Grundlagen der Algorithmentheorie und ihre Anwendungen

Johann Michael Bruhn

Bartholomä 1976

Erstellt und eingereicht als

Schriftliche Hausarbeit zur Zweiten Prüfung
für das Lehramt an Realschulen im Dezember 1976
vorgelegt von
Johann Michael Bruhn
Schwäbisch Gmünd, 20. August 1976
Referent: Prof. Dr. rer. nat. Klaus Menzel

Satz: 2005 von Johann Michael Bruhn
gesetzt aus der Times 11p
mit LaTeX 2ε und
gedruckt mit Ghostview und Ghostscript

Inhaltsverzeichnis

Einführung

David Hilbert, der große Göttinger Mathematiker, legte auf dem **Zweiten Internationalen Mathematikerkongreß** in Paris im Jahre 1900 eine Liste von 23 schwierigen, noch ungelösten Problemen vor. Im Nachhinein könnte man von einer nicht allzu leichten Hausarbeit an die Mathematiker für das neu begonnene Jahrhundert sprechen. Hilbert ist zu diesem Zeitpunkt „von der Lösbarkeit jedes vernünftig gestellten Problems überzeugt“ [10, 219]. Für ihn und andere Formalisten „ist jedes mathematische Problem grundsätzlich entscheidbar“ [10, 216]. Daß bereits im 19. Jahrhundert Unlösbarkeitsbeweise geführt worden waren (Konstruktionen mit Zirkel und Lineal, Auflösbarkeit algebraischer Gleichungen mit Hilfe von Radikalen), wurde dabei keineswegs ignoriert. Da sich diese Beweise nur auf gewisse Hilfsmittel bezogen (Zirkel und Lineal, Radikale), war es grundsätzlich möglich, daß andere Hilfsmittel das Problem lösten [3, 3]. Das zehnte Hilbertsche Problem lautet [19, 11]:

> „Es ist ein Algorithmus aufzustellen, mit dessen Hilfe man von einer beliebigen diophantischen Gleichung[1] aussagen kann, ob sie eine ganzzahlige Lösung besitzt oder nicht.“

Es ist trivial, daß jede diophantische Gleichung entweder Lösungen besitzt oder nicht. Auch waren für gewisse Klassen von diophantischen Gleichungen bereits Algorithmen bekannt, welche die Lösbarkeit entschieden. Lediglich für beliebige diophantische Gleichungen wollte sich schon seit langer Zeit kein Algorithmus finden lassen, und dies änderte sich nach 1900 auch nicht. Man kann Hilbert aber nicht vorwerfen, daß er ein ungeeignetes Hilfsmittel vorgeschlagen habe. Nach einer Formulierung von Emil Post [3, 3] scheint es so zu sein, daß dem Menschen außer den Algorithmen keine weiteren Hilfsmittel für solche Probleme zur Verfügung stehen. Somit wäre die algorithmische Unlösbarkeit einer generellen Unlösbarkeit gleichzusetzen.
Erst im Jahr 1970 gelang es Matijesevic, den folgenden Satz zu beweisen [18, 116]:

> „Es gibt keinen Algorithmus, der zu jeder diophantischen Gleichung entscheidet, ob sie lösbar ist.“

1900 hätte dieser Satz wohl noch allerhöchstes Erstaunen bei den Kongreßteilnehmern hervorgerufen; 1970 wären etliche Mathematiker erstaunt gewesen, wenn ein derartiger Algorithmus gefunden worden wäre: Die Anschauungen hatten sich geändert!
Im Satz von Matijesevic wird ausgesagt, daß die Menge aller Algorithmen kein Element besitzt, welches das zehnte Hilbertsche Problem löst. Zu Beginn des Jahrhunderts wäre dieser Satz nicht zu beweisen gewesen, da es keine mathematisch exakte Definition für „die Menge

[1] Eine Gleichung $f(x_1, x_2, ..., x_n) = 0$, für die nur ganze Zahlen als Lösungen gesucht werden, heißt diophantische Gleichung“ [13, 57]

aller Algorithmen“ gab. Es war wiederum HILBERT, der durch die Formulierung des Entscheidungsproblems für den Prädikatenkalkül seine Kollegen zur präzisen Definition des Algorithmenbegriffs anregte. In der vorliegenden Arbeit soll aufgezeigt werden, wie ein präziser Algorithmenbegriff gewonnen werden kann. Dabei wird auf eine an Beispielen reiche Darstellung Wert gelegt. Auch soll die historische Entwicklung betrachtet werden. Die Algorithmentheorie selbst ist aber so umfangreich, daß sie im Rahmen dieser Arbeit nur exemplarisch behandelt werden kann. Umfangreiche oder abstrakte Beweise werden zugunsten der Anschaulichkeit und des Überblicks vermieden.

Kapitel 1

Probleme, Lösungsverfahren, Algorithmen

Gewisse Festlegungen und Aussagen über Algorithmen müssen willkürlich erscheinen, wenn man nicht an Beispielen die Zweckmäßigkeit dieser Setzungen überprüfen kann. Diese Beispiele sollen nun durch einige Multiplikationsverfahren gegeben werden.

1.1 Allerlei Multiplikationsverfahren

1.1.1 Griechenland um 300 v. Chr.

Die Multiplikation war für die alten Griechen problematisch, da sie kein Zeichen für die Null hatten und auch sonst über kein vollwertiges Stellenwertsystem verfügten. Bei den milesischen Zahlzeichen wird das griechische Alphabet um drei Zeichen vermehrt und man gewinnt durch Querstriche über den Buchstaben Individualzeichen für folgende Zahlen [10, 22]:

> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 200, 300, 400, 500, 600, 700, 800, 900

Eine Multiplikation mußte in der Art einer Polynommultiplikation ausgeführt werden. Für die Zahlen $\overline{\varphi\lambda\vartheta}$ und $\overline{\sigma o\gamma}$ soll dies in einer bereits „übersetzten“ Form durchgeführt werden.

$$\begin{aligned} 739 \cdot 273 &= (700 + 30 + 9) \cdot (200 + 70 + 3) \\ &= 140000 + 49000 + 2100 + \\ &\quad +6000 + 2100 + 90 + \\ &\quad +1800 + 630 + 27 \\ &= 201747 \end{aligned}$$

Dabei waren die einzelnen Multiplikationen und Additionen in einem Zahlensystem durchzuführen, das erst einige Anzeichen eines Stellenwertsystems zeigte. Was Wunder, wenn man das Rechnen gern den Gelehrten überließ. Gleichheitsstriche, Additionskreuz und Multiplikationspunkt gab es noch nicht, statt eines Zeichens schrieb man noch ein ganzes Wort. Die Darstellung des Rechenganges war folglich nicht so knapp und übersichtlich wie heute. Kopfrechner hatten es zwar nicht schwerer als heute, doch ist es nicht einfach, sich den Stand der Rechnung samt Zwischenergebnis zu merken und gleichzeitig die Einzelmultiplikation durchzuführen.

1.1.2 Das Rechenbrett

war schon um etwa 1100 v. Chr. bei den Chinesen bekannt. Ägypter und Römer verwendeten es, in Mitteleuropa war es bis ins 17. Jahrhundert und in Asien ist es noch heute in Gebrauch. Zwischen- und Endergebnisse werden durch die Lage von Steinchen oder Holzperlen dargestellt. Man addiert durch Verschieben von Steinchen. Die Multiplikation wird durch fortlaufende Addition von Ergebnissen der Einzelmultiplikationen durchgeführt. Betrachtet man ein Rechenbrett genauer, so stellt man fest, daß hier ein Gerät zum Rechnen in Stellenwertsystemen vorliegt. Dies muß erstaunen, denn die alten Zahldarstellungen der Chinesen, Ägypter und Römer waren zwar dezimal, aber keine Stellenwertsysteme. Diese Übersetzung in eine andere Zahldarstellung war folglich mehr als nur eine zuverlässige Gedächtnisstütze.

1.1.3 Eines der altindischen Multiplikationsverfahren

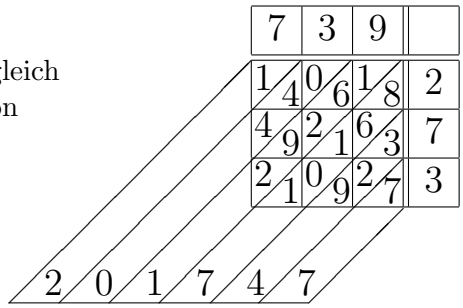
7	3	9	
			2
			7
			3

Abb. 1: Altindische Multiplikation

setzte die „Herstellung einer schachbrettartigen Figur voraus“ [4, 611]. Es läßt sich nur für Stellenwertsysteme gebrauchen und ist dem schulüblichen Ansatz verwandt. Ein solches Raster ist in der nebenstehenden Abb. 1 dargestellt.
Für die endgültige Berechnung ist das Raster nach Abb. 2 zu erweitern. Aus dieser Abbildung läßt sich auch leicht ersehen, wie die Zwischenwerte einzutragen sind: Zehner über die Diagonale und Einer darunter. Durch die Diagonalen werden Schrägspalten gebildet. Hat man alle Felder ausgefüllt, so beginnt man von rechts mit der Addition aller Zahlen im gleichen Schrägspalt.
Der Einerwert des Additionsergebnisses wird unten im Schrägspalt notiert; der Zehnerwert kommt als Übertrag nach links in die nächste Schrägspalte. Durch Vergleich mit der schulüblichen Darstellung kann man erkennen, daß man mit der indischen Methode Überträge erst bei der abschließender Addition beachten muß, dagegen muß stets ein entsprechendes Raster vorhanden sein.

```
 739·273
-------
 1478--
  5173-
   2217
-------
 201747
```

Abb. 2: Methodenvergleich Multiplikation



1.1.4 Moderne Rechenmaschinen

Große Unterschiede ergeben sich in den Verfahren, wenn man die Aufgabe einer mechanischen oder einer elektronischen Rechenmaschine anvertraut. Am Ergebnis wird sich aber nichts ändern, unabhängig davon, ob man es durch Zahnräder oder Transistoren gewinnt. Bemerkenswert ist dabei, daß der elektronische Rechner mit einem Minimalalphabet von zwei Zeichen auskommt (Kontakt offen, Kontakt geschlossen).

1.1.5 Die russische Bauernmethode

Die Darstellung von Zahlen in einem Stellenwertsystem durch nur zwei Zeichen läßt sich bereits bei Leibniz finden. Diese Dualzahlen werden in der Schule aber erst seit ein paar Jahren behandelt. Trotzdem war im alten Rußland eine Methode bekannt, die sich einer „verkappten Kodierung in Dualzahlen“ [9, 368] bediente. Für eine Multiplikation mußte man nur halbieren, verdoppeln und addieren können. Zuerst sei die schriftliche Darstellung gezeigt.

739	273
~~1478~~	~~136~~
~~2956~~	~~68~~
~~5912~~	~~34~~
11824	17
~~23648~~	~~8~~
~~47296~~	~~4~~
~~94592~~	~~2~~
189184	1
201747	

Das Verfahren läßt sich folgendermaßen beschreiben:

1. Multiplikand und Multiplikator werden nebeneinander geschrieben.
2. Der Multiplikator wird halbiert und das Ergebnis ohne Bruchanteil unter den Multiplikator gesetzt. Mit der neuen Zahl wird ebenso verfahren. Man setzt dies fort, bis man 1 erhält.
3. Der Multiplikand wird verdoppelt und das Ergebnis unter den Multiplikanden geschrieben. Man setzt das Verfahren fort, bis man in der Zeile angelangt ist, in der die 1 unter dem Multiplikator steht.
4. Steht in einer Zeile der Multiplikatorspalte eine gerade Zahl, so wird die gesamte Zeile gestrichen.
5. Alle nicht gestrichenen Zahlen der Multiplikandenspalte werden addiert und bilden das Ergebnis der Multiplikation.

In ähnlicher Form war dieses Verfahren bereits den alten Ägyptern bekannt [10, 12].

1.2 Eine erste Präzisierung

Die in **1.1** beschriebenen oder teilweise nur angedeuteten Multiplikationsverfahren ließen sich noch eindeutiger festlegen, so daß die inhaltliche Deutung von Trachtenbrot darauf zutreffen würde [19, 7]:

> „Unter einem **Algorithmus** versteht man eine genaue Vorschrift, nach der ein gewisses System von Operationen in einer bestimmten Reihenfolge auszuführen ist und nach der man alle Aufgaben eines gegebenen Typs lösen kann.“

Setzt man milesische Zahlzeichen in das indische Raster ein, so wird das Ergebnis falsch sein, oder das Verfahren kann nicht vollständig ablaufen. Ein Algorithmus löst folglich nur ein Problem, wenn er auf die benutzten Zeichen anwendbar ist. Bedenkt man dies, und soll der Algorithmenbegriff präziser beschrieben werden, so sind zuvor einige umgangssprachliche Begriffe zu erweitern [12, 31]:

> "Ein **Alphabet** ist eine endliche, geordnete Menge von Zeichen. Elemente eines Alphabets heißen auch **Buchstaben**. Ein **Wort** (über dem Alphabet A) ist eine endliche Folge von Zeichen (aus dem Alphabet A). Die Menge der Worte über dem Alphabet A bezeichnen wir mit A*."

Ein Problem wird folglich durch ein Wort dargestellt. Der Begriff Wort umfaßt dabei nicht nur Buchstaben- und Ziffernfolgen, sondern auch Anordnungen von Steinchen auf einem Rechenbrett, Stellungen von Zahnrädern und Rasten bei einer mechanischen Rechenmaschine und die Zustände der Speicher eines elektronischen Rechners. Wendet man auf solche Worte die entsprechender Vorschriften an, so erhält man nach einigen Operationen wieder ein Wort: das Ergebnis. Um von den, eventuell störenden Eigenheiten eines Menschen loszukommen, stelle man sich vor, daß eine Maschine, welche nicht näher beschrieben wird, die Operationen genau nach Vorschrift durchführt. So verstanden, wird das ein Problem darstellende Wort in die Maschine eingegeben, und die Maschine gibt das durch ein Wort dargestellte Ergebnis aus; deshalb spricht man von Eingabe- und Ausgabewerten. Verwendet die Maschine Zeichen, welche bei den Eingabewerten nicht vorkommen, so unterscheidet man zweckmäßigerweise ein Eingabealphabet und ein Arbeitsalphabet sowie gegebenenfalls ein Ausgabealphabet. Mittels dieser Begriffe ist uns eine weitere Präzisierung möglich [12, 33]:

> "Ein **Algorithmus** $\mathcal{A}$ besteht aus einem **Eingabealphabet I**, einem **Ausgabealphabet O** und einem **Arbeitsalphabet A** sowie aus einer **Vorschrift V**, mit deren Hilfe bei gegebenem Wort W $\in$ I* das **Ausgabewort W'** $\in$ O* gefunden wird: $\mathcal{A}$ = (V, I, A, O)."

Der Begriff Vorschrift kann ganz im Sinne einer Funktion bzw. Abbildung verwendet werden. Damit wird klar, daß einem Algorithmus nicht ein kompliziertes Verfahren zugrunde liegen muß, bereits eine einfache Formel genügt als Vorschrift. Auch sind die erlaubten Eingabewerte als Definitionsmenge und die Ausgabewerte als Bildmenge zu verstehen. Besteht die Eingabe aus n Wörtern, so ist die Vorschrift des Algorithmus in der Art einer n-stelligen Funktion zu verstehen. Man darf auch nicht annehmen, daß Algorithmen nur solche Verfahren sind, welche der Laie durchführen kann, aber nur der Fachmann versteht. Richtig dagegen ist, daß der Laie den Algorithmus richtig anwenden kann, ohne den mathematischen Hintergrund verstanden haben zu müssen. Aus der obigen Präzisierung läßt sich noch erkennen, daß die Vorschrift und das ablaufende Verfahren eine endliche Größe haben müssen, da W' sonst nicht auffindbar wäre. Dagegen soll es unerheblich sein, ob der Durchführung technisch oder gar physikalisch Grenzen gesetzt sind. Um die Zahl $1000^{1000^{1000}}$ in eine dezimale Darstellung zu bringen, ließe sich sicher ein Algorithmus finden. Dagegen wird die Menschheit wohl nie über genügend Zeit, Raum und Materie verfügen, um den Algorithmus durchführen zu können. Trotz seiner unermeßlichen Anzahl von Einzelschritten ist dieser Algorithmus aber grundsätzlich endlich und damit im hier verwendeten Sinne durchführbar.
Dagegen sind die Verfahren zum Dividieren und Wurzelziehen nicht grundsätzlich endlich. Um auch solche Verfahren als Algorithmen betrachten zu können, muß hier immer angegeben

werden, bei welcher Stelle nach dem Komma das Verfahren abbrechen soll. Da andererseits der EUKLIDische Algorithmus für negative ganze Zahlen nicht abbricht, während er bei positiven ganzen Zahlen den Anforderungen genügt, dürfen negative ganze Zahlen nicht als Eingabewerte auftreten. Zusätzlich muß sichergestellt sein, daß Maschine oder Mensch über die in der Vorschrift angegebenen Fertigkeiten verfügen (Lesen, Verschieben, Entscheiden, Rechnen, Schreiben). Auch soll das Verfahren nach richtiger Eingabe vollkommen selbständig ablaufen können, ohne jede Hilfe oder Beeinflussung von außen: zwei gleiche Maschinen in unterschiedlicher Umgebung könnten sonst bei gleicher Eingabe unterschiedliche Ergebnisse liefern. Aus gleichem Grunde darf es innerhalb der Vorschrift auch keinen Ermessensspielraum für den Durchführenden oder Zufälligkeiten im Ablauf geben, der Ablauf muß vollkommen determiniert sein.

