Frühe Zeugnisse der „software“

Friedrich L. Bauer

Ich erinnere mich noch gut daran, als ich Anfang der fünfziger Jahre zum ersten Mal auf das Wort ‚software‘ stieß. Es war ein Oxymoron, die rhetorische Figur der Verbindung zweier sich scheinbar ausschließender Begriffe: ‚soft‘, weich und ‚ware‘, Ware wie in ‚hardware‘, lt. *Cassel's* Eisenwaren. In Verbindung mit kleineren und größeren Rechenanlagen wurde das Wort von den Computeringenieuren etwas abschätzig gebraucht zur Bezeichnung der wenigen damals verfügbaren generellen Programmierhilfen, wie Überwacher-Programme für den Programm-Lauf, *post mortem*-Programme für Fehlerdiagnose, Interpretier-Programme für Kommandosprachen (SHORT CODE, 1949), Lader für Unterprogramme und Assemblier-Programme zur Speicherzuordnung (Wilkes et al. 1951), die über die schon seit 1947 für die EDSAC und für die BINAC vorgesehenen Bibliotheksprogramme (‚routines‘), beispielsweise für spezielle Funktionen, hinausgingen. Tatsächlich bestand für die Überheblichkeit der hardware-Ingenieure noch genügend Grund, hing es doch damals sehr von ihrem Geschick ab, ob die Maschinen zuverlässig liefen. Die Hersteller unterschätzten weithin die Bedeutung dieses neuen Zweiges der Technik, bei Telefunken war zu der Zeit, als die Ankündigung der Rechenanlage TR4 vorbereitet wurde, die Zahl der software-Entwickler deutlich unter der Zahl der hardware-Entwickler: die Fortschritte in der software geschahen damals überwiegend im akademischen Bereich. Die Software-Leute begannen jedoch zusehends, selbstbewusster zu werden und nutzten den Vergleich mit den ‚Eisenwaren‘ der Ingenieure, auf die intellektuelle Überlegenheit der reinen ‚Anweisungen an den menschlichen Geist‘, wie es im Patentgesetz so sinnig formuliert war, hinzuweisen. Meilensteine in der Verselbständigung der Software waren die programmiersprachlichen Übersetzungsprogramme (in den USA, 1952 aufkommend, etwas irreführend noch ‚compiler‘ genannt): FORTRAN von IBM (John Backus 1956), ALGOL 58 und ALGOL 60 von einer internationalen akademischen Gruppe (von Leuten der IBM Users Group SHARE als ‚austere academic body‘ geschmäht), LISP von John McCarthy. Die Hersteller der nunmehr ‚computer‘ genannten Rechenanlagen gingen jedoch nach und nach dazu über, selbst software anzubieten, als ‚kostenlose Zusatzleistung‘, deren Entwicklungskosten aber selbstverständlich über die verkaufte hardware ‚hereingeholt werden mussten‘ (Albert Endres, 1993). Unbestrittenes Vorbild, insbesondere für PASCAL und C, wurde ALGOL mit der Blockstruktur, den zusammengesetzten Anweisungen, den rekursiven Prozeduraufrufen, den geschachtelten Anweisungen, den der Länge nach unbeschränkten Variablennamen.

Rechenschemata

Lange vor dem Erscheinen programmierbarer Rechenmaschinen gab es die Notwendigkeit, umfängliche Berechnungsaufgaben mittels geeigneter Formulare aufzubereiten. Ein weithin bekanntes Beispiel ist das Horner-Schema, Abb. 1 zeigt es in einer Fassung aus den dreißiger Jahren. K. Zuse wurde damals durch ein Formular für Berechnungen von

DOI 10.1007/s00287-006-0114-8

Friedrich L. Bauer
Nördliche Villenstraße 19
82288 Kottgeisering

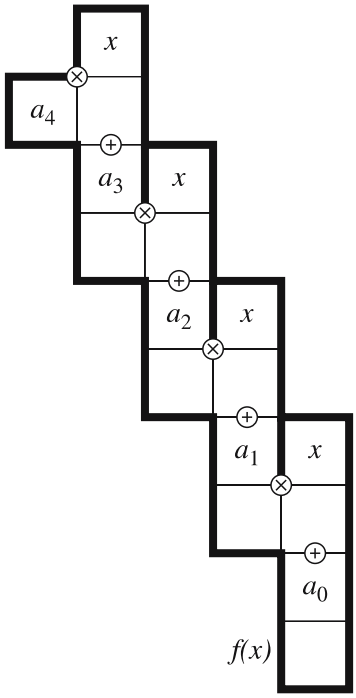

Abb. 1 Horner-Schema für $f(x) = a_4x^4 + a_3x^3 + a_2x^2 + a_1x + a_0 = (((a_4x + a_3)x + a_2)x + a_1)x + a_0$

Flächenmomenten, die in der Baustatik gebraucht wurden, auf seinen Weg zum Computer gebracht. Zu den umfänglichsten Aufgaben, für die man damals Tischrechenmaschinen heranzog, gehörte die Berechnung von Fourierkoeffizienten nach den Formeln von Carl Runge. Abbildung 2 zeigt eine Serie von zwölf Rechenblättern, die Paul Terebesi, auf Anregung von Alwin Walther in Darmstadt, 1930 publizierte. Formulare waren nach heutiger Sprechweise Programme, die an menschliche Rechner gerichtet waren, die entweder ‚im Kopf' rechneten oder mit einfachen Geräten, oft nur Rechenschieber oder Addiermaschinen. Formulare waren die Vorläufer der Software.

