Herbert Melenk 2
REDUCE

Michael Monagan 6
Research and Development in Maple

Roman Maeder 10
Mathematica

Johannes Grabmeier 13
Axiom, ein Computeralgebrasystem mit abstrakten Datentypen

Allan Bonadio 16
Theorist - a User Interface for Symbolic Algebra

A. Kemper, O. Kluge, K. Morisse, H. Naundorf, W. Wiwianka und B. Fuchssteiner 19
MuPAD

Bernhard Kutzler 21
Der Mathematik-Assistent DERIVE Version 2

Gerhard J.A. Schneider 25
CAYLEY - Eine kurze Beschreibung

Kurt Behnke 27
MACAULEY - ein Computeralgebrasystem für Algebraische Geometrie und Kommutative Algebra

C. Batut, D. Bernardi, H. Cohen, M. Olivier und N.-P. Skoruppa 30
The PARI-GP package

R. Bündgen, G. Hagel, R. Loos, S. Seitz, G. Simon, R. Stübner und A. Weber 33
SAC-2 in ALDES
Ein Werkzeug für die Algorithmenforschung

Martin Schönert 38
GAP- Groups, Algorithms and Programming

A. Giovini und G. Niesi 42
CoCoA: A User-Friendly System for doing Computations in Commutative Algebra

U. Schneiders und P. Serf 44
Das Algorithmen-System SIMATH in der Zahlentheorie

M.E. Pohst und N. Schulte 47
KANT - ein Software-Paket für zahlentheoretische Anwendungen

Marc A.A. van Leeuwen 49
LIE, a software package for Lie group computations

Heinz Kredel 51
MAS - Modula-2 Algebra System

Peter Dräxler 55
Das Darstellungstheorie-Progammpaket CREP

Volkmar Felsch 56
SOGOS und SPAS

J. Apel und U. Klaus 57
FELIX - ein Computeralgebra-System für konstruktive Algebra

B. Fuchssteiner und W. Wiwianka 61
Symmetriesuche per Computer

Zum Titelbild:
M. Jünger und W.R. Pulleyblank 64
Minimale aufspannende Bäume in der Euklidischen Ebene mit graphischem Optimalitätsbeweis

Aufgelesen 66

Liebe Leserin, lieber Leser

Vom 16. bis zum 20. September findet in Bielefeld die Jahrestagung der Deutschen Mathematiker-Vereinigung statt.
Sie halten das mathPAD-Sonderheft zu dieser Tagung in den Händen. Alle Systeme, die auf der für diese Tagung organisierten Computeralgebrademonstration vorgestellt werden, sind hierin beschrieben. Wir sind den Autoren für die Mühe, die sie damit auf sich genommen haben, dankbar.
Unser besonderer Dank gilt *Michael Jünger* und *William R. Pulleyblank* für das schöne Titelbild nebst Beschreibung.
Neu aufgenommen wurde die Kolumne *Aufgelesen*. Wir wollen diese Spalte auch in Zukunft fortsetzen.

Redaktion: Benno Fuchssteiner,
Waldemar Wiwianka und Klaus Hering
Universität - GH Paderborn
D - 4790 Paderborn
E-mail: mathpadjournal@uni-paderborn.de

REDUCE

Herbert Melenk, Konrad-Zuse-Zentrum für Informationstechnik BERLIN

Einleitung

Die erste Version von REDUCE wurde Ende der sechziger von Anthony C. Hearn entwickelt und publiziert. Ausgangspunkt waren damals die formalen Rechnungen in der Hochenergiephysik (Feynman-Graphen, Wirkungsquerschnitte etc.), die von Hand nur schwer und zeitraubend fehlerfrei durchführbar sind. Obwohl das heutige REDUCE vom Leistungsumfang her mit den damaligen Ansätzen überhaupt nicht mehr vergleichbar ist, stimmt doch die Ausrichtung auf schwierige mathematisch - formale Rechnungen im Bereich der angewandten Mathematik (Natur - und Ingenieurwissenschatfen) noch überein, wenn auch mit einem erheblich erweiterten Spektrum.
Heute läuft REDUCE auf einer Vielzahl von Rechnerplattformen, vom DOS Personal Computer bis hinauf zum Cray Supercomputer.
Trotz seines für ein Programmsystem hohen Alters wird REDUCE in einem kontinuierlichen Prozess ständig weiterentwickelt und erneuert. Mitte 1991 wurde REDUCE in der Version 3.4 freigegeben. Information über die verfügbaren Implementierungen erhält man über electronic Mail, indem man die Botschaft 'send info-package from info' an 'redlib@elib.zib.berlin.de' oder 'reduce-netlib@rand.org' schickt. An diesen Stellen erhält man auch Information über die zahlreichen verfügbaren Anwenderpakete von REDUCE.

1 Problemlösung

REDUCE ist primär ausgerichtet auf die Lösung mathematischer Probleme im Bereich technisch - wissenschaftlicher Anwendungen, und zwar insbesondere auf größere Probleme mit sehr umfangreichen oder schwierigen formalen Rechnungen. Dementsprechend finden sich zwar in REDUCE durchaus Operatoren, die eine Aufgabe unmittelbar und vollständig lösen, wenn z.B. kleinere Gleichungssysteme zu lösen oder Determinanten (mit numerischen und / oder symbolischen) Elementen zu berechnen sind. Typischer ist jedoch, daß erst eine Kombination von Einzelschritten eine gewünschte Antwort ergibt und deshalb wird bei der Entwicklung von REDUCE vor allem Wert auf die Bereitstellung leistungsfähiger und modularer Operatoren gelegt, die wie Bausteine zum Zusammenstellen einer Problemlösung genutzt werden.
In Einzelfällen kann es sogar erforderlich sein, einen vollständig neuen symbolischen Kalkül einzurichten. REDUCE bietet dafür einen reichen Satz von Schnittstellen auf mehreren Ebenen an, und die Bibliothek enthält zahlreiche Vorbilder.

2 Datentypen, Strukturen

2.1 Elementare Ausdrücke

Das zentrale Objekt in REDUCE ist der formale Ausdruck, der nach den üblichen mathematischen Regeln aufgebaut ist aus

- Zahlen (ganz, rational, gerundet, reell oder komplex)
- Symbolen (Namen, evtl. indiziert)
- Funktionsausdrücken
- Operationssymbolen $+, -, *, /, **$
- Klammern für die Steuerung der Präzedenz.

Ein Symbol kann hierbei die Rolle eines Stellvertreters für einen unbestimmten Wert übernehmen (formale Variable), ihm kann aber auch ein Ausdruck als Wert zugewiesen werden. Im zweiten Falle wird sein Wert immer dann eingesetzt, wenn das Symbol in einem Ausdruck zitiert wird.
Beispiele für elementare Ausdrücke:

```
3.1415928       % Bruch
a               % einfache Variable
(x+y)**2 / 2    % quadratischer Ausdruck
log(u)+log(v)   % Funktionen
```

2.2 Aggregate

Daneben gibt es die Datenstrukturen, die Aggregate von formalen Ausdrücken darstellen:

- Ein `Liste` ist eine lineare Anordnung von Ausdrücken

```
{2,3,5,7,11,13,17,19}
```

- Ein **Array** ist eine „rechteckige“ mehrdimensionale indizierte Struktur
- Eine **Matrix** ist eine Struktur aus Zeilen und Spalten

Zwischen Matrizen passender Dimension bzw. zwischen Matrizen und Skalaren sind arithmetische Operationen entsprechend den Regeln der linearen Algebra definiert. Daneben können Elemente einer Matrix einzeln wie Arrayelemente angesprochen werden.

3 Programmierparadigmen

Für die Formulierung von symbolischen Auswertungen und Algorithmen bietet REDUCE verschiedene Paradigmen an.