1.3 Der EUKLIDische Algorithmus

Der EUKLIDische Algorithmus soll als allgemeines Verfahren hier kurz erläutert werden, da in 2.4 seine Existenz und Zuverlässigkeit vorausgesetzt wird. Auch wird damit das Wesen eines Algorithmus beispielhaft verdeutlicht. Bei der Beschreibung des EUKLIDischen Algorithmus folgen wir den Ausführungen von SCHEID [17, 68ff]. Der EUKLIDische Algorithmus bestimmt den größten gemeinsamen Teiler zweier natürlicher Zahlen a und b, kurz ggT(a, b) genannt, durch eine Kette von Divisionen mit Rest.

$$
\begin{array}{rcrcrcl}
\mathrm{a} & = & v_1 & \cdot & \mathrm{b} & + & r_2 \quad \text{mit } 0 < r_2 < \mathrm{b} \\
\mathrm{b} & = & v_2 & \cdot & r_2 & + & r_3 \quad \text{mit } 0 < r_3 < r_2 \\
r_2 & = & v_3 & \cdot & r_3 & + & r_4 \quad \text{mit } 0 < r_4 < r_3 \\
r_3 & = & v_4 & \cdot & r_4 & + & r_5 \quad \text{mit } 0 < r_5 < r_4 \\
\vdots & & & & & & \\
r_{n-3} & = & v_{n-2} & \cdot & r_{n-2} & + & r_{n-1} \quad \text{mit } 0 < r_{n-1} < r_{n-2} \\
r_{n-2} & = & v_{n-1} & \cdot & r_{n-1} & + & r_n \quad \text{mit } 0 < r_n < r_{n-1} \\
r_{n-1} & = & v_n & \cdot & r_n & + & 0
\end{array}
$$

Für das Beispiel a = 4081 und b = 2585 ergibt sich folgende Darstellung:

$$
\begin{array}{rcrcrcr}
4081 & = & 1 & \cdot & 2585 & + & 1496 \\
2585 & = & 1 & \cdot & 1496 & + & 1089 \\
1496 & = & 1 & \cdot & 1089 & + & 407 \\
1089 & = & 2 & \cdot & 407 & + & 275 \\
407 & = & 1 & \cdot & 275 & + & 132 \\
275 & = & 2 & \cdot & 132 & + & 11 \\
132 & = & 12 & \cdot & 11 & + & 0
\end{array}
$$

Der letzte von Null verschiedene Rest ist der gesuchte ggT:

$$ggT(a, b) = r_n \text{ und } ggT(4081, 2585) = 11$$

Dieses Verfahren ist nur tauglich, wenn

1. das Verfahren immer endet,
2. für r_n bei beliebigem $(a, b) \in N \times N$ gilt: $ggT(a, b) = r_n$

3. r_n für jedes Paar $(a, b) \in N \times N$ bestimmbar ist.

zu 1.: Für die Folge der Reste gilt $b > r_2 > r_3 > r_4 \ldots > r_{n-1} > r_n$, d.h. sie ist monoton fallend. Da alle $b, r_i \in N$ sind und N als kleinstes Element 1 enthält, muß das Verfahren spätestens hier abbrechen.
zu 2.: Der größte Teiler von r_n ist sicherlich r_n selbst. Da andererseits $r_{n-1} = r_n \cdot v_n$ gilt weiter $\mathrm{ggT}(r_n, r_{n-1}) = r_n$. Weiter folgt aus

$$\begin{array}{rcrcrcl} r_{n-2} & = & & v_{n-1} & \cdot & r_{n-1} & + & r_n \\ r_{n-2} & = & & v_{n-1} & \cdot & r_n \cdot v_n & + & r_n \\ r_{n-2} & = & r_n \cdot (& v_{n-1} & \cdot & v_n & + & 1) \end{array}$$

und $r_{n-1} = r_n \cdot v_n$
daß $\mathrm{ggT}(r_{n-1}, r_{n-2}) = r_n$ ist, falls $(v_{n-1} \cdot v_n + 1)$ und v_n keinen gemeinsamen Primfaktor haben.
Nimmt man einen solchen gemeinsamen Primfaktor an, so folgt aus der allgemeinen Regel $d \mid a + b$ und $d \mid a \Rightarrow d \mid b$, daß dieser Faktor gleich 1 sein müßte oder daß $v_n = 1$. Beides führt zum Widerspruch: 1 ist kein Primfaktor und aus $v_n = 1$ folgt $r_n = r_{n-1}$. Dieser Schluß wird nach oben fortgesetzt.
zu 3.: Die Division mit Rest ist in N immer durchführbar, mit 1. und 2. läßt sich r_n folglich immer bestimmen. Auch $a < b$ ist keine Einschränkung, es ergibt sich lediglich ein weiterer Rechenschritt.

$$\begin{array}{rcl} 2585 & = & 0 \cdot 4081 + 2585 \\ 4081 & = & 1 \cdot 2585 + 1496 \\ & \ldots & \end{array}$$

Das Verfahren wird bei [12, 13] wie folgt beschrieben:

1. Schritt: Notiere die beiden Zahlen.
2. Schritt: Führe die Division mit Rest durch.
3. Schritt: Ist der Rest gleich Null, so gib die zweite Zahl als ggT an und beende die Rechnung.
4. Schritt: Setze die zweite Zahl an die Stelle der ersten und den Rest an die Stelle der zweiten. Fahre mit dem zweiten Schritt fort.

1.4 Die Regula falsi

Führt ein Algorithmus nach endlich vielen Schritten immer zu einer eindeutig bestimmten Lösung, so nennt man dies einen abbrechenden Algorithmus. Kann aber die Lösung nur beliebig genau angegeben werden, so spricht man von nicht abbrechenden Algorithmen. Die Unterscheidung richtet sich nach dem allgemeinen Fall: z.B. ist das schriftliche Divisionsverfahren ein nicht abbrechender Algorithmus, da sich neben endlichen Dezimalbrüchen auch unendliche Dezimalbrüche als Ergebnisse einstellen.
Die Regula falsi gehört ebenfalls zu den nicht abbrechenden Algorithmen. Adam Riese (*1492, †1559), Bergbaubeamter und nebenberuflich Leiter einer Rechenschule und Verfasser eines weit verbreiteten Rechenbuches, beschrieb die Regula falsi im Jahr 1550 folgendermaßen [10, 65]:

„Ist eine Regel das man durch zwu falsche zaln die man der auffgab nach examiniert / die rechte zal haben mag / thu jhm also / Nim für dich eine zal der auffgab gemes, versuch die / komet zuvil bezeichne mit dem zeichen +, das is plus / sagt aber die zal der wahrheit zu wenig, so beschreib das mit dem zeichen -, das ist minus / Nach dem nim eine andere zal für dich, die examinir auch / Als dann setz beide zaln mit jren lügen / sagen die zaln der wahrheit beide zuviel / oder beide zu wenig / Nim ein lugen von der andern, das da bleibt ist dein teiler / darnach multiplizir kreutzweis ein falsche zal mit der anderen falschen zal lügen / nim auch voneinander, das da bleibt teil in vorgemachtem teiler, so hastu die rechte und wahrhafftige zal / sagt aber ein falsche zal der wahrheit zuuil, die ander zu wenig / Addir beide lügen behalt für deinen teiler / darnach multiplicir kreutzweis / addir auch und teile ab, so hastu berichtigung der frag.“

Durch den Wandel der Sprache fällt es nach 426 Jahren schwer, den Anweisungen des Rechenmeisters zu folgen. Andererseits ist es nicht uninteressant zu lesen, wie damals ein Algorithmus beschrieben wurde. In moderner Ausdrucksweise ist die Regula falsi ein Verfahren zur Berechnung einer Nullstelle einer reellwertigen Funktion $x \to f(x)$.

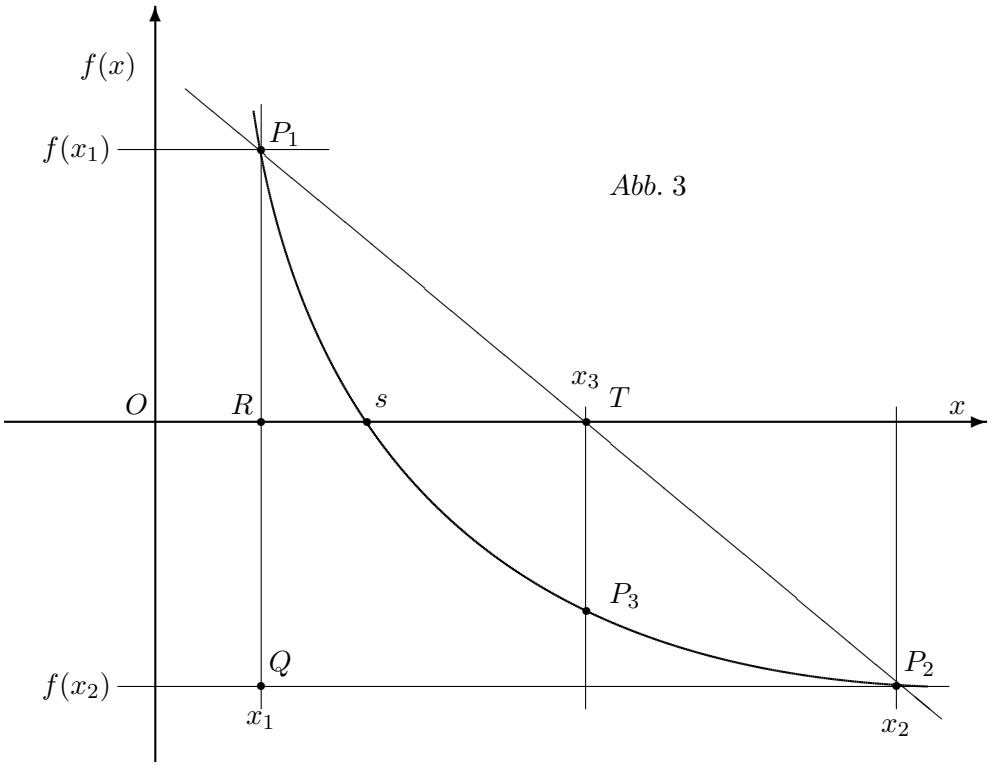


Abb. 3

Bei dem Verfahren wird die Nullstelle s durch eine Intervallschachtelung angenähert. Das erste Intervall sei durch die Näherungswerte x_1 und x_2 bestimmt. Ist die Funktion stetig und $f(x_l) \cdot f(x_2) < 0$, d.h. die zwei Funktionswerte haben verschiedene Vorzeichen, so gibt es im Intervall $]x_1, x_2[$ eine Nullstelle. Das Intervall sei so gewählt, daß es in $]x_1, x_2[$ nur eine Nullstelle und keinen Wendepunkt gibt. Der Schnittpunkt der Geraden $\overline{P_1P_2}$ mit der x-Achse ergibt den Näherungswert x_3. Rechnerisch gewinnt man x_3 durch die Formel

$$x_3 = x_1 - f(x_1) \cdot \frac{x_2 - x_1}{f(x_2) - f(x_1)}$$

Diese Formel erhält man über die Ähnlichkeit der beiden Dreiecke P_1RT und P_1QP_2 bzw. den 2. Strahlensatz. Sie behält ihre Gültigkeit auch dann, wenn der Graph f steigt oder $x_2 < x_1$ gilt.

Für den Ablauf des Verfahrens soll nun eine Vorschrift angegeben werden. Die Operationen werden schrittweise ausgeführt. Beginnt ein Schritt mit einer Bedingung, so darf nach den Anweisungen des Schrittes nur verfahren werden, wenn die Bedingung zutrifft. Durch Sprunganweisungen können Schritte wiederholt oder übersprungen werden.

1. Schritt: Fixiere die Rechenvorschrift der stetigen Funktion f, die Näherungswerte a und b sowie die positive Zahl ϵ, welche die obere Fehlerschranke angibt.
2. Schritt: Berechne $x = a - f(a) \cdot \frac{b - a}{f(b) - f(a))}$
3. Schritt: Falls $\mid f(x) \mid < \epsilon$, so ist x der Näherungswert für die Nullstelle und man beende das Verfahren.
4. Schritt: Falls $f(x) \cdot f(a) < 0$, so erhält b den Wert von x und man setzt das Verfahren mit dem 2. Schritt fort.
5. Schritt: a erhält den Wert von x und man setzt das Verfahren mit dem 2. Schritt fort.

Vorbedingungen und Verfahren stellen sicher, daß $\mid f(x) \mid$ immer kleiner wird. Da die Näherungswerte x auf die Nullstelle zulaufen, wird $\mid f(x) \mid$ den Wert ϵ einmal unterschreiten. Der 3. Schritt stoppt dann die Berechnung. Damit ist ein Näherungswert mit der gewünschten Genauigkeit gefunden.

1.5 Das Acht-Damen-Problem

Die bisher aufgezeigten Verfahren waren allesamt numerischer Natur und wurden als fertige Lösungen vorgestellt. Nunmehr soll ein nicht reinnumerisches Problem gestellt werden, dessen Lösung schrittweise erarbeitet und verbessert wird. Das Problem lautet [5, 26]:

> „Auf einem Schachbrett sollen 8 Damen so plaziert werden, daß sich je 2 Damen gegenseitig nicht bedrohen. Das Problem besitzt mindestens eine Lösung."

Gesucht ist der Algorithmus, der die Anzahl aller Lösungen ermittelt. Wem das Problem vorgelegt wird, der sucht zuerst erfahrungsgemäß kaum planmäßig, mehr zufällig nach den Lösungen. Ein derart zufälliges Vorgehen könnte nach sehr langer Zeit zum Erfolg führen, doch treten manche Lösungen bereits mehrfach auf, ohne daß man bis dahin mit Sicherheit alle Lösungen gefunden hat.
Da man 8 Figuren nur in endlich vielen Möglichkeiten auf 64 Feldern plazieren kann, müßte man nur nach einer Aufzählregel alle Möglichkeiten durchlaufen, um nach endlich vielen Schritten die obige Unsicherheit zu überwinden. Nach jedem Schritt wäre zu überprüfen, ob die Plazierungsmöglichkeit auch eine Lösung darstellt. Bevor man damit beginnt, sollte man den voraussichtlichen Aufwand abschätzen. Beim Lottospiel hat man 6 Kreuze auf 49 Felder zu verteilen und kommt dabei auf 13 983 816 Möglichkeiten. Beim Acht-Damen-Problem

wären es aber 4 426 165 368. Überprüft man pro Sekunde eine Stellung, so wäre man 140 Jahre lang ununterbrochen beschäftigt. Auch ein schneller Computer würde noch einige Tage brauchen. Eine wesentliche Verbesserung ergibt sich, wenn man von vornherein bedenkt, daß bei den Lösungen in jeder Spalte nur eine Dame sitzen darf. Die Berechnung kann bereits 264 mal schneller ablaufen, denn nun gibt es nur noch $8^8 = 16777216$ Möglichkeiten. Weiterhin darf auch in jeder Reihe nur eine Dame sein. Es bleiben noch „lächerliche“ $8! = 40320$ Möglichkeiten zu untersuchen. Damit hat man nur noch das 0,00001fache des ursprünglichen Rechenaufwandes. Unter diesen 8! Möglichkeiten sind jetzt noch jene auszuscheiden, die keine Lösungen sind, weil sich Damen über eine Diagonale bedrohen.

