fehler frühzeitig aufdecken.

In jedem Fall wird man beim Feststellen verdächtiger Vorgänge den Gang der Rechnung schleunigst unterbrechen und muss dann aus dem geleisteten Unsinn auf die Ursache, d.h. auf den Fehler im Programm zurückschliessen. Dies ist oft schwierig und erfordert gelegentlich das befehlsweise Durchspielen des Programms von Hand. Jedenfalls dürfte klar sein, dass Programmierungsfehler ausserordentlich kostspielig und zeitraubend sind und daher unter allen Umständen vermieden werden müssen. Es wurde bereits erwähnt, dass das Aufstellen eines Flussdiagramms das Programmieren erleichtert und damit automatisch auch Programmierungsfehler vermeiden hilft. Deshalb sollte kein grösseres Problem ohne Aufstellung eines Flussdiagramms programmiert werden.

## § 9 Automatisches Programmieren

Die Tatsache, dass das Programmieren selbst bei relativ einfachen Problemen oft gegen 100 Arbeitsstunden erfordert und dann die Elimination der Programmierungsfehler sogar den kostspieligen Rechenautomaten noch einige Stunden in Anspruch nehmen kann, ruft gebieterisch nach Massnahmen zur Vereinfachung des Programmierens, die wirksamer sind als das Flussdiagramm.

Solche Methoden sind tatsächlich entwickelt worden, sie werden gewöhnlich als «automatisches Programmieren» bezeichnet. Indessen hat der Verfasser bereits 1952 gezeigt, dass es möglich ist, noch einen Schritt weiter zu gehen und *die Maschine das Programm selbst berechnen zu lassen*, wenn nur die dem Problem zugrundeliegenden Formeln in geeigneter Weise aufnotiert werden (sogenannte algorithmische Schreibweise).

Diese normierte Schreibweise lautet z. B. für die Berechnung von $\ln x$ (10stellig genau) mit Hilfe der Halbwinkelformel des hyperbolischen Cosinus wie folgt:

$$(x^2+1) : 2x \Rightarrow p_0,$$

$$(x^2-1) : 2x \Rightarrow q_0;$$

for $k$ from 1 to 20:

$$\sqrt{\tfrac{1}{2}(1+p_{k-1})} \Rightarrow p_k,$$

$$q_{k-1} : p_k \Rightarrow q_k;$$

if $p_k - 1{,}00005$ neg, $\quad 3q_k : (2+p_k) \Rightarrow \ln x,$

if not, continue

repeat $k$.

Die ausserordentlich umfangreichen und zeitraubenden Arbeiten für die Aufstellung eines Programms für die Programmberechnung sind zur Zeit im Gange, dürften aber noch längere Zeit beanspruchen. Wenn aber dieses Programm einmal vorliegt, wird es nur noch nötig sein, ein Problem in der oben angedeuteten normierten Schreibweise zu formulieren; alles übrige wird der Rechenautomat besorgen.