3.1 algebraischer Tischrechner

Der einfachste Zugang ist, REDUCE wie einen Tischrechner für symbolische oder numerische Formeln zu benutzen. Formeln können miteinander verknüpft, für die spätere Weiterverwendung benannt und mit einer Reihe leistungsfähiger Operatoren wie z.B. Differentiation, Integration, polynomialer GGT, Faktorisierung und vieles mehr, bearbeitet werden. Jede unmittelbar eingegebene Formel wird sofort bearbeitet mit dem Ziel, die bestmögliche Vereinfachung zu finden, und das Resultat wird ausgegeben, sobald es verfügbar ist.
Beispiel: Taylorpolynom für $x * sin(x)$

```
for i:=0:5 sum
  sub(x=0,df(x*sin(x),x,i)) * x**i
           / factorial(i);

    1   4    2
 - ---*X  + X
    6
```

3.2 imperative algebraische Programmierung

Die einzelne Formelauswertung mit sofortiger Resultatausgabe ist nur ein Spezialfall für eine Anweisung aus der REDUCE Programmiersprache. Diese gestattet die Formulierung komplexer Auswertungsaufträge wie bedingte Anweisungen, Anweisungsgruppen, Blöcke, iterative Schleifen (über eine Zählvariable oder über die Elemente einer Liste), bis hin zur Definition von parametrisierten Prozeduren mit lokalen Variablen.
Beispiel: Rekursives Programm für das Erstellen einer Basis aus Legendre Polynomen nach der Rekursionsformel
$P_{n+1}(x) = ((2n+1)xP_n(x) - nP_{n-1}(x))/(n+1)$
Der infixe Operator „.“ fügt ein neues Element vorn an eine Liste an.

```
procedure Legendre_basis(m,x);
  % Start mit Basis der Ordnung 1
   Legendre_basis_aux(m,x,1,{x,1});

procedure Legendre_basis_aux(m,x,n,ls);
    % ls enthaelt Polyonme n, n-1, n-2 ...
  if n>=m then ls     % fertig
  else Legendre_basis_aux(m,x,n+1,
  (((2n+1)*x*first ls - n*second ls)/(n+1))
       . ls);
```

Ein Aufruf dieser Prozedur:

```
Legendre_basis(3,z);

  5   3    3
{---*Z  - ---*Z,
  2        2

  3   2    1
 ---*Z  - ---,
  2        2

 Z, 1}
```

3.3 regelorientierte Programmierung

Globale algebraische Zusammenhänge können in REDUCE mit Hilfe von Regeln deklariert werden. Eine Regel beschreibt, daß formale (Teil-)Ausdrücke eines bestimmten Musters durch andere Ausdrücke rekursiv zu ersetzen sind, evtl. unter Berücksichtigung gewisser Einschränkungen. Derartige Regeln lassen sich sowohl global aktivieren, als auch lokal explizit auf ganz bestimmte Auswertungsschritte beschränken.
Es können auch ganze Kalküls auf diese Weise definiert werden, wobei die Regelform der üblichen mathematischen Notation (Fallunterscheidung) recht nahe kommt.
Beispiel: Definition von Hermite-Polynomen basierend auf einer Rekursionsformel:

```
operator Hermite;
Hermite_rules:=
{Hermite(0,~x) => 1,
 Hermite(1,~x) => 2*x,
 Hermite(~n,~x) => 2*x*Hermite(n-1,x)
          -2*(n-1)*Hermite(n-2,x)
              when n>1};
let Hermite_rules;
```

Erzeugen eines Hermite-Polynoms:

```
Hermite(4,z);
    4       2
16*Z  - 48*Z  + 12
```

3.4 symbolische imperative Programmierung

Die bisher beschriebenen Porgrammiermodi beziehen sich auf eine rein algebraische Interpretation der Daten; dies erlaubt eine anwendungsnahe Formulierung mathematischer Zusammenhänge (Regeln) und Auswertungsschritte (Prozeduren). Die Formeln werden für jeden Schritt einzeln intern in die für die Auswertung benötigte Form konvertiert. Bei umfangreichen und zeitaufwendigen Algorithmen kann es zur Einsparung von Rechenzeit sinnvoll sein, die Hin- und Rückkonvertierung der Zwischenergebnisse zu umgehen und direkt auf die internen Routinen von REDUCE zuzugreifen.

4 Algebraische Auswertung

Die Auswertung von Ausdrücken stellt das Herzstück von REDUCE dar und kann wegen der großen Komplexität hier nur angedeutet werden. Von zentraler Bedeutung bei der maschinellen Bearbeitung von mathematischen Formeln generell ist das Problem der Entscheidbarkeit der Identität zweier Formeln, z.B. die Entscheidung $a + b = b + a$ unter der Annahme einer kommutativen Addition.
Es ist wohlbekannt, daß dieses Problem äquivalent ist mit dem Problem der Erkennung der Null bzw. mit der Existenz eines endlichen Algorithmus für die Überführung einer beliebigen Formel in eine äquivalente kanonische Form. Eine universelle kanonische Form gibt es nicht; nur für Teilbereiche wie z.B. Polynome, rationale Funktionen, Ideale sind kanonische Formen bekannt. REDUCE baut deshalb auf einer kanonischen Form für rationale Funktionen (=Quotienten von multivariaten Polynomen) auf, bei denen als Variablen (REDUCE: Kerne) sowohl einfache Symbole wie auch Funktionsausdrücke mit beliebig komplexer Parameterstruktur auftreten dürfen. REDUCE versucht, mit geeigneten Regeln möglichst viele der Funktionen in den Bereich der systematischen kanonischen Form einzubeziehen. Für Funktionen, die sich dem generellen algorithmischen Zugriff entziehen (z.B. Wurzeln) wird versucht, mit Hilfe heuristischer Regeln ein akzeptables Verhalten zu erreichen.

5 Näherungen

Beim Umgang mit symbolischen Ausdrücken wird meist exaktes Rechnen verlangt, insbesondere wenn Algorithmen aus dem Bereich der klassischen Algebra zum Einsatz kommen. Hierfür bietet REDUCE als Zahlbereiche ganze Zahlen beliebiger Länge und - darauf aufbauend - rationale Zahlen, modulare Zahlen (= Rechnen im endlichen Restklassenkörper), auf Wunsch um die imaginäre Einheit i zu komplexen Zahlen erweitert.
Mit diesen Zahlen kann beliebig gerechnet werden. Jedoch ist der Auswertung von transzendenten Funktionen dadurch eine Grenze gesetzt, daß ihr Resultat mit Hilfe des aktiven Zahlbereiches (erweitert um Symbole wie π, e) exakt dargestellt werden kann.
Rechnerunterstütztes symbolisches Rechnen kann aber auch außerhalb der klassischen Algebra, etwa im Zusammenhang mit Näherungsverfahren der Analysis oder für klassische numerische Mathematik mit Gewinn eingesetzt werden.

5.1 Potenzreihen

Für die Ermittlung formaler Näherungen bietet REDUCE Potenzreihen an, und zwar wahlweise abgebrochene Taylorreihen in mehreren Veränderlichen oder in einer Veränderlichen Taylorreihen mit variabler Anzahl von Termen. Potenzreihen können einerseits als abstrakte algebraische Objekte aufgefasst werden, andererseits auch als Näherungen gesuchter Funktionen, z.B. im Kontext von Differentialgleichungen.

5.2 Gerundete Zahlen

Auch Gleitkommazahlen sind in REDUCE enthalten, jedoch sind sie gegenüber den üblichen in Hardware realisierten Gleitkommazahlen in zweifacher Hinsicht erweitert:

- die Mantissenlänge ist in beliebiger Größe frei wählbar,
- es gibt keine obere Grenze für den Betrag einer Zahl, d.h. der Exponent ist nicht beschränkt.

Technisch wird dies erreicht, indem die hardwaremäßig angebotenen Gleitkommazahlen in parametrisierte Software eingebettet werden.

5.3 Schnittstelle zu externen numerischen Programmen

Einen wichtigen Einsatzbereich der Formelmanipulation bilden die gemischt symbolisch-numerischen Verfahren, wenn z.B. der symbolische Teil formale Eigenschaften einer Aufgabe zur Vorbereitung und Konditionierung eines numerischen Algorithmus umsetzt. Beispiele sind etwa die automatische Herstellung von Jacobi-Matrizen für ODE-Solver oder die Reduktion der Ordnung eines Systems auf Grund formaler Symmetrien. Für die Zusammenarbeit von symbolischen und numerischen Komponenten eines Verfahrens bietet REDUCE hier die Erzeugung kompletter Programme in den Programmiersprachen FORTRAN, PASCAL oder C an.