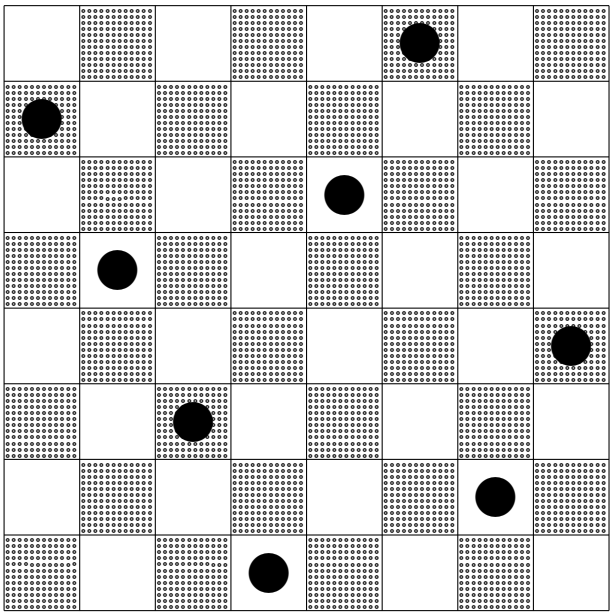


Abb. 4:
Eine Lösung des
Acht-Damen-Problems

Angenommen, man verfügt über eine Aufzählregel für alle 8! Möglichkeiten, wobei man die erste Dame in die erste Spalte setzt, die zweite Dame in die zweite Spalte usw.
Man braucht dann für die letzten beiden Damen keinen Platz zu suchen, wenn bereits in der 6. Spalte alle Felder bedroht werden. Dadurch verringern sich die Möglichkeiten noch weiter. Andererseits kann dann auch die Anzahl der Lösungen nicht groß sein. Eine Lösung liegt damit fast schon auf der Hand:

> i sei die Nummer einer Spalte, D_i die Dame der Spalte i und d_i die Nummer der Reihe, in welcher die Dame D_i steht. Die Stellung der Dame D_i ist durch die Koordinaten (d_i, i) festgelegt. D_1 wird in ihrer Spalte in die erste „erlaubte“ Reihe gesetzt: Feld $(1, 1)$. D_2 wird anschließend auf $(3, 2)$ gesetzt, da die Dame $D_1(1, 2)$ und $(2, 2)$ sperrt. Nun folgen die restlichen Damen. Findet die Dame D_i in ihrer Spalte i keinen unbedrohten Platz, so rückt D_{i-1} auf den nächsten freien Platz in ihrer Spalte vor. Nun versucht man erneut, D_i und die folgenden Damen zu setzen. Findet aber auch D_{i-1} keinen Platz, so muß D_{i-2} versetzt werden. Läßt sich auch für D_8 ein unbedrohtes Feld finden, so hat man eine Lösung erhalten. Man notiert diese Lösung und fährt mit dem Verfahren fort. Das Verfahren endet, wenn D_1 aus d_8 verschoben werden soll.

Die exakte Beschreibung fällt wesentlich formaler und leider auch unanschaulicher aus:

1. Schritt: Man setze $a = 0$ und $i = 1$.

2. Schritt: Man setze $d_i = 1$.

3. Schritt: Falls für ein $k = 1, 2, \cdots, i-1$
   $d_k = d_i$ (d.h. gleiche Zeile)
   oder $\mid d_i - d_k \mid = i - k$ (d.h. gleiche Diagonale), so springe zum 6. Schritt.

4. Schritt: Falls $i = 8$ (Lösung!), so erhöhe a um 1, erniedrige i um 1 und springe zum 6. Schritt.

5. Schritt: Erhöhe i um 1 und springe zum 2. Schritt.

6. Schritt: Erhöhe d_i um 1 (neuer Platz für D_i). Falls $d_i \leq 8$ so springe zum 3. Schritt.

7. Schritt: Erniedrige i um 1 . Falls $i = 0$, so beende das Verfahren.

8. Schritt: Springe zum 6. Schritt.

Der Algorithmus wird dadurch kompliziert, weil man von der Möglichkeit, nur eine Dame in eine Reihe zu setzen, nicht unmittelbar Gebrauch machen kann. Man hat immer erst zu prüfen, ob in der soeben besetzten Reihe bereits eine Dame ist. Andererseits leuchtet unmittelbar ein, daß bei diesem Verfahren keine Stellung zweimal durchlaufen wird.
Eine Halbierung des Rechenaufwandes erreicht man, wenn man bedenkt, daß die zur Lösung L_i symmetrische Stellung $\overline{L_i}$ ebenfalls Lösung ist. Reihen, Spalten und Diagonalen des Schachbretts bilden eine Menge G von ausgezeichneten Geraden. Die Damen bilden eine Menge P von ausgezeichneten Punkten auf diesen Geraden. Für eine Lösung L_i gilt: Je zwei Punkte P_j liegen nicht auf einer Geraden aus G. Durch die Geraden- und Inzidenztreue der Spiegelung bleibt dieser Umstand für $\overline{L_i}$ erhalten. G geht in G über. Damit ist auch $\overline{L_i}$ eine Lösung. Somit kann man das Verfahren abbrechen, wenn D_1 von (4, 1) nach (5, 1) wechselt. Man verdoppelt einfach die Anzahl der inzwischen gefundenen Lösungen. Eine doppelt gezählte Lösung kann es nicht geben, da D_1 bisher nur in d_1, d_2, d_3 oder d_4 war und die hier verwendete Spiegelung die Dame D_1 somit nur in d_5, d_6, d_7 oder d_8 bringen kann. Neue Lösungen kann es nicht geben, denn nach obigem Gedankengang wären diese oder ihr Spiegelbild bereits ermittelt worden.
Anmerkung: Das Acht-Damen-Problem läßt sich in ein n-Damen-Problem für ein Brett mit n Reihen und n Spalten verallgemeinern. Der Algorithmus ließe sich entsprechend abwandeln. Für $n = 1$ ergibt das Problem wenig Sinn. Ansonst erhält man für a folgende Werte:

n	2	3	4	5	6	7	8
a	0	0	2	10	4	40	92

Eine einfache Formel läßt sich für das Problem anscheinend nicht finden. Für $n = 8$ ist die Lösung per Hand in wenigen Stunden zu ermitteln, bei $n = 12$ sind bereits mehrere Wochen anzusetzen. Ein Beispiel für eine noch größere Beschleunigung des Rechenvorganges findet man bei Trachtenbrot [19, 13ff] und Menzel [12, 27ff] in Form des Fünfzehnerspiels.

1.6 Die Entwicklung des Algorithmenbegriffs

1.6.1 Historisches

Allgemeine Verfahren zur Bewältigung von Problemen sind so alt wie die Mathematik selbst. So war der sogenannte Satz des Pythagoras und damit verbundene Problemlösungen schon

vor 4000 Jahren in Babylonien und Ägypten bekannt. Den EUKLIDischen Algorithmus gab es schon 300 v. Chr. und das Sieb des ERATOSTHENES seit etwa 200 v. Chr. Das Wort „Algorithmus“ ist wesentlich jünger. Wie die Vorsilbe al andeutet, hat es seine Wurzel im Arabischen. Es ist von AL CHWARIZMI, dem Namen eines usbekischen Mathematikers, abgeleitet. MUHAMED IBN MUSA AL CHWARIZMI[1] wirkte um 810 bis 840 als Astronom am Hofe AL MAMUNS, des Sohnes HARUN AL RASCHIDS, in Bagdad und verfaßte das Werk „Hisâb aljabr w'al-muqâbalah“. Man könnte es als ein umfassendes Lehrbuch über die von Indern und Arabern erarbeitete Gleichungslehre bezeichnen. Das Werk enthielt neben Erbteilungsaufgaben auch eine systematische Theorie der quadratischen Gleichungen. „aljabr“ ist die Wurzel des abendländischen Begriffs „Algebra“ [11, 1]. Anfänglich war ein Algorithmus also ein Verfahren zur Umformung und damit Lösung von Gleichungssystemen. Später wird Algorithmus „zum Inbegriff jeder Rechenvorschrift schlechthin“ [11, 1]. Die arabischen Lehren beeinflußten wiederum den Spanier RAYMUNDUS LULLUS (ca. 1235 bis 1315) zu seiner „Ars magna“. Zwar sind die Ausführungen LULLUS nicht allzu hoch zu schätzen, wohl aber die ihnen zugrunde liegende Idee: Ein allgemeines Verfahren sollte auf kombinatorischer Grundlage alle „Wahrheiten“ auffinden lassen. Diese Idee blieb nicht ohne Wirkung.
Seine algebraischen Algorithmen zur Auflösung von Gleichungen sah GEROMONO CARDANO (*1501, †1576) „ganz im Zeichen der lullischen Kunst, wie der Titel seines Werkes verrät: Artis magnae seu de regulis algebraicis liber unus“ [6, 28]. Für RENÈ DESCARTES (*1598, †1650) war durch die Verwendung von Koordinaten in der Geometrie der Weg frei für die algebraische und damit algorithmische Lösung aller geometrischen Probleme. Darin lag die Neuerung, denn Koordinaten wurden bereits im Altertum benutzt. Er irrte aber, wenn er alle algebraischen Probleme für algorithmisch lösbar hielt. GOTTFRIED WILHELM LEIBNIZ (*1646, †1716) bemühte sich nicht nur um die Entwicklung einzelner Algorithmen, sondern auch „um die Einsicht in das Wesen des Algorithmus“ selbst [6, 29]. In dem Wirrwarr der lullischen Begriffe trennte und unterschied er eine *ars inveniendi* (heute Erzeugungsverfahren) und eine *ars iudicandi* (heute Entscheidungsverfahren). Trotz tieferer Einsicht blieb LEIBNIZ von LULLUS noch so weit beeinflußt, daß er an die Existenz einer allumfassenden Methode glaubte, welche die Lösung eines jeglichen Problems ermöglichte; nicht nur solche von mathematischer oder logischer Natur, auch die Moral, die Physik, die Medizin und die Metaphysik sollten erfaßt werden [15, 5f]. Er erkannte klar, daß man allgemeine Verfahren einer Maschine anvertrauen kann und zeigte dies durch Konstruktion und Bau einer Rechenmaschine für die vier Grundrechnungsarten. Dagegen war der Bau einer logischen Maschine mit den Mitteln der Zeit nicht zu leisten. LEIBNIZ Idee dazu, der „Calculus ratiocinator“, ist der Vorläufer des modernen Logikkalküls [2]

1.6.2 Moderne Ansätze

Arbeiten über die Logik und zur mathematischen Grundlagenforschung zogen ab der Mitte des vorigen Jahrhunderts neue Erkenntnisse über Algorithmen nach sich. GEORGE BOOLE (*1815, †1864) behandelte in seinem 1847 erschienenen Buch „The mathematical analysis of

[1] MESCHKOWSKI [13] gibt AL CHWARIZMI als Bibliothekar an, der etwa von 780 bis 850 lebte und dessen voller Name Abu' Abdallah Muhammad ibn Musa al-Huwarismi lautet.

[2] Kalkül wird im allgemeinen synonym für Algorithmus gebraucht. Das Wort selbst ist vom altrömischen „calculi“ abgeleitet, den Kalksteinchen, mit denen man auf den Rechenbrettern hantierte. Eine verbindliche Festlegung oder Trennung der Begriffe besteht nicht. In der Literatur läßt sich eine geringe Tendenz erkennen, daß man ein formalisiertes Regelsystem zur schrittweisen Umwandlung von Zeichenreihen als Kalkül bezeichnet, während ein Algorithmus auch durch eine umgangssprachliche Beschreibung gegeben ist.

logic, being an essay towards a calculus of deductive reasoning“ und im 1854 erschienenen Hauptwerk „An investigation of the laws of thought, in which are founded the mathematical theories of logic and probabilities“ die Logik mit algebraischen Methoden: eine „Algebra der Logik“ war geschaffen.
Im Zuge von Axiomatisierungsbestrebungen in vielen Bereichen der Mathematik erstellte Giuseppe Peano (*1858, †1932) im Jahr 1891 das heute bekannteste Axiomensystem für die natürlichen Zahlen. Er verwendete eine besondere Symbolik, behielt aber die Struktur natürlicher Sprachen bei.
Durch eine 1879 vorgestellte Begriffsschrift hat sich Gottlob Frege (*1848, †1925) „das größte Verdienst um die Aufstellung eines Logikkalküls“ [10, 210] erworben. Die unzweckmäßige Symbolik verhinderte aber, daß sich die Gedanken Freges sofort durchsetzten. Dies erreichten erst Bertrand Russell (*1872, †1970) und Alfred North Whitehead (*1861, †1947) durch ihre „Principia Mathematica“. Das monumentale Werk erschien dreibändig zwischen 1910 und 1913. Darin wird nachgewiesen, „daß sich große Teile der Mathematik mit Hilfe eines Logikkalküls deduzieren lassen“ [7, 30]. 1899 erschien von Hilbert „Grundlagen der Geometrie“, worin ein vollständiges Axiomensystem der ebenen Euklidischen Geometrie entwickelt wurde. Ein Axiomensystem ist nach Hilbert erst brauchbar, wenn es die Unabhängigkeit, Vollständigkeit und Widerspruchsfreiheit seiner Sätze gezeigt hat. Ein befriedigender Beweis für die Widerspruchsfreiheit, „nach Hilbert das Merkmal mathematischer Existenz“ [10, 216] ließ sich aber nicht führen. Lediglich einen Beweis zur relativen Widerspruchsfreiheit konnte er angeben: „Ist die Theorie der reellen Zahlen widerspruchsfrei, so auch die axiomatische Euklidische Geometrie“ [1, 233] „Eine formalisierte Theorie heißt widerspruchsfrei, wenn es nicht möglich ist, einen Satz S und außerdem seine Negation ¬ S abzuleiten“ [13, 283]. Sollte es Bereiche geben, in denen sich Vollständigkeit und Widerspruchsfreiheit gegenseitig ausschließen - neuere Untersuchungen zeigen dies -, so wäre wohl eher auf die Vollständigkeit zu verzichten. „Eine formalisierte Theorie heißt vollständig, wenn jeder Satz der Theorie bewiesen oder widerlegt werden kann“ [13, 279]. In diesem Zusammenhang wurde von Hilbert das allgemeine Entscheidungsproblem formuliert. „Dieses Problem kann als die Frage verstanden werden, ob bei einem beliebigen (endlichen) Axiomensystem $\mathcal{AS}$, das sich in der Symbolsprache einer bestimmten Logik anschreiben läßt, für eine beliebige, ebenfalls in dieser Sprache darstellbare Aussage A entscheidbar ist, ob A Folgerung aus $\mathcal{AS}$ ist oder nicht. Die Existenz eines Entscheidungsverfahrens für diese Logik bedeutet dabei, daß man nicht nur einen Kalkül hat, mit dem genau die Menge aller Folgerungen aus einem beliebigen in der Logiksprache darstellbaren Axiomensystem $\mathcal{AS}$, der Reihe nach hergeleitet ('aufgezählt') werden kann (einen sog. vollständigen Logik-Kalkül oder mit einem Terminus von LEIBNIZ: eine 'ars inveniendi'), sondern ebenso einen Kalkül für die Aufzählung der Menge aller Ausdrücke der Logiksprache, die Nichtfolgerungen von a sind“ [1, 238]. Zum Problemkreis der Entscheidbarkeit und Aufzählbarkeit gesellt sich noch das Problem der Berechenbarkeit in Gestalt der berechenbaren Funktion. Die hierfür richtungsweisende Arbeit „Begründung der elementaren Arithmetik durch die rekurrierende Denkweise ohne Anwendung scheinbarer Veränderlicher mit endlichem Ausdehnungsbereich“ wurde 1923 von dem norwegischen Mathematiker Thoralf Skolem (*1887, †1963) veröffentlicht. Seine im heutigen Sinne primitivrekursiven Funktionen wurden 1934 von Kurt Gödel (*1906) nach der Idee von Jacques Herbrand (*1908, †1931) zum Konzept der allgemein-rekursiven Funktionen entwickelt; nachdem Wilhem Ackermann (*1896, †1962) schon 1928 ein Beispiel einer berechenbaren, aber nicht primitiv-rekursiven Funktion gegeben hatte. Von GÖDEL war bereits 1931 das epochemachende Werk „Über formal unentscheidbare Sätze der Prin-

cipia Mathematica und verwandter Systeme“ erschienen. Darin wird gezeigt, „daß in jedem System Sätze existieren, deren Wahrheit mit den Mitteln des Systems nicht entschieden werden kann“ [10, 216] und „daß die Widerspruchsfreiheit eines Systems mit den Mitteln des Systems selbst nicht nachweisbar ist“ [10, 217]. Dies heißt, „daß gewisse mathematische Probleme nicht mit den Algorithmen einer präzise definierten Klasse von Algorithmen lösbar sind“ [8, 2]. Nach den allgemein-rekursiven Funktionen kamen um 1936 weitere Präzisierungen des Algorithmenbegriffs: die μ-kursiven Funktionen durch Gödel und Stephen Cole Kleene (*1909), die λ-definierbaren Funktionen durch Kleene und Alonzo Church (*1903) und die Beschreibung einer automatisch arbeitenden Maschine, heute Turingmaschine genannt, durch den englischen Logiker Alan Mathison Turing (*1912, †1954) und den amerikanischen Logiker Emil L. Post. Post und Turing kamen unabhängig voneinander zu ganz ähnlichen Beschreibungen.
Die erste direkte Präzisierung des Algorithmenbegriffs lieferte Andrej Andrejewitsch Markov (*1903), Sohn des gleichnamigen russischen Mathematikers, in der 1951 erschienenen „Theorie der Algorithmen“. Seine heute als Markovsch bezeichneten Algorithmen nannte er selbst natürliche Algorithmen. Alle diese formalen Präzisierungen haben sich als äquivalent erwiesen. Dazu gehören auch noch die Graphenschemata von R. Péter von 1958. Sicher ist, daß alle Algorithmen im präzisen Sinn auch Algorithmen im intuitiven Sinne sind. Dagegen ist es lediglich eine Erfahrungstatsache, daß sich jeder Algorithmus im intuitiven Sinne durch einen Algorithmus im präzisen Sinne beschreiben läßt. Ein Beweis dafür ist nicht möglich, da der intuitive Algorithmenbegriff nicht exakt zu fassen ist. Solange das Gegenteil nicht bewiesen ist, kann die Churchsche These als richtig angesehen werden [3, 5]: „Der intuitiv gegebene, allgemein gebräuchliche Begriff der berechenbaren arithmetischen Funktion ist identisch mit dem exakt definierten Begriff der allgemeinrekursiven Funktion.“
Church formulierte sie 1936 in seiner Arbeit „An Unsolvable Problem of Elementary Number Theory“. Hier bewies er die Unlösbarkeit des Entscheidungsproblems für die Prädikatenlogik. Weitergehend konnte nun die Unvollständigkeit der Prädikatenlogik der zweiten Stufe sowie die Unentscheidbarkeit und damit Unvollständigkeit der Arithmetik gezeigt werden.