Turings namenlose Software-Idee

Alan Mathison Turing, damals Research Fellow in Teddington, sprach im Januar 1947, auf einem ‚Symposium on Large-Scale Digital Calculating Machinery' an der Harvard Universität, wohl als erster von einer neuen Idee, für die er noch keinen Namen hatte:

‚We are trying to make greater use of the facilities available in the machine to do all kinds of different things simply by programming rather than by the addition of extra apparatus.'

In einem konkreten Fall (‚recording on wire') gab er ein Beispiel und fasste zusammen:

‚Thus, we eliminate additional apparatus simply by putting in more programming.'

Es dauerte nur noch wenige Jahre, bis in der Schweiz an der Z4 Rutishauser, in England an der EDSAC Wilkes, Wheeler und Gill Taten folgen ließen.

Wer prägte den Ausdruck ‚software'?

Bereits 1850 finden sich die Wörter ‚software' und ‚hardware' für zweierlei Abfälle, solche die verrotten und solche, die das nicht tun. Es wird berichtet, dass im Zweiten Weltkrieg die Alliierten im U-Boot-Krieg von der ENIGMA als ‚hardware' und von den Chiffrierunterlagen der Deutschen, die auf Papier gedruckt waren, das sich im Wasser auflöste, als ‚software' sprachen. Vielleicht erklärt dies auch, dass von General Dwight D. Eisenhower der Ausspruch (1947) berichtet wird: ‚There will be no software in this man's army!'.

Etwa um 1957, als die ‚compiler' sich zu den ersten formelgesteuerten Übersetzungsprogrammen entwickelten und damit eine breite Verwendung sich abzeichnete, fand im Zusammenhang mit Rechenanlagen der Ausdruck ‚software' Verwendung. Im *Oxford English Dictionary* von 1960 ist er bereits aufgeführt. Die früheste bekannte Verwendung in der publizierten Literatur stammt vom Januar 1958 im *American Mathematical Monthly*, und zwar in einem Aufsatz ‚The teaching of Concrete Mathematics', der sich bemühte, ‚applied' oder ‚concrete' Mathematik intellektuell anziehend und anregend für Studenten zu machen. Der Verfasser, John Wilder Tukey (1915–2000), war seit 1950 Professor an der Princeton University und gründete 1965 dort das ‚Department of Statistics'. Sein Wirken war breit gefächert; zusammen mit James Cooley entwickelte er die Schnelle Fourier-Transformation (FFT) mit ihren weitreichenden Folgen für die Berechnung von Spektren. In dem erwähnten Artikel schrieb Tukey:

‚Today the „*software*" comprising the carefully planned interpretive routines, compilers and other aspects of automative (sic!) programming are at least as important to the modern electronic calculators as its „*hardware*" of tubes, transistors, wires, tapes and the like.'