6 Ein/Ausgabe

Normalerweise gibt REDUCE Resultate im interaktiven Betrieb auf dem Bildschirm in einer zweidimensionalen Form aus, wobei z.B. Exponenten nach oben gerückt sind und Quotienten wenn möglich mit einem langen Bruchstrich dargestellt werden. Neben dieser an der üblichen mathematischen Notation orientierten Ausgabeform bietet REDUCE weitere spezielle Ausgabeformen an:

- lineare Ausgabeform: die Daten sind zur späteren Wiedereingabe in REDUCE oder ein anderes System geeignet,
- Fremdsyntax: die Ausdrücke werden in der Syntax von FORTRAN, C oder einer anderen Programmiersprache ausgegeben zur Einfügung in numerische Codes,
- TEX: direkte Aufbereitung der Formeln für die Verwendung in Publikationen.

Beispiele für $q := (x + y)^3$:

```
      3        2            2     3
Q := X  + 3*X *Y + 3*X*Y  + Y

Q := X**3 + 3*X**2*Y + 3*X*Y**2 + Y**3$

      Q=X**3+3.*X**2*Y+3.*X*Y**2+Y**3

\begin{displaymath}
q=x^{3}+3 x^{2} y+3 x y^{2}+y^{3}
\end{displaymath}
```

Ein- und Ausgabe können auch in Files umgelenkt werden.

7 Offenes System

Ein Merkmal, das REDUCE von den meisten am Markt erhältlichen Systemen unterscheidet, ist seine durch eine langjährige Tradition untermauerte Offenheit:

- REDUCE ist in einer Sprache RLISP geschrieben, welche die Funktionalität von LISP in einer anwenderfreundlichen Syntax anbietet.
- REDUCE wird traditionell mit allen Quellen (auch für den Umsetzer RLISP nach LISP) ausgeliefert.
- REDUCE ist offen gegenüber anwenderseitigen Erweiterungen, die entweder auf dem algebraischen Niveau anwendernah oder auf dem symbolischen (=LISP) Niveau systemnah vorgenommen werden können.
- Da REDUCE von LISP automatisch die Fähigkeiten zum dynamischen Laden von Programmen, zum inkrementellen Kompilieren und zum Redefinieren von Programmen erbt, ist sogar der Kern von REDUCE gegenüber lokalen Änderungen offen.

Die Offenheit von REDUCE hat dazu geführt, daß in den letzten Jahren eine erhebliche Anzahl von Moduln in den verschiedensten Anwendungsbereiche von Anwendern entwickelt wurden, die den Einsatzbereich von REDUCE erheblich erweitern. Diese Pakete sind in einer Bibliothek über Netzzugang abrufbar.

Literatur

[1] J.H. Davenport, Y. Siret, E. Tournier: Computer Algebra, second printing, Academic Press, London, 1989

[2] Anthony C. Hearn, REDUCE User's Manual, Version 3.4, The Rand Corporation, Santa Monica (CA), 1991

[3] Gerhard Rayna, REDUCE, Software for Algebraic Computation, Springer, New York, 1987

[4] REDUCE Network Library, Bibliography, reduce-library@rand.org, wird laufend ergänzt

Research and Development in Maple.

Michael Monagan, ETH-Zürich

Maple is a system for doing mathematical computations. It can do both symbolic and numerical calculations and also contains facilities for 2 and 3-dimensional graphical output. Maple itself is a full programming language. Most of the mathematical and graphical facilties are written in this programming language. Maple also includes tools for producing Mathematical output for LaTeXand *troff*, and translating mathematical formulae into Fortran or C code with optimization.

In this article we mention some of the current research and development being done in Maple, in particular, some of the new Mathematical facilities that have been added to the latest version of Maple, Maple V, and also to the current development version, Maple 5.1. Also, Maple V contains completely new user interfaces with 3 dimensional colour graphics for the PC under MS/DOS, the X and Sunview windowing systems, the Macintosh and Amiga.

The Maple Project

Maple began as a university research project in November 1980 at the university of Waterloo, in Canada. There were two main design objectives. The first was to design and implement a programming language for implementing mathematical algorithms which was powerful and efficient enough so that almost all of the mathematical algorithms could be written in it, rather than in a systems implementation language like C or Lisp. The second was to design the system so that the amount of memory it required was kept small so that more storage would be available for calculations. It is well known that symbolic computation systems are "memory hogs" and excessive memory demands cause problems when several users try to run such systems simultaneously on a time shared machine.

Thus a lot of the initial design effort went into designing and improving the programming language so that it had the right programming tools, data structures, and mathematical primitives so that programs for indefinite integration, polynomial factorization, etc. could be written in it, and would execute reasonably efficiently. Choices were made at various levels so as to minimize memory requirements e.g. a small system kernel, compact data structures, and space efficient algorithms. Maple V requires typically under 500 kilobytes to start up compared with say Mathematica and Macsyma which require over 5 megabytes. It has the following 3 components

1. The user interface *Iris* contains the parser and routines for 2-dimensional output of formulae, file and graphical output. The teletype version of *Iris* for ASCII terminals is about 20,000 lines of C code.

2. The Maple kernel, which supports the programming language, data structures, and storage management, contains routines for doing basic arithmetic e.g. arbitrary precision integer arithmetic, univariate polynomial arithmetic over $\mathbf{Z}$ mod n. It is 20,000 lines of C code.

3. The Maple library, which contains over 90% of the mathematical facilties, including integration, polynomial factorization etc., is about 130,000 lines of Maple code. It contains over 2000 functions in number theory, group theory, polynomial algebra, linear algebra, calculus, statistics, etc.

There is also a growing number of functions and packages in the Maple *share* library which have been contributed by users from around the world. The share library also contains bug fixes to older versions, copies of the Maple technical newsletters, other software tools, etc. It is updated bi-monthly as new applications codes are contributed and is now distributed electronically by an electronic mail server and anonymous `ftp`. In the following we show examples of facilities in Maple that reflect some of Maple's capabilities and current Mathematical research and development.

Indefinite Integration.

Maple V has an implementation of the Trager-Rothstein method for rational function integration. The integral of a rational function is expressed as a sum of log's over the roots of a polynomial. For example, consider

$$\int \frac{5\,x^2+3}{x^5+6\,x^3+9\,x+1}\,dx$$

The difficulty in this example is that the denominator does not factor, hence we cannot split the rational function into partial fractions. Maple returns the following nice compact answer

$$\sum_{\alpha=\mathrm{RootOf}(Z^5+3Z-1)} \alpha \ln(x+\alpha^2)$$

where the summation index α runs over the complex roots of Z^5+3Z-1.
Maple V has an implementation of the Risch integration procedure for the log and exponential cases. The Risch algorithm is a complete decision procedure for the elementary functions. That is, given an elementary function, the algorithm either outputs the integral, or, if it fails, it has proven that the integral cannot be expressed in terms of elementary functions. In Maple 5.1 this has been extended to handle algebraic functions e.g.

$$\int 2\frac{2x^4+1}{(x^5+x)\sqrt{x^4+1}}\,dx$$

$$= -\frac{\sqrt{x^4+1}}{x^4+1} - \ln\left(-\frac{1+\sqrt{x^4+1}}{x^2}\right)$$

Differential Equations.

Maple includes several algorithms for solving ordinary differential equations including a facility for obtaining a power series solution, e.g. for the differential equation

$$y''(x) - x sin(x) y(x) = 0$$

Maple V finds the series solution

$$y(0) + y'(0)x + \frac{1}{12}y(0)x^4 + \frac{1}{20}y'(0)x^5 + O(x^6)$$

A new facility in Maple 5.1 by Manual Bronstein is a Risch type algorithm for solving a linear ODE in a given field. That is, given a linear OED over a field K, the algorithm finds a solution in K if it exists, e.g. Maple 5.1 finds

$$\frac{C_1 x}{x^6+1}$$

as a solution to

$$(x^9+x^3)y'''(x) \;+\; 18x^8y''(x) \;-$$
$$90xy'(x) \;-\; 30(11x^6-3)y(x) \;=\; 0$$

where C_1 is an arbitrary constant and K in this example is $\mathbf{Q}[x]$.

Definite Integration.