1.6.3 Anwendungen

Nachdem die Grundlagen erarbeitet waren, konnte man daran gehen, eine Reihe von ungelösten Problemen mit den neuen Erkenntnissen zu bearbeiten. 1914 hatte der norwegische Mathematiker A. Thue die „Probleme über Veränderungen von Zeichenreihen nach gegebenen Regeln“ veröffentlicht. Er beschrieb darin das Wortproblem für gewisse Halbgruppen. 1946 und 1947 konstruierten Markov und Post unabhängig voneinander konkrete Beispiele von Halbgruppen, in denen das Wortproblem unlösbar ist. 1955 schließlich bewies Pjotr Sergewitsch Novikov die „algorithmische Unlösbarkeit des Wortproblems in der Gruppentheorie“. Für diese Arbeit erhielt er 1957 den Leninpreis.
Auf die algorithmische Unlösbarkeit des zehnten Hilbertschen Problems wurde bereits hingewiesen. Die Regeln zur Umformung von endlichen Zeichenreihen wurden so weit entwickelt, daß man heute nicht nur von Kalkülen spricht, sondern bereits von Sprachen. So werden die 1943 in „Formal Reductions of the General Combinatorical Decision Problem“ von Post beschriebenen Kalküle heute sowohl Postsche Kalküle als auch Postsche Sprachen genannt. Eine besondere Klasse der Postschen Sprachen sind die formalen Sprachen von N. Chomsky. Er entwickelte sie seit 1955 im Rahmen linguistischer Untersuchungen. Die Chomskyschen

formalen Sprachen bilden ein Modell für die natürlichen Sprachen. Mittels solcher Modelle wird es möglich, Programmiersprachen für Rechenanlagen zu entwickeln und automatische Übersetzer zu konstruieren. Texte von Programmiersprache in Maschinensprache zu übersetzen ist dabei wesentlich einfacher als die Übersetzung natürlicher Sprachen. Letzteres ist zwar bereits möglich, doch die Ergebnisse sind nicht ganz befriedigend.
Betrachtungen über TURINGmaschinen und konkrete Rechenanlagen führten ab 1955 zur Automatentheorie.
Anhand des Acht-Damen-Problems wurde in dieser Arbeit veranschaulicht, wie man durch zweckmäßige Überlegungen Rechenaufwand sparen kann. Mathematiker und Informatiker faßten Rechenzeit und Speicherplatz als die Kompliziertheit eines Algorithmus zusammen und entwickelten seit 1965 die Komplexitätstheorie. Ergebnisse liegen bereits vor, die Theorie kann aber noch nicht als abgeschlossen angesehen werden.

Kapitel 2

Die TURINGmaschine

Die vorläufige Präzisierung des Algorithmenbegriffs kann nicht als Definition verwendet werden, da in ihr der Begriff „Vorschrift“ einer genaueren Beschreibung bedarf. Aus der Vielzahl der bereits genannten Möglichkeiten soll nun die TURINGmaschine als Beispiel für eine mathematisch einwandfreie und handhabbare Fassung des Algorithmenbegriffs verwendet werden.

2.1 Aufbau einer TURINGmaschine

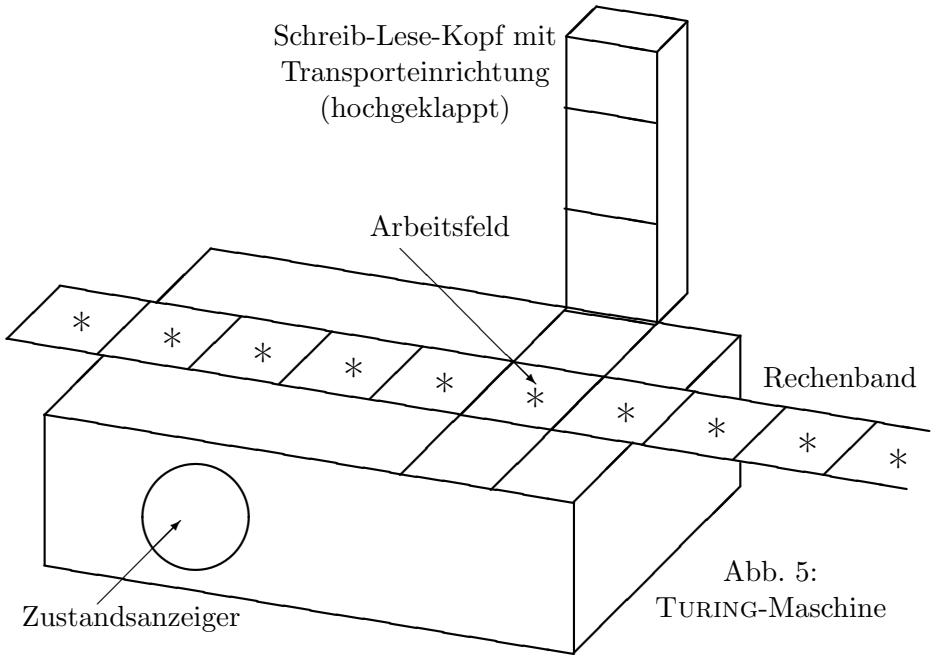


Abb. 5:
TURING-Maschine

Eine TURINGmaschine, im folgenden kurz Tm genannt, ist keine konkret vorhandene und operierende Maschine, sondern ein mathematisches Denkmodell. Aus Gründen der Anschaulichkeit wird die Tm aber in einer technisch durchaus herstellbaren Form beschrieben. Der praktische Wert einer Tm wäre gering, da bei ihr selbst eine elementare Addition noch aus vielen Einzelschritten besteht. Die mit ihr gewonnenen theoretischen Erkenntnisse sind dage-

gen von großem Nutzen.

Die Tm besitzt zwei Speicher. Der interne Speicher ist ein Operationswerk, welches endlich vieler diskreter Zustände fähig ist. Der externe Speicher ist das Rechenband, kurz Band genannt. Hierauf werden die Eingabe-, Zwischen- und Ausgabewerte geschrieben. Das Band ist in einzelne Felder aufgeteilt; in jedem Feld kann höchstens ein Buchstabe stehen. Das Band ist nur einseitig begrenzt, damit wird eine einfache Numerierung der Felder möglich, andererseits können beliebig viele und beliebig lange Wörter auf das Band gedruckt werden. Allerdings dürfen es nur endlich lange und endlich viele Wörter sein, ansonsten würde die Berechnung nicht enden. Um Buchstaben lesen und schreiben zu können, besitzt die Tm einen Schreib-Lese-Kopf. Das Feld des Bandes unter dem Schreib-Lese-Kopf heißt Arbeitsfeld. Es soll nie gleichzeitig zwei Arbeitsfelder geben. Auf den Befehl r macht die Transporteinrichtung das Feld rechts vom Arbeitsfeld zum neuen Arbeitsfeld. Sie kann weiterhin die Befehle l (links) und s (Stopp) ausführen.
Der Ablauf einer Berechnung läßt sich folgendermaßen beschreiben: Die Tm befindet sich im Anfangszustand. Das beschriftete Band wird eingelegt und die Tm startet, wenn der Schreib-Lese-Kopf heruntergeklappt wird. Durch ihren Zustand und den Buchstaben im Arbeitsfeld ist das weitere Verhalten der Tm eindeutig bestimmt: Der Schreib-Lese-Kopf druckt einen neuen Buchstaben in das Arbeitsfeld, die Transporteinrichtung tritt in Aktion, und das Operationswerk nimmt einen neuen Zustand an. Im neuen Zustand liest die Tm über den Schreib-Lese-Kopf im Arbeitsfeld den neuen Buchstaben ab, und der obige Vorgang wiederholt sich. Ist die Berechnung beendet, so erfolgt die Stoppanweisung: dabei klappt der Schreib-Lese-Kopf hoch und gibt den Blick auf das Arbeitsfeld frei. Da das Band einseitig begrenzt ist, kann die Berechnung auch wegen Bandüberschreitung abbrechen. Auch hier klappt der Schreib-Lese-Kopf hoch, doch wirft die Transporteinrichtung dann das Band ganz aus der Maschine, damit Stoppanweisung und Bandüberschreitung klar zu unterscheiden sind.
Es muß noch gesagt werden, daß ein „neuer“ Zustand auch der alte sein kann. Gleiches gilt für Arbeitsfeld und Buchstaben.
Zur Vereinfachung wird vereinbart, daß die Tm beim Start immer auf das erste freie Feld nach dem zu bearbeitenden Wort angesetzt wird (d.h. rechts davon). Freie Felder werden durch das leere (uneigentliche) Symbol * angezeigt. In der Umgebung des zu bearbeitenden Wortes sollen keine weiteren Wörter stehen, d.h. sie sind mit * bedruckt. Nach der Berechnung bleibt die Tm hinter dem bearbeiteten Wort (Ergebnis) stehen.

2.2 Die TURINGtafel

Das Verhalten der Tm wird durch ein Programm bestimmt das in der TURINGtafel, kurz Tt genannt, festgelegt ist. Die Zeile

$$q_0 \quad * \quad ! \quad r \quad q_1$$

$q_0 * ! r q$, einer Tt stellt eine bedingte Anweisung dar und liest sich wie folgt: Befindet sich das Operationswerk im Zustand q_0 und auf dem Arbeitsfeld der Buchstabe *, so wird in das Arbeitsfeld der Buchstabe !\[1\] gedruckt, das Arbeitsfeld wird in das nächste rechte Feld verlegt, und das Operationswerk nimmt den Zustand q_1 an.

[1] Umgangsprachlich gilt ! nicht als Buchstabe, dieses Zeichen wird fortan vereinfachend als Strich bezeichnet.

Durch eine weitere Zeile erhält man eine Tt, mittels welcher die Tm den Nachfolger einer natürlichen Zahl berechnet.

q_0	$*$	!	r	q_1
q_1	$*$	$*$	s	q_0

Unter Einschluß der Null stellt man die natürliche Zahl m durch eine Folge von m+l Strichen auf dem Rechenband dar. Die Zahlen $0, 1, 2, 3, \cdots$ haben folglich die Darstellung !, !!, !!!, !!!!, $\cdots$ auf dem Rechenband. Setzt man die Tm nach Vereinbarung im Feld hinter den m+1 Strichen an, so druckt sie einen Strich, geht nach rechts und nimmt den Zustand q_1 an. Im neuen Feld kann die Tm nur auf * stoßen, denn die Umgebung soll nach Vereinbarung von anderen Zeichen frei sein. Nach Zeile 2 der Tt ist nun * zu drucken (oder die Tm läßt * stehen), die Tm stoppt und geht in den Zustand q_0 über. Auf dem Band stehen vor dem Arbeitsfeld jetzt m+2 Striche. Steht auf dem Rechenband eine Folge von 5 Strichen, und die Tm startet mit Feld 6 als Arbeitsfeld,

$*$	!	!	!	!	!	**$*$**	$*$	$*$	$*$
0	1	2	3	4	5	6	7	8	9

Abb. 6a

so hat das Band folgendes Aussehen, wenn der Befehl $q_0 * ! \, r \, q_1$ ausgeführt worden ist.

$*$	!	!	!	!	!	!	**$*$**	$*$	$*$
0	1	2	3	4	5	6	7	8	9

Abb. 6b

Arbeitsfeld ist 7 und der Zustand ist q_1. Nach der Ausführung des Befehls $q_1 \; * \; * \, s \, q_0$ ist die Berechnung beendet und auf dem Band stehen $5 + 1 = 6$ Striche. Das Arbeitsfeld ist jeweils dick eingerahmt. Liest die Tm im Arbeitsfeld den Buchstaben ! so kann sie darauf nicht reagieren, weil die Tt keine entsprechende Zeile hat. Wegen der getroffenen Vereinbarung kann dieser Fall auch gar nicht auftreten. Es schadet aber andererseits nicht, wenn man die Tt für diese Fälle erweitert. Besteht das Arbeitsalphabet nur aus dem Buchstaben ! und aus dem leeren Zeichen * und kann das Operationswerk nur die Zustände q_0 und q_1 annehmen, so sind mit der folgenden Tt alle Fälle berücksichtigt.

q_0	$*$	!	r	q_1
q_0	!	!	r	q_0
q_1	$*$	$*$	s	q_0
q_1	!	$*$	s	q_0

Die Zeilen 1 und 3 sind bereits bekannt. Würde als erstes Arbeitsfeld - entgegen der Vereinbarung (060) ein Feld innerhalb eines Wortes gewählt, so bewirkt die Zeile 2, daß das Arbeitsfeld schrittweise nach rechts an das Ende des Wortes verlegt wird. Auch die Zeile 4 wird nur in einem nicht vereinbarungsgemäßen Fall durchlaufen. Steht nach dem zu bearbeitenden Wort nur ein einziges leeres Symbol * und gleich anschließend ein weiteres Wort, so wird mittels Zeile 4 das erste Zeichen dieses Wortes durch * ersetzt, und die Tm stoppt. Die Berechnung wurde damit ordnungsgemäß ausgeführt, aber auch die weitere Bandinschrift verändert. Nach dem Stoppbefehl kann ohne weiteres sogleich der nächste Nachfolger berechnet werden. Durch die Verallgemeinerung der bisherigen Ausführungen ist die Definition der TURINGtafel möglich:

Gegeben sei der mit a_0 bezeichnete leere Buchstaben und das Alphabet $A = \{a_1, a_2, a_3, \cdots, a_N\}$ mit $N \geq 1$ sowie eine Menge $Q = \{q_0, q_1, q_2, \cdots, q_M\}$ mit $M \geq 0$. Die q_i heißen Zustände, die a_k Buchstaben. Eine TURINGtafel Tt besteht aus einer fünfspaltigen und $(M+1) \cdot (N+1)$-zeiligen Matrix. Eine einzelne Zeile hat die Form

$q_i \; a_k \; a_{i,k} \; v_{i,k} \; q_{i,k}$

Die beiden ersten Spalten bestehen aus den $(M+1) \cdot (N+l)$ verschiedenen Paaren $q_i \, a_k$ mit $0 \leq i \leq M$ und $0 \leq k \leq N$. In der dritten Spalte steht der durch q_i und a_k bestimmte Buchstabe $a_{i,k} \in A$. In der vierten Spalte steht entweder r, l oder s (für rechts, links oder Stopp). Die fünfte Spalte beinhaltet den durch q_i und a_k bestimmten Folgezustand $q_{i,k} \in Q$.

Grundsätzlich sind Tm und Tt streng zu unterscheiden. In gleicher Weise, wie man eine Verordnung von dem nach dieser Verordnung arbeitenden Beamten unterscheidet. Doch auch hier gibt es Gleichsetzungen.
In der Literatur wird die Tt teilweise durch eine vierspaltige Matrix festgelegt. Die $v_{i,k}$ tauchen dann in der dritten Spalte auf. Die Tt wird dadurch in den meisten Fällen länger.
Mit Vorteil verwendet man für die Tt Tabellenform. Die $q_i \in Q$ kommen in die Eingangsspalte, die $a_k \in A$ in die Eingangsreihe. Das zu q_i und a_k passende $a_{i,k} \, v_{i,k} \, q_{i,k}$ findet sich dann im Feld in der i-ten Reihe und k-ten Spalte. Die Tt für die Nachfolgerfunktion von erhielte dann die Form nach Abb. 7a. Diese Darstellung verkürzt sich und wird übersichtlicher, wenn man $a_{i,k}$ und $q_{i,k}$ nicht notiert, falls $a_k = a_{i,k}$ oder $q_i = q_{i,k}$; Abb. 7b zeigt dies.

	$*$	$!$
q_0	$! \, r \, q_1$	$! \, r \, q_0$
q_1	$* \, s \, q_0$	$* \, s \, q_0$

Abb. 7a

	$*$	$!$
q_0	$! \, r \, q_1$	r
q_1	$s \, q_0$	$* \, s \, q_0$

Abb. 7b

2.3 Beispiele für TURINGmaschinen

2.3.1 Addition von Strichzahlen

Die beiden natürlichen Zahlen m und n sind zu addieren. Dazu setzt man mit einem Feld Abstand $m+1$ und $n+1$ Striche auf das Band. In das Feld zwischen den Strichfolgen wird der leere Buchstabe * gesetzt. Wie schon bei der Nachfolgerfunktion verwenden wir auch hier einen Strich zusätzlich, um auch die Null darstellen zu können. Für die Aufgabe $3 + 2$ muß das Band dann vor der Berechnung das Aussehen nach Abb. 8a haben.