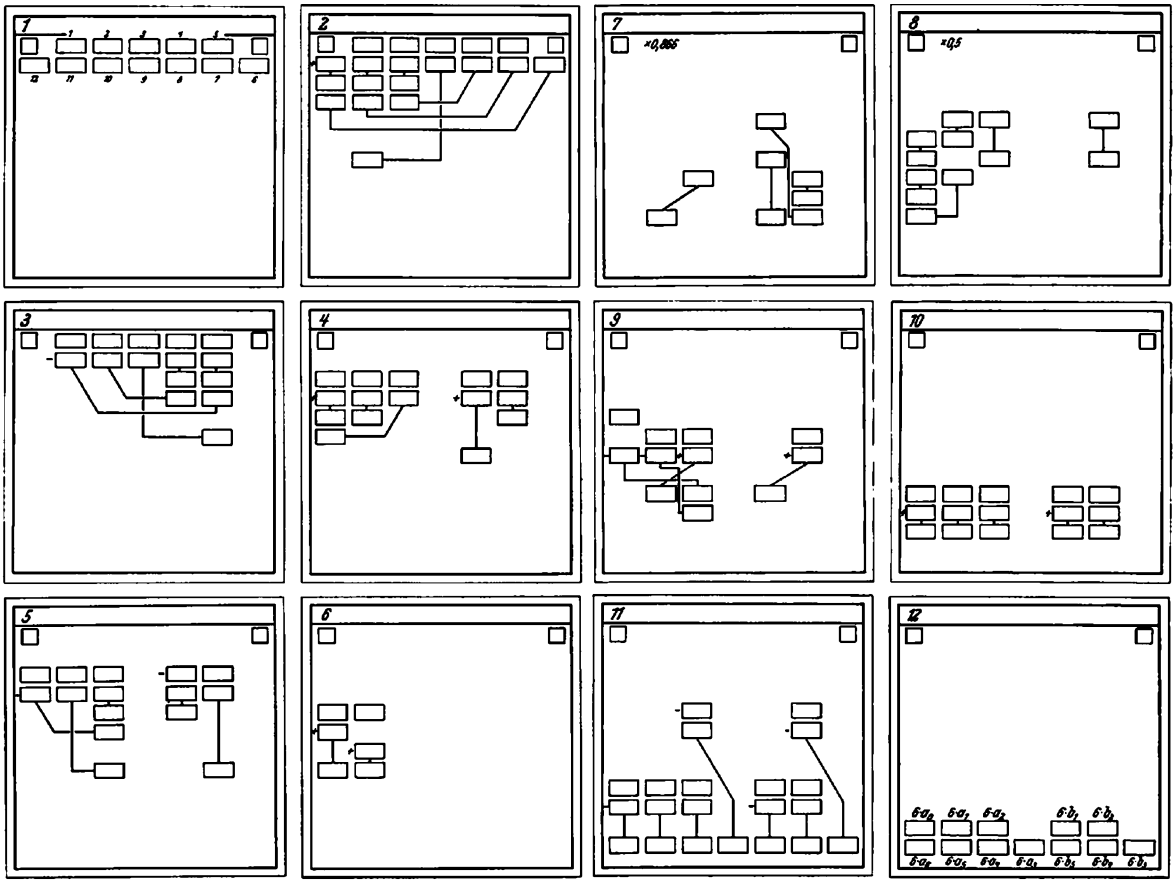

Abb. 2 Zwölf Terebesi-Schablonen (1930) für die Berechnung von Fourier-Koeffizienten

Tukey setzte *software* (und *hardware*) demonstrativ in doppelte Anführungszeichen, dadurch auf die Neuartigkeit der Wortbildung hinweisend. Paul Niquette behauptet, er habe das Wort schon 1953 geprägt.

Heinz Zemanek schrieb nicht ganz zu Unrecht ‚Software started with the translation of algebraic formulas into machine code'. Er bezog sich dabei auf Zuse und Rutishauser.

Rutishausers ‚Automatische Rechenplanfertigung'

Konrad Zuse hatte 1944 die Idee eines ‚Planfertigungsgeräts' (er benützte den Ausdruck ‚Plan' statt ‚Programm') zur Erleichterung der Programmierung der Z4, die er aber in den Wirren des letzten Kriegsjahrs nicht mehr weiterverfolgen konnte. Als er in der Nachkriegszeit, 1949, die in Hopferau, im Allgäu, gerettete Z4 an die ETH vermieten konnte, bekam er auch Kontakt mit Heinz Rutishauser, dem Assistenten von Eduard Stiefel. Womöglich erzählte er ihm von seiner Idee (für die er 1952 den Namen ‚Programmator' benutzte), aber Genaues darüber berichtete Rutishauser nicht. Zuse stellte das Planfertigungsgerät, das seinen ‚Plankalkül' (der im übrigen die erste hochentwickelte Programmiersprache war) verarbeiten sollte, nicht her; Howard Aiken rüstete jedoch noch 1952 die Maschine Harvard Mark III mit einer ‚Programming Machine' aus.

Rutishauser hatte jedenfalls schon 1951 den glänzenden Einfall, statt ein eigenes Gerät zu bauen, die Rechenanlage selbst zur Planfertigung zu benützen, also ein ‚programmierendes Programm', wie es Andrei Ershov bald darauf nannte[1] einzusetzen – konkret etwa auf der Züricher Z4. Rutishauser trug 1951 über sein Verfahren zur ‚Automatischen Rechenplanfertigung' auf einer Tagung vor, 1952 erfolgte die Publikation bei Birkhäuser

[1] als Buch aus dem Russischen übersetzt von M. Nadler, unter dem Titel *Programming Program for the BESM computer* bei Pergamon Press, London-New York-Paris 1959 erschienen.

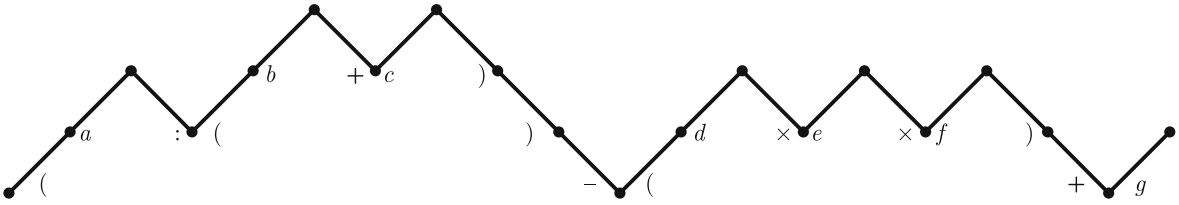

Abb. 3 Rutishausers Klammergebirge für vollständig geklammerte arithmetische Formeln

als ‚Mitteilungen aus dem Institut für Angewandte Mathematik, ETH Zürich, Nr. 3'. ‚That the same computer that solved a problem could prepare its own instructions was a critical moment in the birth of software' (Paul E. Ceruzzi, 1998).

Rutishausers Algorithmus war auf übliche mathematische Formeln mit zwei- und womöglich auch einstelligen Operationen und Klammerpaaren gerichtet. Er bewirkt den Abbau der durch die Klammern geschachtelten Strukturen, wie sie beispielsweise in der üblichen Infix-Schreibweise arithmetischer oder logischer Ausdrücke vorkommen, von innen nach aussen. Aus der Formel wird bei Rutishauser ein dem Syntaxbaum und der Schachtelstruktur äquivalentes ‚Klammergebirge' hergestellt, das die vorgeschriebene Auswertungsreihenfolge eindeutig festlegt. Das zugehörige Programm in einer Dreiadress-Sprache der Maschine ist unmittelbar ablesbar beziehungsweise erzeugbar.

Rutishauser gibt für die Formel $(a : (b + c)) - (d \times e \times f) + g$ ein Beispiel (Abb. 3), das den Aufbau des Klammergebirges zeigt: eine öffnende Klammer sowie ein Variablenzeichen erzeugen einen aufwärts gerichteten Strich, ein (zweistelliges) Operationszeichen sowie eine schließende Klammer erzeugen einen abwärts gerichteten Strich.