Maple V includes a new method for computing classes of definite integrals based on evaluating the derivatives of special functions, e.g. for integrals of the form

$$\int_a^\infty t^n \exp(-ut^s)\ \ln(bt)^m \quad \begin{matrix}\sin(ct^r)\ dt\\ \text{or}\quad \cos\end{matrix}$$

where $r \in \{0, s, 2s, s/2\}$, n is a rational, and other restrictions on the constants apply. In general these integrals can be expressed in terms of the incomplete Gamma function, the Riemann Zeta function, the Meijer G function, and other special functions e.g.

$$\int_0^\infty t\ln(t)e^{-\sqrt{t}}\,dt \;=\; 44 - 24\,\gamma$$

where γ is Eulers constant. This method is being extended in Maple 5.1 to allow error functions, and rational functions in t, and other classes of definite integrals.

Numerical Integration.

The numerical integration routine includes two capabilities not found in numerical systems which exploit the symbolic capabilities of Maple. The basic numerical algorithm used is Clenshaw-Curtis quadrature which computes an integral to d digits of precision such that the number of function evaluations done is "polynomial" in d. This means that it is possible to evaluate integrals to 50 digits of precision in a reasonable time. The other feature is that it automatically transforms integrands with singularities in them. This is done by exploiting Maple's series facility at the endpoints of the range of integration, e.g.

$$\int_0^\infty t\ln(t)e^{-\sqrt{t}}\,dt \;=\; 30.14682404236321134 5$$

Eigenvector Computation.

Maple V contains routines for computing with algebraic numbers. Algebraic numbers allow one to express eigenvectors[1] in a nice exact form even if the roots of the characteristic polynomial cannot be expressed as exact radicals). E.g. given the matrix

$$\begin{bmatrix} 1 & 3 & 0\\ 3 & 4 & 3\\ 0 & 3 & 0\end{bmatrix}$$

[1]eigenvectors can also be computed numerically

Maple returns the eigenvectors in the form

$$[1, \frac{\alpha}{3} - 1/3, -5/9 - \frac{5\,\alpha}{9} + \frac{\alpha^2}{9}]$$

where α is a root of the characteristic polynomial $\lambda^3 - 5\,\lambda^2 - 14\,\lambda + 9$. Since Maple 5.1 includes routines for computing with algebraic functions, symbolic eigenvectors can be obtained for matrices with polynomials and rational functions as entries.

Polynomial Factorization.

Maple V can factor univariate polynomials over the rationals and more generally, algebraic number fields, and finite fields. For example, the polynomial $x^8 + 8$, which is irreducible over $\mathbf{Q}$, factors over $\mathbf{Q}(\sqrt[4]{-8})$ as

$$\left(x^4 + \sqrt{-8}\right)\left(x^2 + \sqrt[4]{-8}\right)\left(x^2 - \sqrt[4]{-8}\right)$$

Maple 5.1 includes factorization of multivariate polynomials over algebraic number and algebraic function fields. For example the polynomial $x^4 + 2\sqrt{3}y^2 - 3 - y^4$ factors over the field $\mathbf{Q}(\sqrt{3})$ as

$$\left(x^2 - \sqrt{3} + y^2\right)\left(x^2 - y^2 + \sqrt{3}\right)$$

Asymptotic Series.

Maple V includes an implementation of a new model of "hierarchical" series in which various singularities about the point of expansion are handled e.g. for $e^{x^2}(1 - \mathrm{erf}(x))$ Maple obtains the asymptotic series

$$\frac{1}{\sqrt{\pi}x} - \frac{1}{2\sqrt{\pi}x^3} + \frac{3}{4\sqrt{\pi}x^5} + O(x^{-7})$$

Consider the recurrence equation

$$f(n+1) \;=\; sin(f(n))$$

which obviously converges to 0. How fast does it converge to 0? In particular, what is $S = \sum_{n=1}^{\infty} f(n)$? Although the recurrence equation has no "closed form" solution, the following asymptotic series for $f(n)$ is found by Maple

```
> r := f(n+1)=sin(f(n)):
> asympt(rsolve(r,f(n)),n);
```

$$\frac{\sqrt{3}}{\sqrt{n}} + \frac{\_C - \frac{3\sqrt{3}\ln(\sqrt{n})}{5}}{n^{3/2}} + O(n^{-2})$$

Thus S is approximately $\sum_{n=1}^{N} \frac{\sqrt{3}}{\sqrt{n}}$ for large N. Again, although this sum has no closed form, we determine that it diverges from the following asymptotic expansion

```
> asympt( sum(1/sqrt(n),n=1..N), N );
```

$$2\sqrt{N} + \zeta(1/2) + \frac{1}{2\sqrt{N}} - \frac{1}{24\,N^{3/2}} + O(N^{-7/2})$$

Algorithmic Differentiation.

The `D` operator in Maple 5.1 is capable of differentiating functions of n variables and simple procedures. For example

```
> D( x -> exp(x)*x^2 );
                    2
   x -> exp(x) x  + 2 exp(x) x
```

In Maple 5.1, we have extended this to procedures which contain assignment statements, if statements, and loops, i.e. algorithmic differentiation. Example, consider $f(x) = x\cos(x) - x^3 + 2\,x^2\cos(x)^2$ computed by the Maple procedure

```
> f := proc(x) local c,t;
>   c := cos(x);
>   t := x^2;
>   x*c-t*x+2*t*c^2;
> end:
```

The derivative of this function f' is given by

```
> D(f);
proc(x) local c,t,cx,tx;
  cx := -sin(x); c := cos(x);
  tx := 2*x; t := x^2;
  c+cx*x-tx*x-t+2*tx*c^2+4*t*c*cx
end
```

The advantage of differentiating programs instead of formulae is that programs are more general, and represent formulae in a more compact way. We are also looking at computing Gradient's and Jacobian's of functions given by procedures.

Share Library

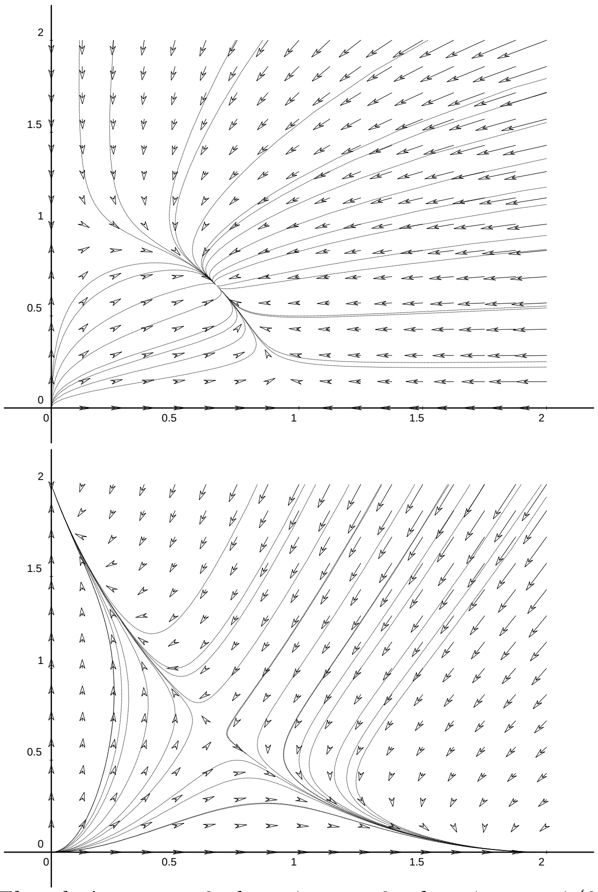
The share library contains functions and packages covering a variety of areas e.g. a package of routines for rational function approximation, an implementation of Ritt-Wu's characteristic sets method, a package of routines for computing with real algebraic numbers, etc. Also included is a routine to plot the phase space with flow lines for a second order autonomous differential equation by Daniel Schwalbe. Here is an application of this routineWe give an application of this routine.

The standard model for two populations x and y which compete for resources is

$$x' = ax - bxy - cx^2$$

$$y' = dx - eyx - fy^2$$

where a, b, c, d, e, f, are positive constants. a and d are the natural growth rates of the populations x and y respectively. b and e measure the effect the populations have on each other. c and f measure the effects of the size of the populations on themselves.