$*$	$!$	$!$	$!$	$!$	$*$	$!$	$!$	$!$	$*$	$*$
0	1	2	3	4	5	6	7	8	9	10

Abb. 8a

Nach der Berechnung müssen bei gestoppter Maschine 5+1 Striche vor dem Arbeitsfeld auf dem Band stehen.

Abb. 8b

	$*$	!
q_0	$l\, q_1$	$*\, s$
q_1	l	$*\, l\, q_2$
q_2	$!\, r\, q_3$	$l\, q_2$
q_3	$l\, q_0$	r

Dazu muß nach dem Start der Strich aus Feld 8 nach 5 gebracht werden. Nach einem Rechtslauf bis 8 wird in 7 der Buchstabe ! durch * ersetzt, und die Tm stoppt. Die Streichung des letzten Striches ist notwendig, da ansonsten $(3 + 1) + (2 + 1) = (5 + 2)$ Striche auf dem Band vor dem Arbeitsfeld stehen würden. Die Tafel dieser Additionsmaschine ist in nebenstehender Abb. 9 dargestellt.
In der nachfolgenden Darstellung wird die Berechnung in einzelnen Schritten aufgezeigt. Durch Anwendung des Befehls von Schritt 1 auf die Bandinschrift von Schritt 1 entsteht die Bandinschrift von Schritt 2 usw. Das Arbeitsfeld ist jeweils eingerahmt.

Schritt	Bandinschrift	entspr. Befehl der Tt
1	* ! ! ! ! * ! ! ! ⊛	q_0 * * l Q_1
2	* ! ! ! ! * ! ! ⊡ *	q_1 ! * l Q_2
3	* ! ! ! ! * ! ⊡ * *	q_2 ! ! l Q_2
4	* ! ! ! ! * ⊡ ! * *	q_2 ! ! l Q_2
5	* ! ! ! ! ⊛ ! ! * *	q_2 * ! r Q_3
6	* ! ! ! ! ! ⊡ ! * *	q_3 ! ! r Q_3
7	* ! ! ! ! ! ! ⊡ * *	q_3 ! ! r Q_3
8	* ! ! ! ! ! ! ! ⊛ *	q_3 * * l Q_0
9	* ! ! ! ! ! ! ⊡ * *	q_0 ! * s Q_0
10	* ! ! ! ! ! ! ⊛ * *	

Es zeigt sich, daß der Fall $q_1 *$ bei der Berechnung nicht vorkommt. Dies wäre auch nur möglich, wenn entgegen der Vereinbarung das leere Zeichen $*$ vor dem ersten Arbeitsfeld stehen würde. Durch den Befehl q_1 * * l Q_1 läuft die Tm so lange nach links, bis sie entweder etwas zur Berechnung findet oder das Bandende erreicht ist. In jedem Fall stoppt die Tm nach endlich vielen Schritten. Es stellt sich die Frage, ob die Tm nach der Tt von Abb. 9 für alle natürlichen Zahlen das richtige Ergebnis liefert. Dies wäre mittels vollständiger Induktion leicht zu beweisen. Hier wird als erster Schritt dazu lediglich die Addition $0 + 0$ gezeigt.

Schritt	Bandinschrift	entspr. Befehl der Tt
1	* ! * ! ⊛	q_0 * * l Q_1
2	* ! * ⊡ *	q_1 ! * l Q_2
3	* ! ⊛ * *	q_2 * ! r Q_3
4	* ! ! ⊛ *	q_3 * * l Q_0
5	* ! ⊡ * *	q_0 ! * s Q_0
6	* ! ⊛ * *	

2.3.2 Umwandlung von Strichzahlen in Dezimalzahlen

Solange es die Umstände nicht erfordern, werden wir zwischen einer Zahl und ihrer Darstellung nicht unterscheiden. Aus „Strichdarstellung“ und „Dezimaldarstellung“ von natürlichen Zahlen wird dadurch abkürzend „Strichzahlen“ und „Dezimalzahlen“. Die Leistung der Additionsmaschine ist zwar durchaus befriedigend, doch ist ! statt 0 und !!!!!!!!!!!!!! statt 14

zumindest ungewohnt. Es wäre gut, wenigstens das Ergebnis der Addition als Dezimalzahl zu haben. Dazu muß zuallererst das Alphabet um die Zeichen 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 erweitert werden. Sodann müßte eine Maschine konstruiert werden, welche die Strichzahlen in Dezimalzahlen umwandelt. Die Idee dazu ist einfach, für jeden mit * überdruckten Strich müßte die Dezimalzahl um 1 erhöht werden. Probleme könnten sich bei der Null *!* und den Überträgen bei den Dezimalzahlen (9 nach 10 und 99 nach 100 usw.) ergeben. Auch ob die Tm in jedem Fall im vereinbarten Feld stehen bleibt, wäre zu überprüfen. Die Entstehung der nachfolgenden Tabelle soll nicht näher erläutert werden, dagegen wird die Zuverlässigkeit dieser Tt an einigen Beispielen gezeigt.

	$*$	!	0	1	2	3	4	5	6	7	8	9
q_0	l	$*lq_1$	s	s	s	s	s	s	s	s	s	s
q_1	$0rq_4$	$*lq_2$	rq_4	rq_4	rq_4	rq_4	rq_4	rq_4	rq_4	rq_4	rq_4	rq_4
q_2	$1rq_3$	lq_2	$1rq_3$	$2rq_3$	$3rq_3$	$4rq_3$	$5rq_3$	$6rq_3$	$7rq_3$	$8rq_3$	$9rq_3$	$0lq_2$
q_3	lq_1	rq_3	r	s	s	s	s	s	s	s	s	s
q_4	sq_0	s	s	s	s	s	s	s	s	s	s	s

Zunächst die Umwandlung von $**!*$ in $*0**$.

Schritt	Bandinschrift	entspr. Befehl der Tt
1	* * ! ⊛	q_0 * * l Q_0
2	* * ⊛ *	q_0 ! * l Q_1
3	* ⊛ * *	q_1 * 0 r Q_4
4	* 0 ⊛ *	q_4 * * s Q_0
5	* 0 * *	

Bereits die Umwandlung von $**!!!!**$ in $*3*****$ ist etwas langwieriger.

Schritt	Bandinschrift	entspr. Befehl der Tt
1	* * ! ! ! ! ⊛ *	q_0 * * l q_0
2	* * ! ! ! ⊡ * *	q_0 ! * l q_1
3	* * ! ! ⊡ * * *	q_1 ! * l q_2
4	* * ! ⊡ * * * *	q_2 ! ! l q_2
5	* * ⊡ ! * * * *	q_2 ! ! l q_2
6	* ⊛ ! ! * * * *	q_2 * 1 r q_3
7	* 1 ⊡ ! * * * *	q_3 ! ! r q_3
8	* 1 ! ⊡ * * * *	q_3 ! ! r q_3
9	* 1 ! ! ⊛ * * *	q_3 * * l q_1
10	* 1 ! ⊡ * * * *	q_1 ! * l q_2
11	* 1 ⊡ ! * * * *	q_2 ! ! l q_2
12	* ⊡ ! * * * * *	q_2 1 2 r q_3
13	* 2 ⊡ * * * * *	q_3 ! ! r q_3
14	* 2 ! ⊛ * * * *	q_3 * * l q_1
15	* 2 ⊡ * * * * *	q_1 ! * l q_2
16	* ⊡ * * * * * *	q_2 2 3 r q_3
17	* 3 ⊛ * * * * *	q_3 * * l q_1
18	* ⊡ * * * * * *	q_1 3 3 r q_4
19	* 3 ⊛ * * * * *	q_4 * * s q_0
20	* 3 ⊛ * * * * *	

Nun wäre noch der Übergang von 999 auf 1001 zu überprüfen. Die Bandinschrift **999!!* sei durch den v-ten Befehl entstanden. Der (v+1)-te Schritt des folgenden Beispiels entspricht

dabei dem Schritt 9 des vorigen Beispiels.Auch diese Tm läuft auf einem ganz oder teilweise leeren Band so weit nach links, bis sie etwas zum Bearbeiten findet oder das Bandende erreicht. Die einzig echte Stoppanweisung in der Tabelle ist $q_4 * *sq_0$. Alle anderen mit s beschrifteten Felder könnten auch frei bleiben, da bei fehlerfreiem Start und fehlerfreier Bearbeitung diese Fälle nicht auftreten können. Da es sich hier um eine mathematische Konstruktion handelt, könnte man einfach eine absolute Fehlerfreiheit voraussetzen. Es ist aber auch reizvoll - und für die Praxis unentbehrlich - mögliche Fehlerquellen zu erkennen und durch entsprechende Anweisungen zu entschärfen.

Schritt	Bandinschrift	entspr. Befehl der Tt
v+1	* * 9 9 9 ! ! [*] *	q_3 * * l q_1
v+2	* * 9 9 9 ! [!] * *	q_1 ! * l q_2
v+3	* * 9 9 9 [!] * * *	q_2 ! ! l q_2
v+4	* * 9 9 [9] ! * * *	q_2 9 0 l q_2
v+5	* * 9 [9] 0 ! * * *	q_2 9 0 l q_2
v+6	* * [9] 0 0 ! * * *	q_2 9 0 r q_2
v+7	* [*] 0 0 0 ! * * *	q_2 * 1 r q_3
v+8	* 1 [0] 0 0 ! * * *	q_3 0 0 r q_3
v+9	* 1 0 [0] 0 ! * * *	q_3 0 0 r q_3
v+10	* 1 0 0 [0] ! * * *	q_3 0 0 r q_3
v+11	* 1 0 0 0 [!] * * *	q_3 ! ! r q_3
v+12	* 1 0 0 0 ! [*] * *	q_3 * * l q_1
v+13	* 1 0 0 0 [!] * * *	q_1 ! * l q_2
v+14	* 1 0 0 [0] * * * *	q_2 0 1 r q_3
v+15	* 1 0 0 1 [*] * * *	q_3 * * l q_1
v+16	* 1 0 0 [1] * * * *	q_1 1 1 r q_4
v+17	* 1 0 0 1 [*] * * *	q_4 * * s q_0
v+18	* 1 0 0 1 [*] * * *	

Die Schwerfälligkeit von TURINGmaschinen erkennt man daran, daß das v von obigem Beispiel größer als eine Million ist. Eine vollständige Auflistung des Programmablaufs würde in der obigen Form etwa 30 000 Seiten beanspruchen. Dabei könnte man aber wegen der geringen Zeilenlänge nur die Bandinschrift in der Umgebung des Arbeitsfeldes wiedergeben.

Man könnte nun auch eine Tm konstruieren, welche Dezimalzahlen in Strichzahlen umwandelt. Durch eine Hintereinanderschaltung von mehreren Tm wird dann auch die Addition von Dezimalzahlen möglich: Aus Dezimalzahlen werden Strichzahlen, man addiert diese und wandelt sie wieder in Dezimalzahlen um. Solche Zusammenschaltungen werden in 2.5 und 2.6 behandelt.

2.4 EUKLIDischer Algorithmus und TURINGmaschine

Bevor man die ggT-Bestimmung einer Tm anvertraut, sollte man sich eigentlich vergewissern, ob diese die Division mit Rest beherrscht. Dies erweist sich aber als überflüssig, da es hier weniger auf die Division, als vielmehr auf die Bestimmung des Divisionsrestes ankommt. Der Divisionsrest zweier natürlicher Zahlen a und b ($a > b$), läßt sich auf der Tm leicht berechnen, indem man b so oft es geht von a subtrahiert. Was von a übrig bleibt, ist der Divisionsrest. Es wäre auch nur eine Frage des Aufwands, eine Tm anzugeben, die eine vollständige Division mit Rest durchführen kann. Mittels obiger Idee läßt sich die folgende Tabelle einer Tt erstellen.

% und & sind dabei Hilfsbuchstaben des Arbeitsalphabetes, sie kommen nicht im Eingabe- oder Ausgabealphabet vor.

	$*$	!	%	&1
q_0	l	$*lq_1$	s	s
q_1	$!lq_2$	l	s	s
q_2	rq_5	$\%rq_3$	l	l
q_3	lq_4	$\&lq_2$	r	r
q_4	rq_2	rq_2	$!l$	$*l$
q_5	lq_6	lq_2	$*r$	$!r$
q_6	sq_0	rq_6	s	s

Abb. 10

Genauer betrachtet, macht man sich bei diesem Verfahren folgenden Umstand zunutze. Ist $ggT(a,b) = z$, dann gilt $\left.\begin{array}{l} z \mid a \Longrightarrow z \cdot x = a \\ z \mid b \Longrightarrow z \cdot y = b \end{array}\right\} a - b = z \cdot x - zy = z \cdot (x - y)$

D. h., ist z Teiler von a und b, so auch von der Differenz $a - b$, für $a > b$.
Sei $a' = a - b$, so kann das Verfahren mit a' und b fortgesetzt werden.
Konkret läuft die Berechnung dann so ab. Die Zahlen a und b werden durch a bzw. b Striche auf dem Band notiert; nicht mit $a+1$, da die Null hier nicht berücksichtigt wird. Die Strichfolgen trennt ein Feld mit dem leeren Buchstaben. Das erste Arbeitsfeld ist das erste Feld hinter der letzten Strichfolge.
Nach dem Start wird nach links gehend der erste im Arbeitsfeld erscheinende Strich durch den leeren Buchstaben ersetzt. Dann sucht die Tm das freie Feld zwischen den Strichfolgen und druckt dort einen Strich hinein.

(***) Von dem ehemaligen Zwischenraum ausgehend, wird nun abwechselnd links ein % und rechts ein & gedruckt, bis das Ende der kürzeren Strichfolge erreicht ist. Alle Buchstaben der mit % oder & überdruckten kürzeren Strichfolge werden anschließend durch $*$ ersetzt. Die verbleibenden, von ! verschiedenen Buchstaben werden durch Striche ersetzt.
Wenn das letzte % oder & überdruckt ist, die Tm das Arbeitsfeld aber noch nicht verlegt hat, so stellen die Striche vor dem Arbeitsfeld - in Laufrichtung gesehen - die Zahl $a' = a - b$ dar. In der anderen Richtung ist b auf dem Band notiert. Ob der Strich im Arbeitsfeld zu a' oder zu b gehört, hängt von der jeweiligen Laufrichtung ab. Nun beginnt wieder der Vorgang, wie er ab (***) beschrieben ist.
Stößt die Tm nach der Rückverwandlung der % bzw. & auf den leeren Buchstaben statt auf einen Strich, so ist das Verfahren beendet. Nach dem Stoppbefehl steht das gesuchte Wort vor dem Arbeitsfeld. Dieses Verfahren besitzt nicht den Vorzug großer Anschaulichkeit und Durchsichtigkeit. Deshalb ist auf den folgenden Seiten der gesamte Programmablauf wiedergegeben.