Ein gleichwertiges Vorgehen mit Hilfe der sogenannten Markov-Algorithmen gab 1954 A.A. Markov und 1959 G. Asser an.

Rutishausers Algorithmus baut das Klammergebirge vom jeweils absoluten Klammermaximum ausgehend ab. In der effizienten Fassung von Bottenbruch (1957) nimmt man sukzessive die rechnerische Ermittlung der jeweils lokal höchsten Gipfelkette mit anschließendem Abbau vor, indem man (von links nach rechts) die erste schließende Klammer sucht, sodann nach links gehend die erste öffnende Klammer (‚Abbau vom ersten Klammermaximum'). Der Algorithmus gleicht damit allerdings einer Springprozession über das Klammergebirge, bei der der Aufwand mit dem Quadrat der Länge des Ausdrucks ansteigt. Rutishauser nahm noch an, dass die Formel explizit geklammert ist, und unter dieser Annahme zeigte Corrado Böhm 1952, dass man die Auswertung auch sequentiell, normalerweise von links nach rechts, vornehmen kann. In üblicher Schreibweise wird jedoch unter Annahme einer Präzedenz der Multiplikation über der Addition und der Subtraktion auf die vollständige Klammerung verzichtet. In FORTRAN wurde ab 1954 (P.B. Sheridan) durch einen vorgeschalteten Durchlauf die Klammerung vervollständigt.

L. Kalmár machte dazu den witzigen Vorschlag, das Multiplikationszeichen $\times$ überall durch die Folge $) \times ($ zu ersetzen (und die ganze Formel extra einzuklammern). Die entstehenden redundanten Klammern störten Kalmár nicht. Dies brachte die Münchner Gruppe (von K. Samelson 1955 in Dresden vorgetragen) dazu, basierend auf dem Entwurf von 1951 des Logikrechners STANISLAUS auch im Fall der unvollständigen Klammerung unmittelbar eine ‚sequentielle Formelübersetzung' unter Verwendung eines Stapels für noch nicht auswertbare Teilstücke der Formel (‚Keller') vorzunehmen. Es entstand daraus das Kellerprinzip der Programmierung, das bald auf alle geklammerten Programmstrukturen ausgedehnt wurde und die Basis von ALGOL 58 war.

Das Kellerprinzip wurde im übrigen 1957 Gegenstand einer Patentanmeldung von F.L. Bauer und K. Samelson; es dürfte sich dabei, notdürftig verkleidet mit Hardware, um eines der ersten Software-Patente handeln (DBP 1 094 019, US Patent 3 047 228). Taschenrechner von HP arbeiteten nach diesem Patent.

Die EDSAC-Schule der Programmierung

1951 erschien das Buch von M.V. Wilkes, D.G. Wheeler und S. Gill *The Preparation of Programs for an Electronic Digital Computer*, das erste tiefergehende Lehrbuch der Programmierung. Neben der Einführung der relativen Adressen und

der Benützung von Bibliotheks-Routinen werden, ganz in der Turingschen Linie, interpretative Hilfsprogramme für Gleitkommarechnung oder Rechnung mit doppelter Genauigkeit auf einer Festkommamaschine vorgesehen. David Wheeler kommt das besondere Verdienst zu, die Benutzung symbolischer Adressen propagiert und in die Programmierung eingeführt zu haben. Von besonderer Bedeutung sind ferner die ‚interpretive routines' und die ‚assembly routines' (von Grace Hopper 1952 als ‚compiling routines' wiedererfunden) zur Benutzung parametrisierter Unterprogramme.

Ein ‚algebraic interpreter', der ‚Short Code', 1949 im Umfeld von John W. Mauchly konzipiert, wurde schließlich 1952 von J. Robert Logan für die UNIVAC codiert, fand aber kaum Beachtung. Auch der Versuch (1951) von Arthur W. Burks, Professor an der University of Michigan, mit Hilfe einer ‚intermediate programming language' eine Abstraktionsstufe höher als in Maschinencode zu programmieren, fand in der ‚business community' keine Gnade. In England schrieb Alick E. Glennie ein notationell interessantes Compilersystem, genannt ‚Autocode', für die Manchester MARK I Maschine, das 1952 in Gebrauch kam; ‚[it] had very little tangible impact' (Donald F. Knuth). 1954 gaben E.Z. Ljubimskii und Sergej Sergeevich Kamynin ihrem System den von Ershov erfundenen Namen ‚Programming Program'. 1951 schrieb R. Abbot ein Programm zur Auswertung logischer Formeln: *Computing Logical Truth with the California Digital Computer*, noch wie bei Rutishauser von innen nach aussen arbeitend.

‚High-Level Programming Languages': Laning und Zierler

Während 1952 die USA unter dem dominierenden Einfluss des retardierenden Howard Aiken noch gegenüber den Briten zurücklagen, änderte sich 1954 das Bild: Der Schritt zu den ‚High-Level Programming Languages' geschah hauptsächlich in den USA, öffentlich erstmals Januar 1954 durch J. Halcombe Laning und Neil Zierler. In ihrem Manual *A Program for Translation of Mathematical Equations for Whirlwind I*, das ab 1952 entstand, taucht das Wort ‚Übersetzung' (translation) erstmals auf, wodurch der Schritt zu Programmier*sprachen* offenkundig wird[2]. Trotz offensichtlicher Mängel des ‚Algebraic Systems': Beschränkungen in der Ausdrucksfähigkeit der Sprache und Langsamkeit in der Übersetzung, war es ‚an elegant concept elegantly realized' (John Backus). Obschon sich Charles W. Adams am MIT für das ‚Algebraic System' einsetzte, blieb es weithin unbeachtet.