The choices $a = 2$, $b = 1$, $c = 2$, $d = 1$, $e = 1/2$, and $f = 1$ represent two species which coexist, i.e. both populations reach an equilibrium value. In the second example, $a = 1$, $b = 1$, $c = 1/2$, $d = 2$, $e = 2$, and $f = 1$ represent two species where only one survives. Which survives depends on the initial condition. Phase plots for the two examples are shown in figure 1. The plots generated in Maple by

```
x1 := (t,x,y)-> 2*x-x*y-2*x^2:
y1 := (t,x,y)-> y-1/2*x*y-y^2:
# list of initial values
N:=seq(seq([i/4,j/4],i=1..5),j=1..5):
phaseplot([x1,y1],0..2,0..2,{N});
x2 := (t,x,y)-> x-x*y-1/2*x^2:
y2 := (t,x,y)-> 2*y-2*x*y-y^2:
phaseplot([x2,y2],0..2,0..2,{N});
```

By linearizing the equations near an equilibrium point we can decide if the equilibrium point is stable or unstable. From the second plot, the equilibrium point $x = 3/2$, $y = 3/2$ (determined by solving the equations $x2(t, x, y) = 0$, and $y2(t, x, y) = 0$) appears to be unstable. We can verify this by using Maple to show that the matrix of partial derivatives at this point has one positive and one negative eigenvalue as follows

```
> linalg[jacobian]
>    ([x2(t,x,y),y2(t,x,y)],[x,y]);
```

$$\begin{bmatrix} 1 - y - x & -x \\ -2\,y & 2 - 2\,x - 2\,y \end{bmatrix}$$

```
> eigenvals(subs({x=2/3,y=2/3},"));
```

$$\frac{\sqrt{11}\sqrt{3}}{6} - 1/2,\ -1/2 - \frac{\sqrt{11}\sqrt{3}}{6}$$

Further Information

The Maple project also has an active news group and a regular newsletter. For further information please contact the author (`monagan@inf.etz.ch`) or Waterloo Maple Software at

160 Columbia St. West
Waterloo, Ontario, N2L-3L3, Canada
Phone: (519) 747-2373 FAX: (519) 747-5285
Email: wmsi@daisy.uwaterloo.ca

Mathematica

Roman Maeder, ETH–Zürich

Mathematica, ein neues Softwarepaket für mathematisches Rechnen, wird im Unterricht und in der Forschung und Entwicklung auf vielen Gebieten eingesetzt. Es hilft einem bei numerischen und symbolischen Rechnungen und kann Resultate graphisch darstellen. Als Beispiel für die Kombination von symbolischem und numerischem Rechnen schauen wir uns die graphische Darstellung von Minimalflächen an.

Ein Werkzeug für computerunterstützte Mathematik

Dank der Fortschritte in der Hardware, in den Betriebssystemen für Kleinrechner und Arbeitsstationen (einfache Benutzerführung) und in der Computer Algebra kann man heute hohe Anforderungen and Software für computerunterstützte Mathematik stellen. Dazu gehören:

- Es soll nicht mehr nötig sein, in umfangreichen Tabellenwerken Zahlen nachzuschlagen. Das Programm hat entweder *Algorithmen* eingebaut, um diese Zahlen zu berechnen, oder die entsprechenden Daten sind in *maschinenlesbarer* Form vorhanden.
- Es muss einfach sein, mathematisches Fachwissen (etwa aus einem Handbuch) in das Programm zu integrieren.
- Dasselbe Programm sollte auf einem Kleinrechner und auf einem leistungsfähigen Grossrechner laufen.
- Eine Programmiersprache erlaubt die Erweiterung des Systems in irgendeinem Wissensgebiet.

Eines der Ziele beim Entwurf von *Mathematica* war es, ein Werkzeug für die meisten Anwendungen zu entwickeln. *Mathematica* wurde ab 1987 von einer kleinen Gruppe unter der Leitung von *Prof. Stephen Wolfram* (Universität von Illinois, USA) entwickelt und ist seit Juni 1988 auf dem Markt.
Der wichtigste Bestandteil ist eine *flexible Programmiersprache*, die es erlaubt, das System zu erweitern. Die Sprache soll mathematisch orientiert sein eine einfache Definition mathematischer Regeln erlauben. Dies führt zu einer reichen Sprache mit vielen eingebauten Funktionen. Die Sprache ist dennoch einheitlich und hat eine innere Konsistenz, die das Erlernen erleichtert. Möglich wurde dies durch die ausschließliche Verwendung von *pattern matching* und Ersetzungsregeln, die allen anderen Programmierelementen, Kontrollstrukturen und Prozeduredefinitionen unterliegen. Dadurch ist es auch einfach, die Sprache interaktiv zu gebrauchen [Mae91].
Außer den genannten Anforderungen sollte ein modernes Softwarepaket sich nahtlos in die Umgebung des Computers, auf dem es läuft, einfügen. Es wird so zu einem Werkzeug, das auch von anderen Programmen benutzt werden kann.

Visualisierung mathematischer Sachverhalte

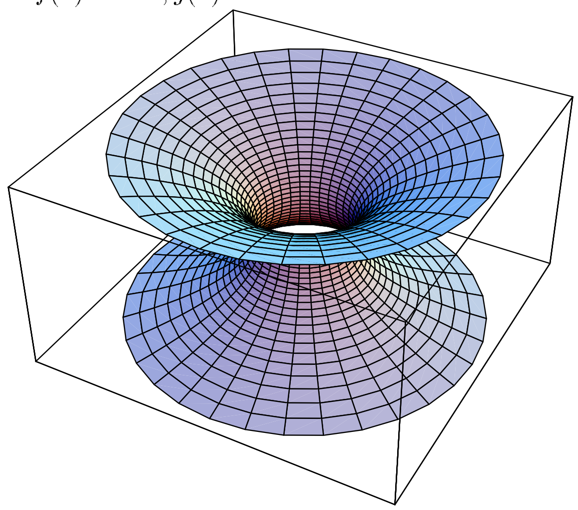
Die Mathematik fordert die Vorstellungskraft heraus. Sich auch die abstraktesten Konstruktionen vorstellen zu können, ist notwendig für eine erfolgreiche Beschäftigung mit der Mathematik. Diese Vorstellungskraft wird entschieden gefördert durch konkrete Modellbildungen, wo immer das möglich ist (also etwa in der Geometrie). Die Herstellung von Modellen oder genauen Zeichnungen ist jedoch schwierig. Visualisierungsprogramme auf Computern wurden schon seit einiger Zeit eingesetzt, um Modelle auf dem Computerbildschirm darzustellen, statt sie tatsächlich zu konstruieren. Problematisch ist jedoch das tiefe Beschreibungsniveau graphischer Sprachen, meist auf der Stufen von Koordinaten von Punkten, die dann zu Flächen zusammengesetzt werden.
Mathematica bietet nun die Möglichkeit, die Daten mit den üblichen mathematischen, also auch symbolischen, Methoden zu berechnen und dann gleich zu zeichnen. Auf geeigneten Computern können die fertigen Bilder interaktiv manipuliert werden (drehen, vergrößern, etc.). Am Beispiel der Minimalflächen wollen wir dies nun beschreiben. Wir beginnen mit der mathematischen Definition solcher Flächen und enden mit einem Trickfilm, der die mathematischen Transformationen dynamisch veranschaulicht.