Schritt	Bandinschrift	Befehl der Tt
1	* ! ! ! ! ! ! * ! ! ! ! ⊛	q_0 * * l q_0
2	* ! ! ! ! ! ! * ! ! ! [!] *	q_0 ! * l q_1
3	* ! ! ! ! ! ! * ! ! [!] * *	q_1 ! ! l q_1
4	* ! ! ! ! ! ! * ! [!] ! * *	q_1 ! ! l q_1
5	* ! ! ! ! ! ! * [!] ! ! * *	q_1 ! ! l q_1
6	* ! ! ! ! ! ! ⊛ ! ! ! * *	q_1 * ! l q_2
7	* ! ! ! ! ! [!] ! ! ! ! * *	q_2 ! % r q_3
8	* ! ! ! ! ! % [!] ! ! ! * *	q_3 ! & l q_2
9	* ! ! ! ! ! [%] & ! ! ! *	q_2 % % l q_2

10	* ! ! ! ! ! % & ! ! ! *	q_2 ! % r q_3
11	* ! ! ! ! % % & ! ! ! *	q_3 % % r q_3
12	* ! ! ! ! % % & ! ! ! *	q_3 % % r q_3
13	* ! ! ! ! % % & ! ! ! *	q_3 ! & l q_2
14	* ! ! ! ! % % & & ! ! *	q_2 & & l q_2
15	* ! ! ! ! % % & & ! ! *	q_2 % % l q_2
16	* ! ! ! ! % % & & ! ! *	q_2 % % l q_2
17	* ! ! ! ! % % & & ! ! *	q_2 ! % r q_1
18	* ! ! ! % % % & & ! ! *	q_3 % % r q_3
19	* ! ! ! % % % & & ! ! *	q_3 % % r q_3
20	* ! ! ! % % % & & ! ! *	q_3 % % r q_3
21	* ! ! ! % % % & & ! ! *	q_3 % % r q_3
22	* ! ! ! % % % & & ! ! *	q_3 ! & l q_2
23	* ! ! ! % % % & & & ! *	q_2 & & l q_2
24	* ! ! ! % % % & & & ! *	q_2 & & l q_2
25	* ! ! ! % % % & & & ! *	q_2 % % l q_2
26	* ! ! ! % % % & & & ! *	q_2 % % l q_2
27	* ! ! ! % % % & & & ! *	q_2 % % l q_2
28	* ! ! ! % % % & & & ! *	q_2 ! % r q_3
29	* ! ! % % % & & & ! *	q_3 % % r q_3
30	* ! ! % % % & & & ! *	q_3 % % r q_3
31	* ! ! % % % % & & & ! *	q_3 % % r q_3
32	* ! ! % % % % & & & ! *	q_3 & & r q_3
33	* ! ! % % % % & & & ! *	q_3 & & r q_3
34	* ! ! % % % % & & & ! *	q_3 & & r q_3
35	* ! ! % % % % & & & ! *	q_3 ! & l q_2
36	* ! ! % % % % & & & & *	q_2 & & l q_2
37	* ! ! % % % % & & & & *	q_2 & & l q_2
38	* ! ! % % % % & & & & *	q_2 & & l q_2
39	* ! ! % % % % & & & & *	q_2 % % l q_2
40	* ! ! % % % % & & & & *	q_2 % % l q_2
41	* ! ! % % % % & & & & *	q_2 % % l q_2
42	* ! ! % % % % & & & & *	q_2 % % l q_2
43	* ! ! % % % % & & & & *	q_2 ! % r q_3
44	* ! % % % % % & & & & *	q_3 % % r q_3
45	* ! % % % % % & & & & *	q_3 % % r q_3
46	* ! % % % % % & & & & *	q_3 % % r q_3
47	* ! % % % % % & & & & *	q_3 % % r q_3
48	* ! % % % % % & & & & *	q_3 & & r q_3
49	* ! % % % % % & & & & *	q_3 & & r q_3
50	* ! % % % % % & & & & *	q_2 & & r q_3
51	* ! % % % % % & & & & *	q_3 & & r q_3
52	* ! % % % % % & & & & *	q_3 * * l q_4
53	* ! % % % % % & & & & * *	q_4 & * l q_4
54	* ! % % % % % & & & * * *	q_4 & * l q_4
55	* ! % % % % % & & * * * *	q_4 & * l q_4
56	* ! % % % % % & * * * * *	q_4 & * l q_4

57	* ! % % % % % * * * * * *	q_4	% ! l q_4
58	* ! % % % % ! * * * * * *	q_4	% ! l q_4
59	* ! % % % ! ! * * * * * *	q_4	% ! l q_4
60	* ! % % ! ! ! * * * * * *	q_4	% ! l q_4
61	* ! % ! ! ! ! * * * * * *	q_4	% ! l q_4
62	* ! ! ! ! ! ! * * * * * *	q_4	! ! r q_2
63	* ! ! ! ! ! ! * * * * * *	q_2	! % r q_3
64	* ! % ! ! ! ! * * * * * *	q_3	! % l q_2
65	* ! % & ! ! ! * * * * * *	q_2	% % l q_2
66	* ! % & ! ! ! * * * * * *	q_2	! % r q_3
67	* % % & ! ! ! * * * * * *	q_3	% % r q_3
68	* % % & ! ! ! * * * * * *	q_3	% % r q_3
69	* % % & ! ! ! * * * * * *	q_3	! & l q_2
70	* % % & & ! ! * * * * * *	q_2	& & l q_2
71	* % % & & ! ! * * * * * *	q_2	% % l q_2
72	* % % & & ! ! * * * * * *	q_2	% % l q_2
73	* % % & & ! ! * * * * * *	q_2	* * r q_5
74	* % % & & ! ! * * * * * *	q_5	% * r q_5
75	* * % & & ! ! * * * * * *	q_5	% * r q_5
76	* * * & & ! ! * * * * * *	q_5	& ! r q_5
77	* * * ! & ! ! * * * * * *	q_5	& ! r q_5
78	* * * ! ! ! ! * * * * * *	q_5	! ! l q_2
79	* * * ! ! ! ! * * * * * *	q_2	! % r q_3
80	* * * ! % ! ! * * * * * *	q_3	! & l q_2
81	* * * ! % & ! * * * * * *	q_2	% % l q_2
82	* * * ! % & ! * * * * * *	q_2	! % r q_3
83	* * * % % & ! * * * * * *	q_3	% % r q_3
84	* * * % % & ! * * * * * *	q_3	& & r q_3
85	* * * % % & ! * * * * * *	q_3	! & l q_2
86	* * * % % & & * * * * * *	q_2	& & l q_2
87	* * * % % & & * * * * * *	q_2	% % l q_2
88	* * * % % & & * * * * * *	q_2	% % l q_2
89	* * * % % & & * * * * * *	q_2	* * r q_5
90	* * * % % & & * * * * * *	q_5	% * r q_5
91	* * * * % & & * * * * * *	q_5	% * r q_5
92	* * * * * & & * * * * * *	q_5	& ! r q_5
93	* * * * * ! & * * * * * *	q_5	& ! r q_5
94	* * * * * ! ! * * * * * *	q_5	* * l q_6
95	* * * * * ! ! * * * * * *	q_6	! ! r q_6
96	* * * * * ! ! * * * * * *	q_6	* * s q_0

Wie man sieht, ist das Verfahren langwierig, aber erfolgreich. Eine Vielzahl der Schritte dient lediglich der Verlegung des Arbeitsfeldes. Andere Arbeitsschritte wiederholen sich in stumpfer Gleichförmigkeit. Dies sollte nicht vergessen lassen, daß gerade diese einfach konstruierte Maschine alle Anforderungen erfüllt, die wir bisher an einen Algorithmus gestellt haben. Dieser Umständlichkeit ist in gewissem Umfang beizukommen.

2.5 Elementarmaschinen und TURINGdiagramme

Wie bereits festgestellt wurde, ist eine Tm durch ihre Tt bestimmt. Diese Tt läßt sich wiederum vorteilhaft in Tabellenform schreiben. In der Regel läßt sich aber weder aus der fünfspaltigen Matrix noch aus der Tabelle in kurzer Zeit das Verhalten der Tm überblicken. Auch eine Liste des Programmablaufs (z.B. Seite 28 ff) macht nicht immer deutlich, was die Tm eigentlich leistet. Auf die bisweilen nicht unerhebliche Länge solcher Listen wurde bereits hingewiesen.
Eine Idee aus dem letzten Absatz von 2.3.2 kann hier Abhilfe schaffen. Dort wurde gezeigt, wie sich ein Problem auch durch zweckmäßiges Zusammensetzen von mehreren Tm bewältigen läßt. Allerdings kann man eine übersichtliche Tt nicht aus mehreren unübersichtlichen gewinnen. Vielmehr werden zuerst die elementaren Handlungsweisen von Tm durch sogenannte Elementarmaschinen, kurz Tem genannt, beschrieben. Dann werden die Tem M_i je nach Bedarf zusammengesetzt. Dies aber nicht wieder als Tafel oder Tabelle, sondern in der Form eines Diagrammes, das man als TURINGdiagramm bezeichnet. TURINGdiagramme erinnern in ihrer Struktur an die Flußdiagramme für die Programmierung von elektronischen Datenverarbeitungsanlagen. In beiden Fällen werden Befehle bzw. Anweisungen durch Pfeile verbunden; man verwendet Verzweigungen und Schleifen. Am leichtesten lassen sich Tem beschreiben, die lediglich einen Buchstaben ins Arbeitsfeld drucken. Tem * druckt den leeren Buchstaben, Tem ! einen Strich. Tem r, Tem l und Tem s sind keine Druckmaschinen, vielmehr bewegen sie das Arbeitsfeld nach rechts oder nach links bzw. stoppen den Arbeitsgang. Aus diesen einfachsten Maschinen lassen sich weitere nützliche Maschinen zusammensetzen. Um deren Wirkungsweise übersichtlich zu beschreiben, bedarf es einiger Vereinbarungen:

$\sim$ ist der nicht bearbeitete Teil des Bandes,
w ist ein Wort über dem Arbeitsalphabet A,
r^n gibt die n-malige Ausführung von r an und
a_k ist der k-te Buchstabe des Arbeitsalphabetes A.

Symbol	Wirkungsweise	Diagramm
R	$\sim \boxed{*} w * \sim \Longrightarrow$ $\sim * w \boxed{*} \sim$	$\neq *$ $\longrightarrow r$
L	$\sim * w \boxed{*} \sim \Longrightarrow$ $\sim \boxed{*} w * \sim$	$\neq *$ $\longrightarrow l$
K	$\sim * w \boxed{*} \sim \Longrightarrow$ $\sim * w * w \boxed{*} \sim$	$L\, r \begin{cases} \xrightarrow{*} R \\ \xrightarrow{a_1} * R^2 a_1 L^2 a_1 \\ \vdots \\ \xrightarrow{a_n} * R^2 a_n L^2 a_n \end{cases}$
K_n	$\sim * w_1 * \cdots * w_n \boxed{*} \sim \Longrightarrow$ $\sim * w_1 * \cdots * w_n * w_1 \boxed{*} \sim$	$L_n\, r \begin{cases} \xrightarrow{*} R_n \\ \xrightarrow{a_1} * R^{n+1} a_1 L^{n+1} a_1 \\ \vdots \\ \xrightarrow{a_n} * R^{n+1} a_n L^{n+1} a_n \end{cases}$

Pfeile weisen im Diagramm den Weg. Bei Verzweigungen folgt man jenem Pfeil, welcher mit dem Buchstaben des Arbeitsfeldes beschriftet ist. Gilt ein Pfeil in allen Fällen, so werden die Buchstaben weggelassen. Gilt ein Pfeil für alle Arbeitsfeldinschriften außer für *, so wird dies durch $\stackrel{\neq *}{\longrightarrow}$ vermerkt ; entsprechendes gilt für $\neq a_k$. Aus einer ganzen Reihe von möglichen Tem sollen hier lediglich vier vorgestellt werden:

1. Die große Rechtsmaschine R geht nach rechts, bis sie einen $*$ findet;

2. die große Linksmaschine L geht ebenso nach links;

3. die Kopiermaschine K kopiert das Wort links vom Arbeitsfeld;

4. die Kopiermaschine K_n kopiert das N-te Wort links vom Arbeitsfeld nach rechts.

2.6 Beispiele für TURINGdiagramm

Bei den folgenden Beispielen werden die Tm danach getrennt, ob sie nun TURING-berechnen oder TURING-entscheiden. Mit einem gewissen Aufwand läßt sich die genaue Definition der beiden Begriffe angeben. Darauf wird hier verzichtet, da sich bereits aus der Bezeichnung und den nachfolgenden Beispielen das hier notwendige Verständnis ergibt.

2.6.1 TURING-berechenbare Funktionen

Die Nachfolgerfunktion $N(x)$

wird durch ! **r** berechnet. Die natürliche Zahl x wird dabei wie auch im nächsten Beispiel durch $(x+1)$ Striche dargestellt.

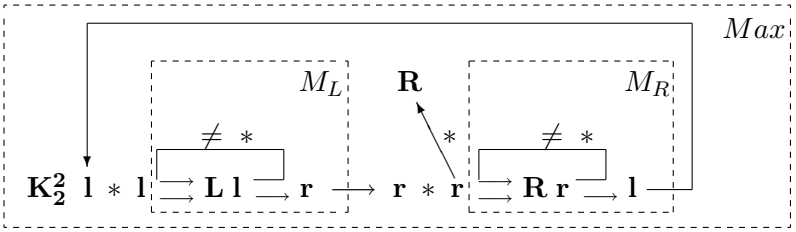
Die Summenfunktion $S(x,y) = x + y$

wurde bereits in 2.3.1 vorgestellt und läßt sich folgendermaßen beschreiben:

$$S = l \ * \ L \,!\, R \, l \, *$$

Die Funktion $Max(x,y)$

wird von der Tm Max berechnet.



K_2^2 bedeutet $K_2\ K_2$ Abb. 11a Tm Max

Schritt	Bandinschrift	bearbeitende Tm
1	∼ * ! ! * ! ! ! ⊛ * * * * * * *	K_2^2 l
2	∼ * ! ! * ! ! ! * ! ! * ! ! ⊡ *	* l
3	∼ * ! ! * ! ! ! * ! ! * ! ⊡ * *	M_L r
4	∼ * ⊡ ! * ! ! ! * ! ! * ! ! * *	* r
5	∼ * * ⊡ * ! ! ! * ! ! * ! ! * *	M_R l
6	∼ * * ! * ! ! ! * ! ! * ! ⊡ * *	* l
7	∼ * * ! * ! ! ! * ! ! * ⊡ * * *	M_L r
8	∼ * * ⊡ * ! ! ! * ! ! * ! * * *	* r
9	∼ * * * ⊛ ! ! ! * ! ! * ! * * *	R
10	∼ * * * * ! ! ! ⊛ ! ! * ! * * *	Ergbnis: !!!

Beim Diagramm der Tm Max sind einige Tem zu der Tem M_L und der Tem M_R zusammengefaßt. Sind mehrere Wörter durch nur einen leeren Buchstaben getrennt, so sucht Tem M_R das rechte Ende dieser Wortreihe. Wie man sieht, kann innerhalb einer Elementarmaschine ein recht komplizierter Vorgang ablaufen. Lediglich das Gesamtverhalten der Maschine sollte elementar verständlich sein (schreiben, lesen, suchen, kopieren).

Die Produktfunktion $P(x, y) = x \cdot y$

berechnet die Tm P nach dem folgenden Diagramm. Das Programm wird durch die Auflistung der einzelnen Schritte für den Fall $x = 3$ und $y = 5$ leicht durchschaubar.

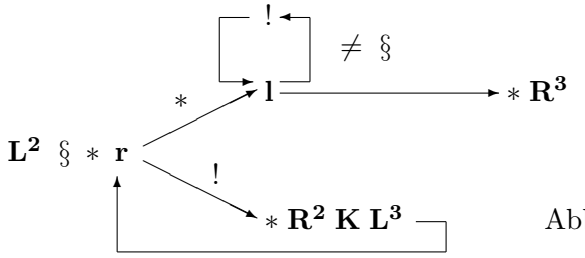


Abb. 11b Tm P

n	Bandinschrift	Tem
1	∼ * ! ! ! * ! ! ! ! ! ⊛ * * * * * * * * * * * * * * * *	L^2
2	∼ ⊛ ! ! ! * ! ! ! ! ! * * * * * * * * * * * * * * * * *	§ r
3	∼ § ⊡ ! ! * ! ! ! ! ! * * * * * * * * * * * * * * * * *	* R^2
4	∼ § * ! ! * ! ! ! ! ! ⊛ * * * * * * * * * * * * * * * *	K
5	∼ § ⊛ ! ! * ! ! ! ! ! * ! ! ! ! ! * * * * * * * * * * *	L^3
6	∼ § ⊛ ! ! * ! ! ! ! ! * ! ! ! ! ! * * * * * * * * * * *	r
7	∼ § * ⊡ ! * ! ! ! ! ! * ! ! ! ! ! * * * * * * * * * * *	* R^2
8	∼ § * * ! * ! ! ! ! ! ⊛ ! ! ! ! ! * * * * * * * * * * *	K
9	∼ § * * ! * ! ! ! ! ! * ! ! ! ! ! ! ! ! ! ⊛ * * * * *	L^3
10	∼ § * ⊛ ! * ! ! ! ! ! * ! ! ! ! ! ! ! ! ! ! * * * * * *	r
11	∼ § * * ⊡ * ! ! ! ! ! * ! ! ! ! ! ! ! ! ! ! * * * * * *	R^2
12	∼ § * * * * ! ! ! ! ! ⊛ ! ! ! ! ! ! ! ! ! ! * * * * * *	K
13	∼ § * * * * ! ! ! ! ! * ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ⊛	L^3
14	∼ § * * ⊛ ! * ! ! ! ! ! * ! ! ! ! ! ! ! ! ! ! ! ! ! ! !	r

```
15    ∼ § * * * ⊛ ! ! ! ! ! * ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! * ! ! ! ! ! *    l→≠§!
16    ∼ § * ! ! * ! ! ! ! ! * ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! * ! ! ! ! ! *    * R3
17    ∼ * ! ! ! * ! ! ! ! ! * ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! * ! ! ! ! ! ⊛
```

Tem § druckt den Hilfsbuchstaben § des Arbeitsalphabets.
Die Grundidee des Programms ist, daß y so oft kopiert wird, wie es x angibt. Die Tm P berechnet die Produktfunktion auch, wenn x oder y gleich sind. Hier steht aber entgegen früherer Bezeichnungen $*$ für die Null.

Die „russische Bauernmethode“ als TURINGmaschine Tm RBM

Diese Methode berechnet die Produktfunktion durchaus zuverläßlich, wie in 1.1.5 bereits gezeigt. Dies leistet auch die Tm RBM, deren Diagramm in Abb. 12 dargestellt ist. Das TURINGdiagramm für die Tm RBM ist wesentlich komplizierter als das Diagramm für Tm P. Zwecks eines besseren Überblicks ist Tm RBM in fünf einzelne Maschinen aufgegliedert.

M_1 schafft die notwendigen Anfangsbedingungen.