Unter dem Einfluss der UNIVAC-Schule überwogen die altmodischen Assembler und Compiler noch ein paar Jahre, im Stillen aber vollzog sich bei IBM der Wandel: Gleichzeitig mit der Entwicklung eines Assemblers für die IBM 704 (1955) konnte ab Januar 1954 John Backus eine Gruppe von Mathematikern um sich versammeln, die die Ansätze von Laning und Zierler weiterführen sollten. Backus wurde anfänglich unterstützt von Irving Ziller, bald darauf von Harlan Herrick und von Robert A. Nelson. Die Programmiersprache sollte mächtiger werden und die Übersetzungszeit auf IBM-Maschinen sollte drastisch verkürzt werden. Diese leicht widersprüchlichen, aber praktisch gut begründeten Forderungen erzwangen einen Kompromiss. FORTRAN, wie das 1957 für die IBM 704 freigegebene System genannt wurde, war zwar erstaunlich schnell, aber es fehlte ihm die Eleganz der Jahrhunderte alten mathematischen Tradition. 1957 gab auch Remington-Rand das System Math-Matic, das von Charles Katz 1955 begonnen worden war, frei.

Die an der TH München gebaute, vom Vorbild Whirlwind beeinflusste, 1955 fertiggestellte binäre Gleitpunktmaschine PERM legte nahe, dass auch die seit 1955 laufenden Anstrengungen der Münchner Mathematiker, Instrumente für das ‚Automatische Programmieren' zu entwickeln, mehr den Ansätzen von Laning und Zierler als denen von Backus folgten, wobei sie im übrigen aber die Vorarbeiten von Rutishauser, insbesondere seine rekursive Unterprogramm-Technik, einbezogen. Das Kellerprinzip, das den Methoden der Springprozession den Garaus machte, ermöglichte, die Eleganz der mathematischen Tradition voll beizubehalten, ohne die Übersetzungsgeschwindigkeit im geringsten zu beeinträchtigen.

Analytische Differentiation und Integration

Es konnte in den frühen fünfziger Jahren nicht verborgen bleiben, dass die (elektronische) Rechenanlage mehr konnte als bloßes Rechnen mit Zahlen und Wahrheitswerten. 1953 begann Harry

[2] In den USA hielt sich jedoch, entgegen der ursprünglichen Bedeutung, lange der Ausdruck ‚compiler'.

G. Kahrimanian[3], Hilfsprogramme für formelmäßige Differentiation zu entwickeln. Er nahm an, dass der Syntaxbaum der zu differenzierenden Formel bereits in Drei-Adress-Form vorlag; die Aufgabenlösung lief darauf hinaus, diesen Baum unter Anwendung der Regeln für die Differentiation nach einer Variablen abzuarbeiten, Regeln für Summen und Differenzen, Produkte und Quotienten (seltsamerweise wurde die Quotientenregel nicht aufgelistet), Potenzen und Logarithmen sowie für einige wenige einstellige ‚elementare' Funktionen. Das Ergebnis wurde wieder als Syntaxbaum ausgegeben. Auch J.F. Nolan untersuchte 1953 die Analytische Differentiation. Erst 1961 wandte sich James R. Slagle[4] der nur partiell definierten Umkehrung der formelmäßigen Differentiation: der formelmäßigen Integration, zu – heuristisch, aber mit beachtlichem Erfolg.

Symbolisches Rechnen, Computeralgebra

Analytische Differentiation und Integration war nur ein Anfang, in dessen Gefolge Symbolisches Rechnen, auch Computeralgebra genannt, aufkam – wichtigste Stichworte sind: Vereinfachung und Vergleich algebraischer Ausdrücke, Faktorisierung von Polynomen, Überführung von Potenzreihen in Funktions-Kettenbrüche, Operationen der funktionalen Analysis; numerisches Rechnen mit beliebiger Genauigkeit.

Bald wurden auch blenderische Phrasen ins Spiel gebracht: Hatte schon 1949 Edmund C. Berkeley mit seinem Buchtitel *Giant Brains or Machines that think* Aufsehen erregt, so wurden mit dem Aufkommen der Software die Ansprüche noch weit höher geschraubt; die Spitze erreichte diese Manie mit der Bezeichnung *General Problem Solver* für eine erste definitiv nichtnumerische Programmiersprache, die Allan Newell und Herbert A. Simon 1959 kreierten, um aussagenlogische Beweise zu führen. John McCarthy schuf im selben Jahr zu diesem Zweck LISP (*List Processing Language*), das wenigstens sagte, was es tat. Heuristische Beweise für Theoreme der ebenen Geometrie programmierten schon 1958 Herbert Gelernter und Nathaniel Rochester, einen Höhepunkt erreichte 1960 Hao Wang (*Proving Theorems by Pattern Recognition*). Hochgestochene Ausdrücke wie *Intelligence Amplifier* (W. Ross Ashby, 1956), *Self-Organizing System* (J.K. Hawkins, 1961), und anscheinend bescheiden *Steps Towards Artificial Intelligence* (Marvin Minsky, 1961) charakterisierten eher die Intelligenz ihrer Erfinder, als den Gegenstand. Journalisten und andere Laien wurden dadurch häufig irregeführt. Unter den Fachleuten wurde *Artificial Intelligence* weithin zum Unwort.