Die Enneper-Weierstrass Parametrisierung von Minimalflächen

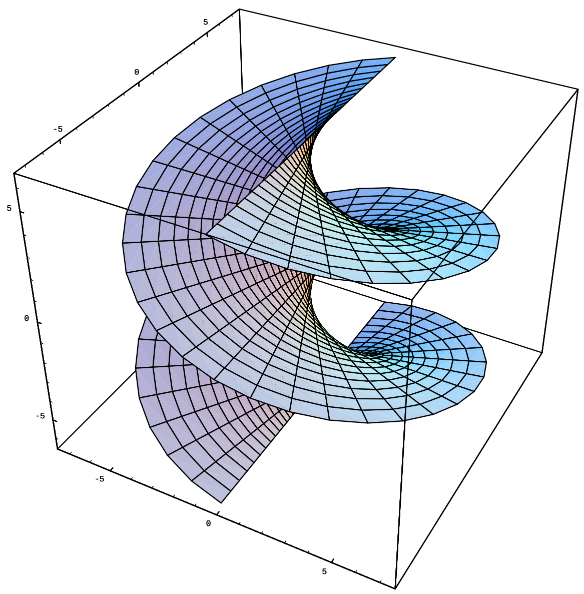
Zur Gewinnung einer Minimalfläche aus einer analytischen Funktion f und einer meromorphen Funktion g benützen wir die *Enneper-Weierstrass* Parametrisierung (siehe z. Bsp. [Hof87,Dic90]). Dabei erhalten wir die drei Koordinaten der Fläche in Abhängigkeit eines komplexen Parameters z als die Realteile von

$$\{\int f - fg^2 dz, \int i(f + fg^2)dz, \int 2fgdz\}$$

Wenn wir nun den Parameter z als $z = x + iy$ schreiben, erhalten wir die übliche Parameterdarstellung einer Fläche im Raum. Für eine graphische Darstellung werten wir die Gleichung für ein rechteckiges Punktegitter aus und verbinden die erhaltenen Punkte mittels Vierecken zu einer Fläche.
Mit $f(z) = e^{-z}, g(z) = e^z$ erhalten wir ein Katenoid.



Statt des Realteiles können wir auch den Imaginärteil nehmen. Dann erhalten wir die Wendelfläche. Dies ist aber äquivalent zum Realteil mit $f(z) = ie^{-z}$, statt e^{-z}.
Einen stetigen Übergang zwischen diesen 2 Flächen erhalten wir durch Multiplikation von $f(z)$ mit einer komplexen Zahl vom Betrage 1, deren Argument sich zwischen 0 und $\pi/2$ ändert, also mit $e^{i\phi}, 0 \leq \phi \leq \pi/2$. Schließlich können wir ϕ den ganzen Wertebereich von 0 bis 2π durchlaufen lassen. Das Programm *Mathematica* gibt einem nun die Möglichkeit, direkt aus der Integralformel die Punkte der Fläche zu berechnen und diese graphisch darzustellen. Wir können auch für viele verschiedene Werte von ϕ je eine Darstellung erzeugen und diese dann—wie einen Trickfilm— hintereinander darstellen. Es entsteht so der Eindruck einer kontinuierlichen Bewegung.



Analytische Fortsetzungen

Wenn die Parametrisierung zum Beispiel Logarithmen enthält, dann gibt es Probleme mit einer rein numerischen Auswertung der Integralformel. Schauen wir uns an, was passiert mit der Wahl $f(z) = z, g(z) = \frac{1}{z}$ wiederum in polaren Koordinaten. Die einzelnen Rechenschritte, die wir im vorherigen Beispiel einzeln interaktiv durchgeführt haben, werden am besten in einer kleinen Prozedur zusammengefaßt, mit dem Namen `Parameterize[f, g, z]`. Damit sieht unsere Rechnung nun so aus:

```
In[2]:= Parameterize[z, 1/z, z]

          2
         z               I   2
Out[2]= {-- - Log[z], - z  + I Log[z], 2 z}
         2               2
In[3]:= % /. z -> r Exp[I phi]

          2 I phi  2
         E        r          I phi
Out[3]= {----------- - Log[E      r],
             2

   I   2 I phi  2          I phi         I phi
  - E          r  + I Log[E      r], 2 E      r}
  2
```

Der Ausdruck `Log[E^(I phi) r]` wird bei numerischer Auswertung immer den Hauptast des Logarithmus wählen, mit $-\pi < \Im(\log z) \leq \pi$.
Der Wert hat deshalb ein Sprungstelle, wenn der Be-

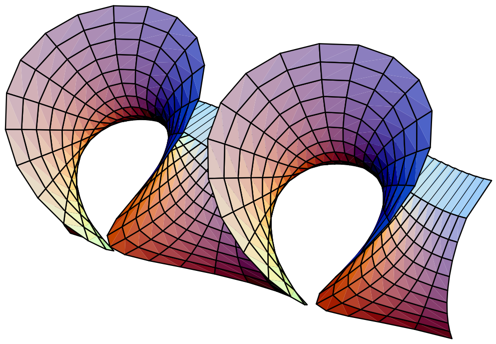
reich für ϕ ungerade Vielfache von π enthält.
Symbolisch ist es naheliegend, `Log[E^(I phi) r]` zu vereinfachen zu `I phi + Log[r]`. Der Imaginärteil ist nun gleich ϕ, ohne irgendwelche Unstetigkeitsstellen. (Es wäre allerdings falsch, die Vereinfachung $\log e^z \to z$ immer automatisch durchzuführen.)
In *Mathematica* werden solche Vereinfachungsregeln als Termersetzungsregeln angegeben. Die Notation `x_` bezeichnet ein Muster, das durch einen beliebigen Ausdruck ersetzt werden kann und das auf der rechten Seite der Regel mit `x` bezeichnet wird. Dies ist vergleichbar mit der Syntax von PROLOG (`X` für `x_`). Die linke Seite von Regeln kann allerdings in *Mathematica* beliebig kompliziert sein.

```
In[4]:= Expand[ % //.
          {Log[a_ b_] :> Log[a] + Log[b],
           Log[E^x_] :> x} ]

Out[4]=
             2 I phi  2
            E        r
 {-I phi + ----------- - Log[r],
                2

          I  2 I phi  2
  -phi + - E         r   + I Log[r],
          2

     I phi
  2 E      r}
```



Unterricht mit *Mathematica*

Mathematica wird bereits im Hochschulunterricht eingesetzt. Es dient natürlich einmal direkt dem Mathematikunterricht. Ziel des Unterrichtes ist es aber immer mehr, den Schülern und Studenten Grundkenntnisse im Lösen mathematischer Probleme mit Computerhilfe zu vermitteln [EM85,Mae87]. Diese Kenntnisse können dann im Unterricht in anderen Fachrichtungen vorausgesetzt werden, so wie etwa heute angenommen wird, daß alle Schüler mit einem Taschenrechner umgehen können.
Der Unterricht wird erleichtert durch die Benutzerschnittstelle, die *Mathematica* anbietet. Damit ist es möglich, Text, Graphik und *Mathematica* Eingabe und Ausgabe in einem hierarchisch strukturierten Dokument zu vereinigen. Lektionen lassen sich damit vorbereiten. Alle Beispiele in einem solchen „Notebook" sind lebendig, die Studenten können damit experimentieren.
Besonders zu erwähnen ist in diesem Zusammenhang die Arbeit von Porta und Uhl [BPU90] im Rahmen der Anstrengungen, den Analysis-Grundunterricht in den USA zu reformieren. Durch den Uebergang vom drillmäßigen Unterricht, wie er heute praktiziert wird, zu einem Kurs, in dem das *Experiment* die Hauptrolle spielt, werden die Studenten stark motiviert. Mit *Mathematica* lassen sich auch viel kompliziertere Beispiele rechnen, als von Hand, was die Rolle der Analysis als *Grundlage* der Ingenieurwissenschaften und der Physik wieder verständlich macht.

[BPU90] Don Brown, Horacio Porta, and Jerry Uhl. Calculus&Mathematica: Courseware for the nineties. *The Mathematica Journal*, 1(1), 1990.

[Dic90] Stewart Dickson. Minimal surfaces. *The Mathematica Journal*, 1(1), 1990.

[EM85] E. Engeler and R. Maeder. Scientific computation: The integration of symbolic, numeric and graphic computation. In B. Buchberger, editor, *Proceedings of Eurocal '85 Vol. I*, volume 203 of *SLNCS*, New York, 1985. Springer-Verlag.

[Hof87] David Hoffman. The computer-aided discovery of new embedded minimal surfaces. *The Mathematical Intelligencer*, 9(3), 1987.

[Mae87] Roman E. Maeder. A collection of projects for the mathematical laboratory. *SIGSAM Bulletin*, 21(3), August 1987.

[Mae91] Roman E. Maeder. *Programming in Mathematica*. Addison-Wesley, second edition, 1991.