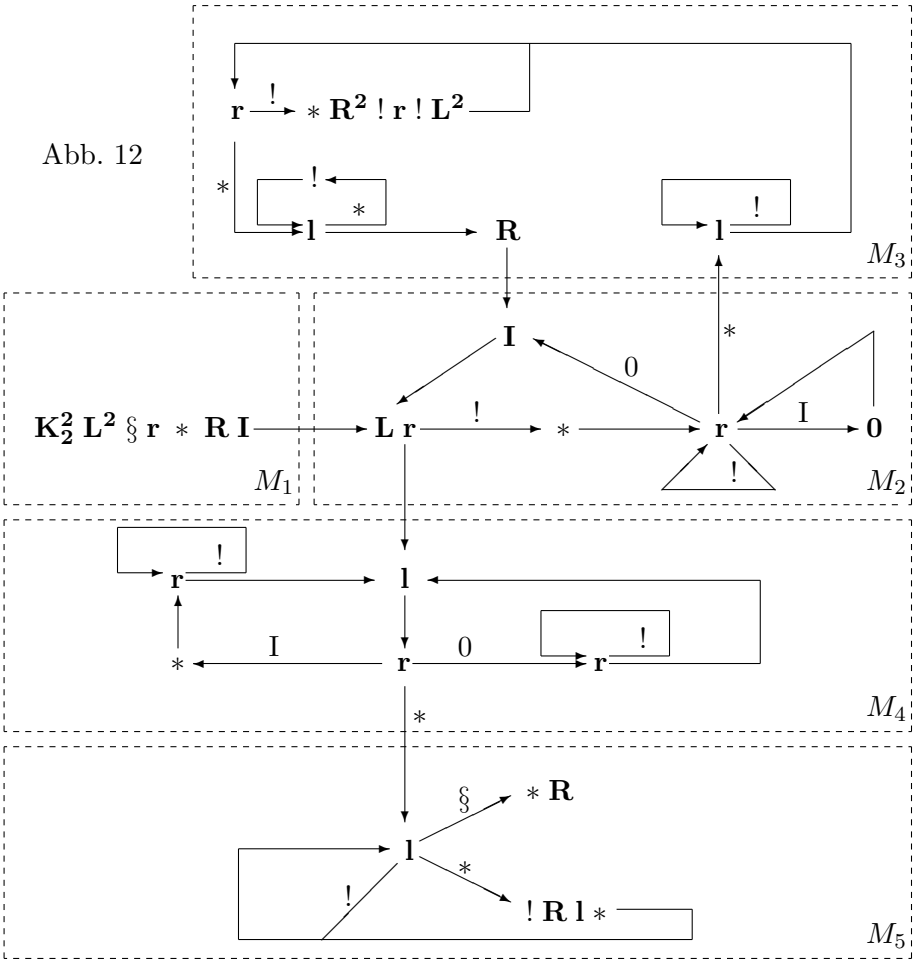
M_2 schreibt x in eine Dualzahl um. Sie wird mit den Zeichen I und 0 geschrieben. Durch M_2 wird das Zeichen I oder 0 der Stelle 2^0 von x dabei vor die Darstellung von $2^0 \cdot y$ gesetzt. Weiter nach rechts steht jeweils vor der Darstellung von $2^i \cdot y$ das Zeichen I oder 0 der Stelle 2^i x.

M_3 verdoppelt y so oft es nötig ist.

M_4 ersetzt nach rechts gehend I und 0 durch *. Alle $I \cdot 2^i \cdot y$ bleiben dabei stehen, während die $0 \cdot 2^i \cdot y$ weggewischt werden.

M_5 sammelt alle übriggebliebenen Striche auf und setzt sie mit einem Leerfeld Abstand hinter die ursprüngliche Aufgabenstellung.

Abb. 12



Die Auflistung des Programmablaufes und das Diagramm selbst machen die Vorgänge etwas deutlicher.

Schritt	Bandinschrift	Tem
1	∼∗!!!!!∗!!!**********	M_1
2	∼∗!!!!!∗!!!§∗!!!!I!!!**********	M_2
3	∼∗!!!!!∗!!!§∗∗!!!0!!!**********	M_3
4	∼∗!!!!!∗!!!§∗∗!!!0!!!∗!!!!!!**********	M_2
5	∼∗!!!!!∗!!!§∗∗∗∗!0!!!0!!!!!!**********	M_3
6	∼∗!!!!!∗!!!§∗∗∗∗!0!!!0!!!!!!∗!!!!!!!!!!∗	M_2
7	∼∗!!!!!∗!!!§∗∗∗∗∗I!!!0!!!!!!I!!!!!!!!!!∗	M_4
8	∼∗!!!!!∗!!!§∗∗∗∗∗∗!!!∗∗∗∗∗∗∗∗!!!!!!!!!!∗	M_5
9	∼∗!!!!!∗!!!∗!!!!!!!!!!!!!!!**********	

Damit ist auch der mathematische Hintergrund der „russischen Bauernmethode“ weitgehend geklärt.

Die TURING-Maschine Tm EUKLID

soll nun als TURINGdiagramm vorgestellt werden.

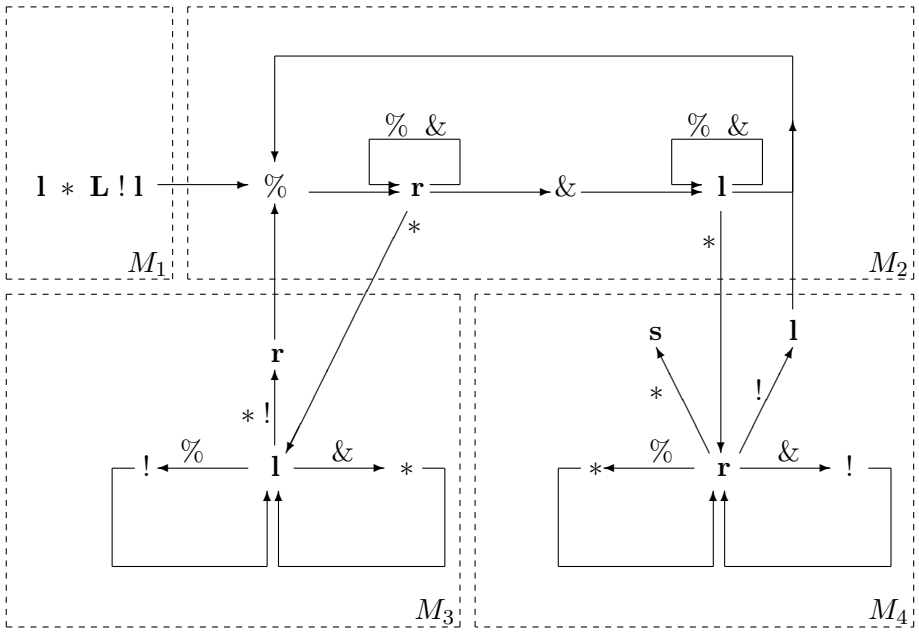


Abb. 13

Wie in 2.4 dargestellt wurde, berechnet die Tm EUKLID den ggT(a, b). Bei der Änderung der Darstellung von einer Tabelle in ein Diagramm unterscheidet man zweckmäßigerweise vier Tem.

M_1 bereitet die Berechnung vor.

M_2 setzt % und & nach Vorschrift.

M_3 und

M_4 ersetzen % und & durch * und M_4 beendet die Berechnung, wenn die notwendige Bedingung erfüllt ist.

Die Zuverlässigkeit der Tm EUKLID nach dem Diagramm von Abb. 13 soll am Fall aus 2.4 erprobt werden, die Schrittnummern beziehen sich auf dieses Beispiel.

Schritt	Bandinschrift	Tem
1	∼ * ! ! ! ! ! ! * ! ! ! ! ⊛	M_1
7	∼ * ! ! ! ! ! ! ! ! ! ! ! * *	M_2
52	∼ * ! % % % % % & & & & ⊛ *	M_3
63	∼ * ! ! ! ! ! ! * * * * * *	M_2
73	∼ ⊛ % % & & ! ! * * * * * *	M_4
79	∼ * * * ! ! ! ! * * * * * *	M_2
89	∼ * * ⊛ % % & & * * * * * *	M_4
96	∼ * * * * * ! ! ⊛ * * * * *	

Die Arbeit der Tm EUKLID ist dabei keineswegs geringer geworden, aber der Betrachter wird jetzt nur noch mit den wesentlichsten Zwischenschritten konfrontiert.

$*!*!\boxtimes$ M_1
$*\boxtimes!**$ M_2
$\boxtimes\%\&**$ M_4
$**!\boxtimes*$

Auch für den trivialen Fall ggT(1,1)ist das Verfahren zu gebrauchen, wie die rechts stehende Darstellung zeigt. Es könnte leicht sein, daß eine Tm gerade in einem solchen Fall versagt, weil Vorbereitung des Rechenganges und der eigentliche Rechengang eng beieinander liegen und sich so stören.

2.6.2 TURING-entscheidbare Relationen

Einstellige Relationen bezeichnet man umgangssprachlich als Eigenschaften, mehrstellige Relationen als Beziehungen. Teilbarkeit und Gleichheit sollen hier als Beispiel dienen.

Die Teilereigenschaft

ist TURING-entscheidbar. Die Tm nT leistet dies für beliebiges $k \in N$ und ein festes $n(n \geq 1)$. Gilt $n \mid k$, so bleibt Tm nT auf $*$ stehen, andernfalls auf !. Tm nT startet und stoppt auf dem ersten Feld nach dem Wort w_k, welches aus k Strichen besteht. Die ersten n Felder des Bandes nach dem Wort w_k müssen leer sein.

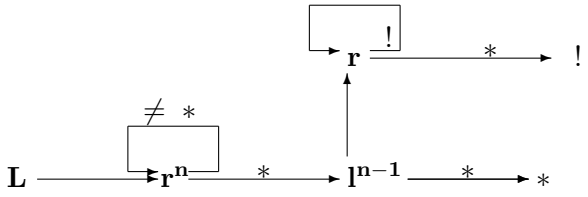


Abb. 14 Tm nT

Die Teilbarkeitsbeziehung

ist auch TURING-entscheidbar. Die Tm pT entscheidet ob $p \mid q$.

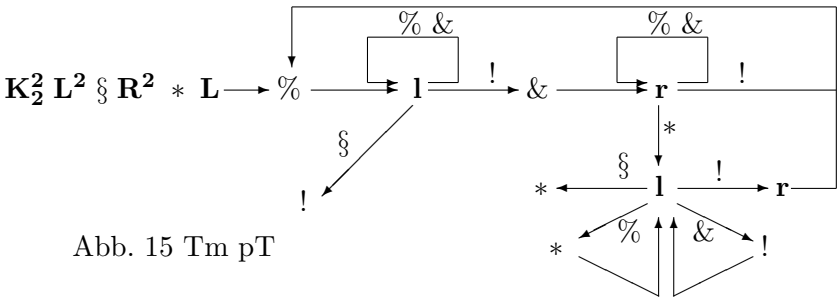


Abb. 15 Tm pT

Auf dem Band stehen vor dem Start (in dieser Reihenfolge) q Striche, ein leeres Feld, p Striche und das leere Arbeitsfeld. Hinter dem Arbeitsfeld ist das Band leer. Dasselbe Feld ist auch wieder das letzte Arbeitsfeld. Gilt $p \mid q$, so steht nach dem Maschinenstopp $*$ im Arbeitsfeld,

andernfalls !. Das in Abb. 15 dargestellte TURINGdiagramm von Tm pT erinnert stark an das Diagramm der Tm EUKLID.

Die Gleichheitsbeziehung

läßt sich durch einen kleinen Umbau der Tm pT leicht TURING-entscheiden.

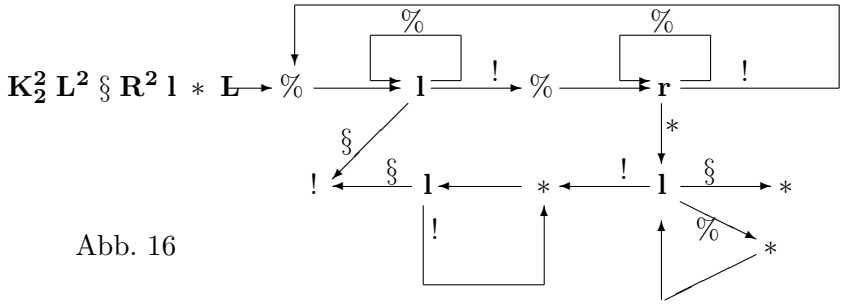


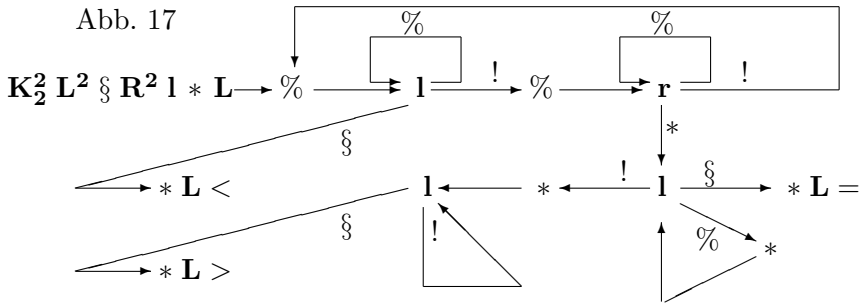
Abb. 16

Für die so entstandene Tm G gelten die gleichen Bedingungen wie für Tm pT. Ist $q = p$, so steht der leere Buchstabe * im letzten Arbeitsfeld.

Die Tm Trichotom

stellt eine weitere Abwandlung dar.

Abb. 17



Sie startet unter den gleichen Bedingungen wie Tm G. Ihr letztes Arbeitsfeld ist aber das Leerfeld zwischen w_q und w_p. Dort wird je nach Rechenverlauf $=$, $<$ oder $>$ gedruckt.
Hier werden also offensichtlich zwei Entscheidungen getroffen. Zuerst wird zwischen $<$ und $\geq$ unterschieden, dann zwischen $=$ und $>$.

Kapitel 3

Entscheidbarkeit und Unentscheidbarkeit

In 1.2 wurde der intuitive Algorithmenbegriff festgelegt. Dann wurde die Tm als eine präzise Fassung des Algorithmenbegriffs vorgestellt. Es bleibt zu prüfen, ob jede Tm als Algorithmus verstanden werden kann, und ob jeder Algorithmus durch eine Tm darstellbar ist. Das erste bietet keine Schwierigkeiten. Eigenschaften und Fähigkeiten von Tm sind so klar, daß sie ohne weiteres unter dem allgemeinen Algorithmenbegriff einzuordnen sind. Jede Tm stellt also einen Algorithmus dar. Die zweite Frage erweist sich als weitaus schwieriger.

3.1 Die CHURCHsche These

Man nehme einmal an, daß sich für ein bestimmtes Problem keine das Problem lösende Tm finden ließe. Dann gibt es offensichtlich drei Möglichkeiten:

1. Die „Konstrukteure“ waren nicht ausdauernd oder findig genug.
2. Die Grundkonzeption der Tm ist unzureichend.
3. Das Problem ist überhaupt nicht lösbar.

zu 1.: Hier heißt es weiterarbeiten und auf die Erfolge der kommenden Jahre und Generationen warten.
zu 2.: Haben TURING und POST ihre Maschinen etwa in unzureichender Art konzipiert? Sollte eine verbesserte Tm über mehrere Operationswerke, mehrere Rechenbänder mit mehreren Schreib-Lese-Köpfen und damit über viele Arbeitsfelder verfügen? Sollte das Rechenband als „eindimensionaler“ Speicher besser durch einen flächenhaften oder räumlichen Speicher ersetzt werden? Derartige Fragen müssen aus Erfahrung verneint werden. Mehr-Bänder-Rechenmaschinen sind zwar bisweilen wesentlich praktischer als Tm, doch stellen sie keine grundsätzliche Notwendigkeit für die Berechenbarkeit einer Funktion oder die Entscheidbarkeit einer Relation dar. Dieser Sachverhalt wurde von TURING sehr bald erkannt. Die von ihm ausgesprochene These, „daß die Begriffe der mit einer TURINGmaschine berechenbaren Funktion und der im gewöhnlichen Sinne berechenbaren Funktion äquivalent sind“ [3, 6], wurde als TURINGsche These bezeichnet. 1937 wies TURING „die Gleichwertigkeit seines Berechenbarkeitsbegriffes und der λ-Definierbarkeit von CHURCH“ nach [3, 6]. Die Äquivalenz der Begriffe λ-Definierbarkeit und allgemein-rekursive Funktion war bereits 1936 von CHURCH, KLEENE

und ROSSER gezeigt worden. Dies veranlaßte CHURCH zu seiner bereits zitierten These.
Da inzwischen die Äquivalenz aller präzisierten Algorithmenbegriffe erfolgt ist, spricht man heute zumeist nur noch von der CHURCHschen These, welche somit die TURINGsche These einschließt. Beweisen läßt sich keine der beiden Thesen. Sie haben etwa die gleiche Gültigkeit wie der Energieerhaltungssatz in der Physik. Es ist zwar sehr unwahrscheinlich, doch man sollte sich in beiden Fällen nicht wundern, wenn sich eines Tages doch noch Gegenbeispiele finden ließen.
zu 3.: Man wird die algorithmische Unlösbarkeit eines Problems in einer mathematisch einwandfreien Art und Weise nur beweisen können, wenn man über einen präzisen Algorithmenbegriff verfügt. Diese Gleichsetzung von intuitivem und präzisem Algorithmenbegriff beinhaltet aber gerade die CHURCHsche These. Solche Beweise lassen sich mit der Tm führen. Wie dies geschieht, zeigen die folgenden Abschnitte.