SMP von Stephen Wolfram markierte weithin sichtbar 1979 den Beginn des ‚Scientific Computing', Wolframs MATHEMATICA wurde am 23. Juni 1988 in den Handel gebracht. MAPLE, entwickelt seit 1980 von Keith O. Geddes, Gaston H. Gonnet an der Universität von Waterloo, erschien bereits 1985 auf der Bildfläche. MATHEMATICA und MAPLE entwickelten sich zu kompletten Systemen des wissenschaftlichen Rechnens. Etwa zu dieser Zeit entstand auch DERIVE, das ein Nachfolger von MuMATH aus den 70er-Jahren war und von Soft Warehouse 1988 präsentiert wurde.

Vorläufer dieser weitverbreiteten Systeme war unter anderem das in den 60er-Jahren von Anthony C. Hearn begonnene REDUCE, das sich stark auf LISP stützte und 1968 in den Handel kam. Ein anderer Vorläufer war bereits 1968 von Carl Engelmann und Mitarbeitern begonnene MACSYMA, das unter dem Patronat des MIT entstand.

Neuere Systeme, die sich zu ebenbürtigen Konkurrenten von MATHEMATICA, MACSYMA und MAPLE entwickelten, sind MATHLAB seit 1984, MATHCAD seit 1985, MUPAD seit 1992. Spezialisiert sind u.a. MAGMA und PARI-GP für algebraische Zahlentheorie, SnapPea für Topologie, GAP für Gruppentheorie, und MathXpert (1997) von Michael J. Beeson für Beweiszwecke.

Symbolisches Rechnen hat sich nämlich inzwischen ausgeweitet zur rechnerischen Durchführung von Beweisen. Der Beweis des Vierfarbensatzes von Kenneth Appel und Wolfgang Haken (1976) durch ausgiebige Fallunterscheidungen gab ein erstes Beispiel – ohne Rechnereinsatz wäre sowohl die Organisation des Beweises wie auch die Überprüfbarkeit seiner Richtigkeit illusorisch gewesen. Zwischenzeitlich haben manche Aufgaben auch die Leistungsfähigkeit der Reinen Mathematiker auf den Plan gerufen. Das anfängliche Geschrei der Orthodoxen, eine solche ‚Mechanisierung der Mathematik' sei ihr Tod, wurde konterkariert durch die Freudenschreie der naiven Anhänger der ‚Künstlichen Intelligenz'. Tatsächlich muss eine kritische Betrachtung in der Mitte liegen: Nicht die Mathematik wird

[3] Proc. Symposium on Automatic Programming for Digital Computers, Office of Naval Research, 13–14. May 1954; PB 111607

[4] J. ACM 10, 507–520

mechanisiert, sondern nur eines ihrer Werkzeuge. Dabei ist jede Menge Einfallsreichtum gefordert; der Reine Mathematiker wird keineswegs arbeitslos. Im Gegenteil:

Die ‚experimentelle Mathematik' führt zu Entdeckungen, wie Jonathan Borwein, Peter Borwein und Karl Dilcher sie machten. Sie wurden 1988 von R.D. North darauf hingewiesen, dass die sehr langsam konvergierende Reihe von Gregory-Leibniz mit 500 000 Termen

$$4 \cdot \sum_{k=1}^{500\,000} \frac{(-1)^{k-1}}{2k-1} =$$

3.14159065358979324046264338326950288419 7

– sie liefert nur fünf korrekte Nachkommastellen von π — nichtsdestoweniger mit π ‚fast überall' übereinstimmt, nämlich nur an den unterstrichenen Stellen nicht. Erst einmal darauf hingewiesen, hatten sie nur geringe Mühe, die Erklärung zu finden, die mit der ‚runden' oberen Grenze 500 000 des Indexes zusammenhängt, und fanden für

$$2 \cdot \sum_{k=1}^{50.000} \frac{(-1)^{k-1}}{2k-1} =$$

1.57078632679489761923132119163975205209858331 4

gegen $\frac{\pi}{2} =$

1.57079632679489661923132169163975144209858469 9

die Abweichungen:

1 –1 5 –61 1385

in Sloanes *Handbook of Integer Sequences* als ‚*Euler numbers*'. Damit wurden sie auf die folgende asymptotische Entwicklung geführt:

$$\frac{\pi}{2} - 2 \cdot \sum_{k=1}^{N/2} \frac{(-1)^{k-1}}{2k-1} \approx \frac{1}{N} + \frac{-1}{N^3} + \frac{5}{N^5} + \frac{-61}{N^7} + \ldots .$$

Ein gänzlich rein-mathematisches Problem ist das folgende: Die drei Gleichungen

$x + y = y + x$ (Kommutativgesetz)

$(x + y) + z = (x + y) + z$ (Assoziativgesetz)

$\neg\big(\neg(x + y) + \neg(x + \neg y)\big) = x$ (Robbins Gesetz)

implizieren nach Herbert Ellis Robbins (1922–2001)

$\neg(\neg x + y) + \neg(\neg x + \neg y) = x$ (Huntingtons Gesetz).

Courants Schüler Robbins stellte die Frage: Gilt die Umkehrung auch? Folgen aus Huntingtons Gesetz die üblichen drei Gesetze, die eine Boolesche Algebra charakterisieren?