[Wol91] Stephen Wolfram. *Mathematica: A System for Doing Mathematics by Computer*. Addison-Wesley, second edition, 1991.

Axiom, ein Computeralgebrasystem mit abstrakten Datentypen

Johannes Grabmeier
Wissenschaftliches Zentrum Heidelberg, IBM Deutschland

1 Überblick

Axiom[1], früher informell mit *Scratchpad* bezeichnet, ist ein Computeralgebrasystem der **dritten Generation** mit einem Designkonzept, das auf **abstrakten, parametrisierten Datentypen**, **objektorientierter** und **polymorpher Programmierung** und einer konsequenter Anwendung **algebraischer Algorithmen** aufbaut. Axiom basiert auf LISP und läuft auf der **Workstation IBM RISC System/6000**[2] sowie auf anderen IBM[3] Rechnern mit dem Betriebssystem AIX[4].
Axiom stellt sowohl einen **Interpretierer** als auch einen **Kompilierer** zur Verfügung. Während der Interpretierer in der Lage ist, Datentypen anzupassen, passende Funktionen auszuwählen und Übergänge selbst durchzuführen, werden vom Kompilierer sogenannte *strongly typed* Programme erwartet. Ein einmal kompiliertes Programm steht dann dem Benutzer im Interpretiermodus in gleicher Weise wie eine Systemkomponente zur Verfügung.
Axiom verfügt über eine sehr reichhaltige **Mathematikbibliothek** mit mehr als 600 Datentypen und mehr als 1700 Funktionsnamen, die mehrfach überladen sein können.
Ein *window*-gestütztes **Multitextsystem** besteht aus **Tutorial**, Beispielrechnungen, Erläuterung der Systemkommandos und einem **Informationssystem** zum Suchen von Funktionen, Querbezügen, Erläuterungen und Beispielen zu den Datentypen. Der **Quellcode** der mathematischen Algorithmen und Datentypen ist dem Benutzer **zugänglich**.
Das System besitzt Schnittstellen zu **FORTRAN**, **SCRIPT** und TeX. **Graphik** ist über X-Windows realisiert und gestattet PostScript[5]-Ausgabe.
Das System wurde und wird von der Computeralgebragruppe unter Leitung von R. Jenks im Forschungszentrum der IBM in Yorktown Heights (USA) mit Beiträgen des wissenschaftlichen Zentrums der IBM in Heidelberg entwickelt.
Einen gute Einführung in das System gibt auch [8].

2 Interpretierer

Der Interpretierer versucht mit automatischen Typanpassungen ein typfreie Sicht der Datentypwelt von Axiom zu ermöglichen. Diese Problemstellung und ihre algorithmische Lösung wird ausführlich in [3] diskutiert. Die *workspace*-Organisation gestattet die Verwaltung von Benutzer-definierten Variablen, Protokollierung der Sitzung, Bezugnahme auf vorhergehende Rechnungen. Weiter wird eine Schnittstelle zum Betriebssytem zur Verfügung gestellt. Zusätzlich können Funktionen kompiliert werden und Dateien eingelesen werden, deren Zeilen schrittweise interpretiert werden.

3 Datentypen

In Axiom gibt es abstrakte Datentypen, (*categories*) — in gewissem Sinne mit mathematischen Kategorien vergleichbar — konkrete Datentypen oder Rechenstrukturen (*domains*) und Funktionspakete (*packages*).
Diese Datentypkonstruktoren gestatten es dem Benutzer, beliebig viele verschiedene Datentypen aufzubauen, auch kann er selbst solche Rechenstrukturen implementieren. Zum Beispiel kann man, ausgehend vom Basisdatentyp *Integer*, Matrizen konstruieren, deren Einträge Polynome mit ganzzahligen Koeffizienten sind. Polymorphe Programmierung bedeutet nun, daß sofort die übliche Arithmetik zur Verfügung steht, da etwa die Funktion zur Multiplikation von

[1] Axiom is a trademark of The Numerical Algorithms Group Ltd.
[2] IBM RISC System/6000 is a trademark of International Business Machines Corp.
[3] IBM is a registered trademark of International Business Machines Corp.
[4] AIX is a trademark of International Business Machines Corp.
[5] PostScript is a trademark of Adobe Systems, Inc.

zwei Matrizen nur die Eigenschaft benutzt, daß die Matrixeinträge Elemente einer Rechenstruktur sind, die zur Kategorie der Ringe gehört.
Darauf aufbauend wäre etwa Adjunktion von i (*Complex*) sowie Verwendung dieser Elemente als Koeffizienten von Polynomen in mehreren Unbestimmten möglich.
Die Funktionen in den Funktionspaketen (*packages*) ergänzen die Datentypen zu einer Mathematikbibliothek.

4 Mathematikbibliothek

Einen Überblick über die Mathematikbibliothek mag die folgende Auswahl geben:

- ganze Zahlen beliebiger Länge
- rationale Arithmetik und allgemeiner Quotientenkörper
- Gleitkommazahlen mit beliebiger Genauigkeit
- Spezielle Funktionen mit beliebiger Genauigkeit
- Gaußsche Zahlen
- faktorisierte Form ganzer Zahlen
- Kettenbrüche
- Partialbrüche
- rationale Zahlen mit verschiedenen Basen
- Radikalerweiterungen rationaler Zahlen und algebraische Zahlen
- ganze Zahlen modulo m
- endliche Körper, realisiert als Polynomrestklassenring, unter Ausnutzung der zyklischen Struktur und Normalbasendarstellung, siehe [6]
- Kardinalzahlen
- Quaternionen, Oktaven
- Cliffordalgebren
- Nichtassoziative Algebren, Lie- und Jordanalgebren, siehe [7]
- Polynome in einer und in mehreren Unbestimmten mit Koeffizienten in beliebigen Ringen
- Lösen von polynomialen Gleichungen vom Grad kleiner 5 durch Radikale
- Einsetzen von Zahlen und Polynomen in Polynome
- Faktorisierung von Polynomen über endlichen Körpern, ganzen Zahlen und algebraischen Erweiterungen der rationalen Zahlen, ein kategorieller Zugang dazu ist in [2] beschrieben
- Interpolation
- Lösen von Systemen polynomialer Gleichungen: Gröbnerbasis und Buchbergeralgorithmus
- Rechnen mit Polynomidealen, Dimension, Primärzerlegung
- Matrizen über beliebigen Ringen
- Determinanten und Permanenten
- Lösen linearer Gleichungssysteme: Gaußscher Algorithmus
- Charakteristische Polynome
- Eigenwerte und Eigenvektoren
- Orthonormale Basen
- Differentiation
- allgemeiner Operatorenkalkül
- indefinite Integration elementarer Funktionen
- definite Integration rationaler Funktionen
- Manipulation von Funktionen: Patternmatching, Termersetzung Anwendung von Regelsystemen
- Lösung von gewöhnlichen Differentialgleichungen, durch Potenzreihen für lineare Gleichungen können teilweise exakte Lösungen gegeben werden
- Grenzwerte, linksseitig, rechtsseitig, im Unendlichen
- Folgen und Potenzreihen
- Taylor, Laurent und Puiseuxentwicklungen
- Partitionen und kombinatorische Listen
- Zahlentheorie: Primzahlen, Eulersche ϕ-Funktion, Legendresymbol
- indefinite Summation (Gosper)
- Permutationen und Permutationsgruppen
- Darstellungstheorie von Gruppen, siehe [4]

5 Graphik

Folgende Graphikmöglichkeiten sind vorhanden.

- Zeichnen von Funktionen im zwei- und im dreidimensionalen Raum
- parametrisierte Kurven
- impliziert definierte Kurven

Die von Axiom berechneten Daten werden dabei an einen Prozess geschickt, der ein neues Fenster mit der Graphik öffnet. Dann kann die Graphik menugesteuert mit der Maus manipuliert werden:

- Vergrößern, Verkleinern und Verschieben
- Drehen
- Änderung des *Lichteinfalls*
- Änderung der Perspektive
- Ausschnitte
- Änderung der Farbskala
- Überlagerung von verschiedenen Graphiken in einem Fenster
- Verschiedene Darstellungen der Flächen

Hervorzuheben ist hier das Zeichnen von implizit definierten Funktionen, da dabei auch die algebraischen Fähigkeiten (Gröbnerbasen) zur Bestimmung der Singularitäten benutzt werden.