3.2 Ein Minimalalphabet

Bei den Beispielen für Tm wurden verschiedene Alphabete verwendet. Es fragt sich, ob dies eine durch das Problem bestimmte Notwendigkeit ist oder nur das Verfahren erleichtert. Dem Alphabet $A = \{a_1, a_2, a_3, a_4\}$ kann man beispielsweise das Alphabet $A' = \{!, !!, !!!, !!!\}$ zuordnen. Damit ließe sich eine Bandinschrift über A in eine Bandinschrift über A' verschlüsseln. Das leere Zeichen * bleibt dabei erhalten. Bei dieser Verschlüsselung wird jeder Buchstabe a_i von A zum Wort w_i über dem Minimalalphabet $A'' = \{*, !\}$.
Für eine fehlerfreie und eindeutige Entschlüsselung müssen die einzelnen Strichfolgen voneinander getrennt werden. Man setzt jeweils das leere Zeichen $*$ hinter ein Wort w_i.

$a_2a_3a_1 \longleftrightarrow !!*!!!*!*$
$a_3 * a_4a_2 * a_2 \longleftrightarrow !!! * *!!!!*!! * *!!*$

Auch eine Tt kann in dieser Weise verschlüsselt werden [12, 49f]. Zuerst trennt man die einzelnen Buchstaben, indem man das leere Zeichen $*$ hinten anhängt.

$q_4! * lq_3 \longleftrightarrow q_4*! * * * l * q_3*$

Weiter vereinbart man: Aus

q_i	werden	$(i+4)$ Striche
r	wird	!
l	wird	!!
s	wird	!!!
$q_4 * r * * * l * q_3 * s*$	wird	!!!!!!!*! * **!!*!!!!!!*!!!*

Bemerkung: Die Tm Z berechne die Funktion Z. Verschlüsselt man die Bandinschrift und die Tt Z, so wird die Tm Z' in aller Regel $Z(x)$ nicht mehr berechnen.
Ersetzt man ! durch 1 und * durch 0, so zeigt sich, daß man jeder Zeile einer Tt eine Dualzahl (aus N) zuordnen kann. Weitergehend kann man auch die ganze Tt in eine Zeile schreiben und findet auch dafür die umkehrbar eindeutig zugeordnete Dualzahl. Das heißt nicht, daß sich aus jeder natürlichen Zahl über ihre duale Darstellung eine Tt finden läßt.
Eine Tt ist immer ein Text von endlicher Länge über einem endlichen Alphabet. Folglich kann die oben aufgezeigte Zuordnung immer erfolgen. Damit kann es aber höchstens abzählbar unendlich viele Tt geben und somit auch nur abzählbar unendlich viele berechenbare Funktionen.

Legt man den Funktionsbegriff von LEONHARD EULER (*1707, †1783) zugrunde, so kann es auch nicht mehr Funktionen geben, denn „einen analytischen Ausdruck aus rechnerisch verknüpften Variablen und Konstanten“ [10, 153] schreibt man leicht als einen endlichen Text über einem endlichen Alphabet.
Dies ist bei einem Funktionsbegriff in der Deutung von GUSTAV LEJEUNE DIRICHLET (*1805, †1859) nicht mehr möglich. Er verstand eine Funktion als Zuordnung, fast schon im modernen Sinne als Abbildung [10, 193]. Eine Menge mit n Elementen kann auf n^n verschiedene Arten auf sich selbst abgebildet werden. Die Menge der natürlichen Zahlen mit der Mächtigkeit $\aleph_0$ kommt auf $\aleph_0^{\aleph_0} = \aleph$ Möglichkeiten. Nach dem DIRICHLETschen Funktionsbegriff gibt es folglich überabzählbar unendlich viele Funktionen, deren Argumente und Werte aus der Menge der natürlichen Zahlen sind.
Trotz eines solchen Existenzbeweises tut man sich schwer, eine nicht berechenbare Funktion anzugeben. Indem man die Zuordnungsvorschrift eindeutig beschreibt, gibt man gleichzeitig eine Berechnungsvorschrift, was der Nichtberechenbarkeit widerspricht.
Dagegen läßt sich die Nichtentscheidbarkeit einer Relation leichter zeigen und über diese dann auch Beispiele für die Nichtberechenbarkeit von Funktionen.

3.3 Das Halteproblem

Nach der Bemerkung in 2.1 ist die Berechnung einer Funktion beendet, wenn die Tm stoppt. Anders gewendet heißt dies, daß die Tm eine Funktion nicht berechnet, wenn sie nicht stoppt. Mit einer nicht stoppenden Tm wäre somit auch eine nicht berechenbare Funktion gegeben.
Dies setzt allerdings voraus, daß der Begriff der Tm in beschränktem Umfang erweitert wird. Die Tm wurde als Präzisierung des Algorithmenbegriffs geschaffen. Da eine nicht stoppende Tm nicht nach endlich vielen Schritten zu einem Ergebnis kommt, stellt sie keinen Algorithmus und damit schon gar keine Tm dar. Beweistechnisch ist es aber günstig, wenn man die Redewendung „Die Tm stoppt nicht“ zuläßt. Mit einer solchen Feststellung verliert die konstruierte Maschine aber die Eigenschaft eine Tm zu sein, d.h. es gibt keine derartige Tm und damit keinen derartigen Algorithmus. Letzteres gilt nur bei Anerkennung der CHURCHschen These.
Bei der Suche nach einer nicht stoppenden Tm - der Widersinn wurde eben erklärt - könnte man sich einer Tm bedienen. Sie müßte so konstruiert sein, daß sie nach der „Betrachtung“ einer beliebigen Tt und eines beliebigen Wortes entscheiden könnte, ob eine Tm mit dieser Tt bei Eingabe dieses Wortes stehen bleibt oder nicht. Dies ist das „Allgemeine Halteproblem für Tm“ [12, 49]:

> Gibt es einen Algorithmus, der bei Angabe einer Tm M und eines Wortes w entscheidet, ob M bei Eingabe von w hält oder nicht?

Die Ausführungen in 3.2 zeigen den Weg für eine systematische Suche. Dazu durchläuft man der Reihe nach die Menge der natürlichen Zahlen in ihrer dualen Darstellung. Jede Dualzahl wird darauf überprüft, ob sie eine Tt in der verschlüsselten Form darstellt oder nicht. Bei jeder so gefundenen Tt wird wiederum geprüft, ob die nach ihr arbeitende Tm stoppt oder nicht. Verwendet man als Eingabewort w die eindimensional geschriebene Tt dieser Tm, so steht man bei der Konstruktion des gesuchten Algorithmus vor dem speziellen Halteproblem [12, 51]:

> Gibt es einen Algorithmus, mit dessen Hilfe man entscheiden kann, ob eine Tm M mit dem Alphabet $A_M = I_M = 0_M = \{*, !\}$ bei Eingabe ihrer eigenen Tt stehen bleibt und das Wort ! ausgibt?

Diese Frage muß mit „Nein“ beantwortet werden, denn es gilt der Satz: Das spezielle Halteproblem ist unentscheidbar. Der Begriff „unentscheidbar“ besagt, daß zu diesem Problem kein Entscheidungsverfahren existiert.
Der Beweis erfolgt indirekt. Die Tm E entscheide das spezielle Halteproblem, dann auch im besonderen für die Tm M. Die eindimensionale Schreibweise der Tt M ist durch das Wort w_M^T gegeben. Stoppt die Tm M bei der Eingabe von w_M^T, und gibt das Wort ! aus, so soll Tm E nach Konstruktion das leere Wort * ausgeben. In allen anderen Fällen gibt Tm E das Wort ! aus.
Das Ein- und Ausgabealphabet von Tm E ist minimal, es gilt $I_E = O_E = \{*, !\}$. Es läßt sich zeigen, daß eine solche Tm mit einem minimalen Arbeitsalphabet auskommen kann. Tm E sei so konstruiert, daß $A_E = \{*, !\}$. Damit gehört Tm E zur Klasse jener Maschinen, auf welche das spezielle Halteproblem zutrifft, und sie muß dann das spezielle Halteproblem auch für sich selbst entscheiden können.
Gibt man der Tm E ihre Tt E als Wort w_E^T ein, so können zwei Fälle auftreten:
Fall 1: Tm E gibt nicht das Wort ! aus.
Fall 2: Tm E gibt das Wort ! aus.
zu Fall 1: Hier könnte man wiederum unterscheiden, ob die Tm E nicht stoppt oder ein Wort $\neq$! ausgegeben hat. Dies wird hinfällig, denn nach ihrer Konstruktion hat Tm E in beiden Fällen das Wort ! auszugeben. Dies ist aber offensichtlich ein Widerspruch.
zu Fall. 2: Auch dies führt zum Widerspruch, denn nun müßte Tm E nach Konstruktion das leere Wort * ausgeben.
Ergibt sich aus den einzigen beiden Möglichkeiten jeweils ein Widerspruch, so muß die Voraussetzung falsch sein. Folglich gibt es keine Tm E, welche das spezielle Halteproblem entscheidet und damit auch keinen entsprechenden Algorithmus.
Auch das allgemeine Halteproblem ist unentscheidbar. Um dies zu beweisen, müßte hier zuerst die universelle Tm U vorgestellt werden. Tm U ist in der Lage, nach Eingabe des Wortes w_M^T die Arbeit der Tm M zu übernehmen, man sagt, Tm U simuliert Tm M.
Der wesentliche Schluß zum Beweis der Unentscheidbarkeit des allgemeinen Halteproblems läßt sich aber auch ohne genaue Kenntnis der Tm U verstehen. Es zeigt sich nach einigen Beweisschritten, daß die Entscheidbarkeit des allgemeinen Halteproblems die Entscheidbarkeit des speziellen Halteproblems nach sich ziehen würde. Dessen Unentscheidbarkeit wurde aber schon bewiesen. Also ergibt sich wiederum ein Widerspruch zur Voraussetzung, daß das allgemeine Halteproblem entscheidbar ist.

Kapitel 4

Abschluß

4.1 Schlußbemerkungen

TURINGmaschine, TURINGtafel, TURING-Berechenbarkeit, TURING-Entscheidbarkeit und das Halteproblem sind als wichtige Begriffe der Algorithmentheorie dargelegt worden. Die TURING-Aufzählbarkeit, der Begriff Konfiguration, die GÖDELisierung und die universelle TURINGmaschine wurden dagegen kaum oder sogar gar nicht angesprochen. Schon diese wenigen Begriffe reichen aus, um sich einen tieferen Einblick in die Theorie der Algorithmen zu verschaffen. Dabei ist die Unentscheidbarkeit des allgemeinen Halteproblems von zentraler Bedeutung, denn eine Vielzahl von Beweisen zu weiterführenden Sätzen beinhaltet folgenden Schluß:

> Das Problem Z ist genau dann lösbar, wenn das Halteproblem entscheidbar ist.

Mit derartigen Schlüssen war es möglich, die Unlösbarkeit des Wortproblems für Semi-Thue-Systeme und Thue-Systeme sowie für Gruppen, die Unentscheidbarkeit der Prädikatenlogik, die Unvollständigkeit der Prädikatenlogik der zweiten Stufe, die Unentscheidbarkeit und die Unvollständigkeit der Arithmetik als auch die Unlösbarkeit des zehnten HILBERTschen Problems zu beweisen.
Aber nicht nur im Bereich der Logik, der mathematischen Grundlagenforschung, der Gruppen- und der Zahlentheorie findet man Interesse für die Ergebnisse der Algorithmentheorie. Die inzwischen selbständige Wissenschaft der theoretischen Informatik nutzt die Algorithmentheorie in besonderem Maße und fördert sie durch Erkenntnisse aus eigenen Arbeiten. Gemeinsame Anliegen ergeben sich beispielsweise aus folgenden Fragestellungen:

1. Wie läßt sich prüfen, ob ein Algorithmus immer das leistet, wofür er erstellt worden ist?
2. Läßt sich für ein bestimmtes Problem ein Algorithmus von minimalem Umfang angeben?
3. Was ist für ein bestimmtes Problem das Minimum an Rechenschritten? Wie wirkt sich dieses Minimum auf den Speicherplatzbedarf aus? Wie wirkt sich eine Minimalisierung des Speicherplatzes auf die Anzahl der notwendigen Rechenschritte aus?
4. Läßt sich für die Anzahl der Rechenschritte und den Speicherplatzbedarf ein gemeinsames Maß finden? (Dies wurde durch die sogenannte Kompliziertheit eines Algorithmus in der Komplexitätstheorie bereits geleistet.)
5. Gibt es Algorithmen, welche Algorithmen erzeugen können, und wie müssen solche Algorithmen beschaffen sein?

6. Kann man einen Algorithmus entwickeln, der aus einer Lösungsidee einen fertigen Algorithmus erstellt?

7. Gibt es für die Darstellung von Algorithmen besonders vorteilhafte Methoden (TURINGtafeln, Flußdiagramme, Programme in Programmiersprache)?

8. Wie weit läßt sich die mathematische Beweisführung algorithmieren?

Die großen elektronischen Datenverarbeitungsanlagen zeigen, welche ungeheuren Leistungen durch zweckmäßige Algorithmen vollbracht werden können.
Andererseits ist mit der Unentscheidbarkeit des Halteproblems gezeigt, daß derartigen Maschinen grundsätzlich und auf immer Grenzen gesetzt sind. Man kann diesen Grenzen näher kommen, indem man sich auf algorithmisch behandelbare Teilgebiete beschränkt. Weiterhin ist denkbar, daß man für bestimmte Gebiete Verfahren entwickelt, welche ein Problem zwar nicht immer lösen, dafür aber ein Maximum an Lösungswahrscheinlichkeit aufweisen.
Ließe sich zeigen, daß Mensch und TURINGmaschine äquivalent sind, so wären in der Algorithmentheorie auch die Grenzen menschlicher Erkenntnis vorgezeichnet. Da aber die Mathematik sich nicht auf die Entwicklung von Algorithmen beschränkt, und der Mensch sich nicht vollkommen determiniert verhält, müssen diese Grenzen an anderer Stelle gesucht werden.

Literaturverzeichnis

[1] Heinrich Behnke, Reinhold Remmert, Hans-Georg Steiner und Horst Tietz : Mathematik 1, Das Fischer Lexikon, Frankfurt am Main, 1964

[2] Heinrich Behnke und Horst Tietz (Herausg.) : Mathematik 2, Das Fischer Lexikon, Frankfurt am Main, 1966

[3] Wilfried Brauer und Klaus Indermark : Algorithmen, rekursive Funktionen und formale Sprachen, BI-Hochschulskripten 817, Mannheim, 1968

[4] Moritz Cantor : Vorlesungen über Geschichte der Mathematik, Band 1, Stuttgart, 1965 (1907)

[5] Volker Claus : Einführung in die Informatik, Mathematik für das Lehramt an Gymnasien, Stuttgart, 1975

[6] Walter R. Fuchs : Knaurs Buch der modernen Mathematik, Exakte Geheimnisse, München, 1966

[7] Hans Hermes : Aufzählbarkeit, Entscheidbarkeit, Berechenbarkeit, Heidelberger Taschenbücher Band 87, 2. Aufl., Berlin, Heidelberg, New York, 1971

[8] Konrad Jacobs (Herausg.) : Selecta Mathematica II, Heidelberger Taschenbücher Band 67, Berlin, Heidelberg, 1970

[9] H. Jung : Ein altes Multiplikationsverfahren, in : MNU, 24. Band, Heft 6, S. 368, Bonn/Frankfurt, 1971

[10] Gerhard Kropp: Geschichte der Mathematik, Heidelberg, 1969

[11] Herbert Lugowski und Hanns Joachim Weinert : Grundzüge der Algebra, Teil I, Leipzig, 1968

[12] Klaus Menzel, J. Schornstein und S. Stief : Elektronische Datenverarbeitung, DIFF, Grundkurs Mathematik, Studienbrief V,1, Tübingen, 1976

[13] Herbert Meschkowski : Mathematisches Begriffswörterbuch, Mannheim, 1971

[14] Herbert Meschkowski : Mathematiker-Lexikon, Mannheim, 1973

[15] MU, Kalküle und Rechenautomaten, Der Mathematikunterricht, 11. Jahrgang, Heft 2, Stuttgart, 1965

[16] MU, Algorithmen, Der Mathematikunterricht, 18. Jahrgang, Heft 4, Stuttgart, 1972

[17] Harald Scheid : Einführung in die Zahlentheorie, Stuttgart, 1972

[18] Claus Peter Schnorr : Rekursive Funktionen und ihre Komplexität, Stuttgart, 1974

[19] B. A. Trachtenbrot : Wieso können Automaten rechnen?, 5. Aufl., Berlin, 1968