Ein Programm namens EQP, von William McCune geschrieben, löste die Frage positiv mit einer Laufzeit von acht Tagen (im Jahr 1996!) und einem Speicherbedarf von 30 Megabytes. Der schließlich ausgedruckte Beweis war jedoch nur 15 Zeilen lang und ging auf eine einzige Druckseite. ‚*You sometimes have to shovel a lot of dirt and gravel to find a diamond*' (M.J. Beeson). Exhaustive Suche (*brute force attack*) ist wenigstens eine *Methode*, aber sie verbietet sich oft aus Aufwandsgründen.

EQP tat nichts als einen Beweis zu finden in der Hoffnung des Benutzers, es gäbe einen. Vermutungen aufzustellen ist eine andere Sache und scheint der Mechanisierung keine Freude zu bereiten, außer man hat nichts dagegen, viele unbeweisbare Vermutungen in die Welt zu setzen.

Ein weiteres, gut überschaubares Beispiel liefern nicht-lineare algebraische Gleichungssysteme, etwa das folgende, aus einem von Jürgen Richter-Gebert behandelten kinematischen Problem herrührend:

$$0 = a^2 + 6a + c^2 - 27$$
$$0 = -6b + b^2 + d^2 - 27$$
$$0 = a^2 - 2ab + b^2 + c^2 - 2cd + d^2 - 100$$
$$0 = a + b - 2x$$
$$0 = c + d - 2y$$

MATHEMATICA liefert auf einem Apple PowerBook G4 (1GHz) in weniger als einer Zehntelsekunde unter sukzessiver Elimination von a, b, c, d eine Gröbner-Basis von 14 Polynomen, deren letztes Element zu der Gleichung führt

$$x^6 + 3x^4y^2 - 40x^4 + 3x^2y^4 - 44x^2y^2 + 400x^2 + y^6 - 4y^4 - 896y^2 = 0\,.$$

Schon im 19. Jh. gab es dank Leopold Kronecker (1823–1891) Methoden, um im Prinzip solche Probleme zu lösen, aber erst die Einführung der Gröbner-Basen (eine Verallgemeinerung der Gauß-Elimination ins Nicht-lineare) durch Wolfgang Gröbner (1899–1980) erlaubte Bruno Buchberger 1965, einen allgemeinen Algorithmus für diese Aufgabe zu konstruieren (der auch von MATHEMATICA verwendet wird); eine Leistung, die den Beginn der Computer-Algebra einläutete. Es handelt sich

hier um ein besonders geeignetes Beispiel, weil die Berechnung von Gröbner-Basen so aufwendig werden kann, dass eine Durchführung ‚von Hand' sich praktisch verbietet.

Das Beweisen von Sätzen, das schon 1960 faszinierend war, zieht auch heute die besten Mathematiker an und bedarf auch der Besten: Die Schwierigkeien sind nach wie vor enorm. Derzeit scheint Larry Wos mit dem System OTTER einen Stern am Himmel der maschinellen Beweise in der Prädikatenlogik erster Stufe zu haben.

Frühe nichtnumerische Software

Programmierte Sprachübersetzung einerseits, programmiertes Schachspiel andererseits traten früh in Erscheinung.

Die Idee einer automatisierten Sprachübersetzung wurde 1933 angestoßen durch den russischen Erfinder P.P. Smirnov-Troyanskiy. Der konsequente nächste Schritt erfolgte, kaum dass die großen elektronischen Rechenanlagen auf den Plan getreten waren, schon 1946 durch Warren Weaver, der in einer Diskussion mit Andrew Booth die Ansicht vertrat, mittels Rechenanlagen müsste eine automatische Übersetzung natürlicher Sprachen möglich sein. Mitte 1952 fand am MIT die ‚First International Conference on Machine Translation' statt; IBM verkündete 1954, dass auf einer IBM 701 eine programmierte Sprachübersetzung Russisch-Englisch gelungen sei; ein diesbezügliches Buch *Machine Translation of Languages* erschien 1955 bei Wiley, New York.

Angeblich mechanisiertes Schachspiel wurde schon von 1769 von Wolfgang von Kempelen, einem Höfling der Kaiserin Maria Theresia vorgeführt, nach Edgar Allen Poe tatsächlich eine Täuschung. Einen mechanischen Apparat für das Endspiel König und Turm gegen König baute 1912 G. Torres y Quevedo. Nachdem Norbert Wiener 1948 kritische Bemerkungen zur programmierten Durchführung des Schachspiels gemacht hatte und Claude Shannon 1950 (*Programming a Computer for Playing Chess*) grundsätzliche methodische Überlegungen publiziert hatte, skizzierte Alan Turing 1951 ein Programm für eine hypothetische Maschine, wobei ihm Verfeinerungen der Methodik (‚stabile Stellung') gelangen. 1952 machte D.G. Prinz den ersten erfolgreichen Versuch, ein Schach-Programm zum Laufen zu bringen, und zwar auf der Maschine Mark I in Manchester. Er beschränkte sich jedoch auf Endspiele. 1953 beschäftigte sich Shannon gründlicher mit den kombinatorischen Problemen des vollen Spiels, aber erst 1956 werden von P. Stein und Stan Ulam auf dem MANIAC in Los Alamos die Arbeiten wieder aufgenommen, eingeengt auf ein Spielfeld von 6-mal-6 Plätzen und ohne Läufer und Bauern, mit einer Vorausschau von zwei Zügen (d.h. von vier Halbzügen). Die Einschränkungen waren gerechtfertigt: jeder Zug brauchte eine Rechenzeit von ungefähr 12 Minuten, das Programm hatte einen Umfang von rund 600 Befehlen. Das Programm war in der Lage, gegen einen Anfänger zu gewinnen. 1957 konnten A. Bernstein und M. de V. Roberts über einen erfolgreichen Versuch auf der IBM 704 ohne Beschränkung des Spielfelds berichten. Bei der gesteigerten Rechengeschwindigkeit von 42 000 Operationen pro Sekunde brauchte ein Zug etwa acht Minuten, das Programm bestand aus rund 7000 Befehlen und konnte gegen einen Amateur mit gelegentlichen Schwächen gewinnen; es wurde einmal nach 22 Zügen besiegt. 1958 war mit einem Programm von A. Newell, J.C. Shaw und H.A. Simon und einer ausführlichen Publikation der Durchbruch geschafft. Von nun an ging es jahrelang stetig aufwärts. Konrad Zuse, der Schach als Musterbeispiel für seine Plankalkül-Programmierung schätzte, ließ sich (wie er in seiner Biographie berichtete, etwa 1938) auf eine Prognose von ungefähr 50 Jahren ein, bis ein Programm auf einer geeigneten Maschine einen Großmeister schlagen würde. 1997 trat das tatsächlich ein: ‚Deep Blue' besiegte Garry Kasparov nach Standardregeln des Turnier-Schachs.

Demgegenüber verblassten bald die Anstrengungen, andere Brettspiele dem Computer zu unterwerfen. Für ein primitives Spiel wie das Streichholzspiel NIM war schon 1901 durch Charles Leonard Bouton die geschlossene Theorie aufgestellt worden; 1951 besiegte NIMROD, ein bei Ferranti gebautes Maschinchen, dementsprechend leicht in einem sensationellen Spiel Adenauers Wirtschaftsminister Ludwig Erhard. Das schon anspruchsvollere Dame-Spiel wurde von Christopher S. Strachey 1952 angegangen, 1956 hatte A.L. Samuel schon ein gutes und 1959 ein ausgezeichnetes Programm verfügbar gemacht, das es schon mit Meistern aufnehmen konnte.

Auch Lernprogramme wurden entwickelt, so für die EDSAC (A.G. Oettinger, 1952: *Programming a Digital Computer* to Learn), ferner Programme

für Literatursuche (Yehoshua Bar-Hillel, 1954: *Logical Syntax and Semantics*; J.W. Parry and A. Kent 1956: *Machine Literature Searching*); und später viele andere Ansätze. die sich, wie etwa ein von L.A. Hiller und L.M. Isaacson geschriebenes Programm, das 1959 auf der ILLIAC Musikstücke komponierte, als mehr oder weniger relevant erwiesen.

Betriebssysteme und System-Software

Viele herausragende Schritte zur Entwicklung des selbständigen Begriffs ‚software' waren bis etwa 1958 geschehen. Die zunehmende Verwendung von großen Speichern, Ein-Ausgabegeräten und Parallelarbeit und die hohe Rechengeschwindigkeit machte es dann ab 1958 dringend erforderlich, auch den menschlichen Operateur mehr und mehr durch die Maschine zu ersetzen (MAD an der University of Michigan: Bernie Galler, Bob Graham, Bruce Arden 1959); es entstanden in den sechziger Jahren mehr und mehr ausgeklügelte Betriebssysteme (‚monitors, supervisor systems, operating systems') und die Verwendung einer System-Software: Binder und Lader, Fehlersucher (debugger), Speicherbereiniger. Ab etwa 1960 bestimmte die Software für den Benutzer das Erscheinungsbild der Computer. Pioniere der Betriebssysteme waren in Deutschland Gerhard Seegmüller und Hans-Rüdiger Wiehle, die 1963 die Arbeit an einem Betriebssystem für die TR 4 von Telefunken begannen.

Die ‚software crisis'

Spätestens 1967 gab es dann sogar eine ‚software crisis': Das US Verteidigungsministerium wurde nervös, als bestellte Computersysteme wegen Mängeln in der Software ihre Zweckbestimmung nicht voll erfüllten und die Sicherheit der Vereinigten Staaten in der Zeit des Kalten Krieges gefährdet schien. Tatsächlich war der Wildwuchs, der mit der akademischen Freiheit verbunden ist, gelegentlich überbordend und führte zu unnützen Arbeiten von Bastlern und „Bit-Fummlern". In einer ‚Study Group on Computer Science', die auf Anregung des US-Delegierten im NATO Science Commitee, Dr. Isidor Rabi, 1967 eingesetzt wurde und zu der ich delegiert wurde, platzte mir bei einer Debatte über die Gründe für die software crisis der Kragen, und ich sagte: ‚The whole trouble comes from the fact that there is so much tinkering with software', und als ich merkte, dass ich einige meiner akademischen Kollegen schockiert hatte, setzte ich nach: ‚What we need is software engineering'. Zur Strafe dafür durfte ich dann die vom NATO Science Committee geförderte erste Konferenz über Software Enineering, 7.–11. Oktober 1968 in Garmisch ausrichten. ‚The conference marked the end of the age of innocence' (Paul E. Ceruzzi, 1998). Software war nach 16 Jahren erwachsen geworden, hatte sich zu einem Wirtschaftsfaktor gemausert, war zum Gegenstand von Sicherheitsbedenken aufgerückt und ist schließlich heute ein Alltagsobjekt.