Literatur

[1] Bronstein, M., Davenport, J.H., Trager, B.M., Symbolic Integration is Algorithmic! Tutorial Notes, Computer and Mathematics, Boston, (1989)

[2] Davenport, J.H., Gianni, P., Trager, B.M., Scratchpad's view of algebra II: A categorial view of factorization, Proc. ISSAC'91, Bonn

[3] Fortenbacher, A., An Efficient Rewrite-Rule-Based Coercion Algorithm for Computer Algebra, submitted to J. Symb. Comp., (1991)

[4] Gollan, H., Grabmeier, J. Algorithms in Representation Theory and their Realization in the Computer Algebra System Scratchpad, Bayreuter Mathem. Schriften, ISSN 0172-1062, Heft 33, 1-23, 1990.

[5] Grabmeier, J., Huber, K., Krieger, U. Das Computeralgebra Scratchpad bei kryptologischen und verkehrstheoretischen Untersuchungen des Forschungsinstituts der Deutschen Bundespost TELEKOM, Technischer Report, IBM Wissenschaftliches Zentrum Heidelberg, (1991)

[6] Grabmeier, J., Scheerhorn, A., Finite Fields in Axiom, in preparation, (1991)

[7] Grabmeier, J., Wisbauer, R., Computations in Algebras of Finite Rank, in preparation, (1991)

[8] Jenks, R. D., Sutor, R. S., Watt, S. M. Scratchpad II: An Abstract Datatype System for Mathematical Computation, in R. Janßen (Ed.), Trends in Computer Algebra, International Symposium Bad Neuenahr, May 1987 Proceedings Lecture Notes in Computer Science, **296**, Berlin 1987, 12–37

Theorist - a User Interface for Symbolic Algebra

Allan Bonadio

1 Introduction

Theorist is a symbolic algebra program that emphasizes a new graphical user interface. The user interface is graphical and interactive and has been well received. In particular, many people find they can use it after merely watching someone else use it, as opposed to the usual learning curves typically associated with such systems.

2 Equation Entry

Equations can be entered into Theorist either through the keyboard or via a palette, which has pictures of each variable, constant, function and operator available.
Theorist requires that all names of variables and functions have appropriate declarations. As new names are encountered in a notebook, Theorist will query the user as to the nature of the name and will allow the user to choose whether the name is declared to be a constant, a variable, a linear operator, or a function. The option is also given to not declare the name, in case it was misspelled. This system avoids the problems inherent in systems that don't use declarations, while avoiding the inconvenience of maintaining declarations by hand.

3 Computer Algebra

The fundamental principle behind interaction in Theorist is a system called the "Propositional Schema". Actually, "Propositional Schema" is a new name for an old idea, used for centuries, where mathematical knowledge is described by equations or inequalities (propositions) and new propositions are derived based upon assumed propositions. It is obvious and intuitive to those who have been educated in the twentieth century. Theorist is usually easier to understand than the typical symbolic algebra program where the user enters a question, in the form of a function enclosing argument expressions that are to be manipulated, and the system displays an answer. In practice, some users who are very familiar with certain command-oriented systems have some trouble adapting to Theorist; this problem has not been encountered with new users.
In Theorist, there are no questions and answers. Instead, there are assumptions and conclusions. Assumptions are equations that the user types in; they have a square icon on the left. Conclusions are equations that Theorist generates when the user does a manipulation; they have a triangluar icon instead. Both are somewhat equivalent in that they can be used in subsequent manipulations. Also, graphs can be drawn using either of them.
For example, the user can type in this equation:

$$y = \sqrt{\tan(x)}$$

The user can then select the x and, holding down the Command (Clover) modifier key, drag the x to the far left of the equation. When the appropriate highlighting displays, the user releases the mouse button, and a conclusion is generated:

$$x = \arctan\left(y^2\right)$$

Another kind of manipulation that can be done graphically is substitution. For example, the user can select the resulting equation from the last example, and can hold down the Command modifier key again and drag it on top of an expression such as this:

$$\int \sqrt{\tan(x)}\, dx$$

to yield this result:

$$2\int \frac{y^2}{y^4+1} dy$$

(Note in this example that the differential operator d actually differentiated $\arctan\left(y^2\right)$ in the process of simplification.) Any equation can be used in this way and any expression that matches the left hand side will be transformed into the right hand side.
Theorist also supports wildcard variables with which the user can make expression patterns that can match a wide variety of expression targets.

4 Graphics

In Theorist, the user can select any equation and request that a graph be created for it. Once created, the graph is an interactive object. The user can scroll or rotate the graph for a different viewpoint. The user can also zoom in or out, or increase or decrease the resolution (number of panels or line segments) in the objects in the graph, by way of buttons around the periphery of the graph.

The simplest transformation of a graph is by clicking and dragging on the face of the graph. The mouse pointer turns into a spread hand, ready to scroll the image. In the case of a two dimensional graph, the graph bounds are scrolled sideways as though it were a large sheet of paper under the user's hand. The new graph image is drawn such that the axes remain fixed relative to the edge of the graph, but the domain and range have been changed.

In the case of a three dimensional graph, when the user clicks down, a cube is displayed showing the bounds of the graphic space of primary attention. Dragging motions from the user result in rotations of this cube. When the mouse button is released, the three dimensional objects are redrawn as seen in the new orientation.

There are many different kinds of graphs available by menu command. In addition, the graphs themselves are programmable in that the user can delete or add more parametrically defined graphical objects to those that make up the graph as created by menu command.

For instance, if the user enters the equation $z = \sin(x)\sin(y)$, and then chooses "Color 3D" from the Graph menu, a three dimensional graph will be created containing a surface object, plus some axis and grid objects. The surface object draws a parametrically defined surface in the three dimensional space. Each axis object draws one axis line; there are three by default in this kind of graph. Each grid object draws a set of parallel grid lines (they are parallel in the space of their parameters but needn't be parallel graphically). The axis and grid objects are merely annotation but work similarly to the way the surface object works. The user can delete any of these, or can add more of any kind, and can change their expressions and other properties by simply clicking and typing.

5 Calculations

With Theorist, the user can do calculations by using the Calculate manipulation. As with any other manipulation, the user can calculate an expression and the result is shown below in a conclusion equation.

In addition, calculations are performed as part of drawing a graph. Drawing a graph merely involves repeatedly calculating expressions that are inside the graph. Those calculations reference variables that are defined by equations outside the graph, such as the original equation the graph was requested to use.

Any equation of a specific form can become a Working Statement, which defines the value of a variable, the definition of a function, or other properties. A Working Statement has a dot in its icon to indicate its special status. The user can choose to turn on or off Working Statement status. In addition, the program itself will turn it on or off in an effort to resolve undefined or multiply defined variables.

For instance, when the user creates a graph of $y = \sin(x)$, the equation becomes a Working Statement that defines y. Theorist then evaluates the vector (x, y) repeatedly in order to plot the points of the line. When it needs to evaluate y, it uses the Working Statement.

Users can create notebooks with any number of Working Statements that are defined in terms of each other, effectively forming numerical pipelines that are evaluated during graph drawing. Users who change any Working Statement will trigger the redrawing of any graphs that depend upon it. Finite recursion is allowed; a conditional function is supplied to prevent such recursion from becoming infinite.

6 Theories

A theory is a set of propositions that are segregated from their surrounding propositions both graphically (with an enclosing box) and mathematically.

A theory represents a limited set of possibilities. Propositions inside of a theory are only assumed to be true within that theory, although propositions outside the theory are valid within the theory.

The integrity of theories is enforced throughout Theorist. Propositions in different theories can be mixed only if one theory is a direct descendent of the other. For instance, a substitution mixing propositions cannot be done between propositions in sibling theories, although they can be done between propositions in a theory and propositions in the theory's grandparent theory.

Theories can be made manually by the user or they can be made automatically as the result of some operation. For instance, Theorist has a numerical root finder that works in conjunction with two dimensional graphs. A user might enter the equation

$y = x^3 - 5x + 1$ and then draw a graph of it and find one or more roots. Each root is placed as an equation in its own theory, for instance, $x = 2.1284$. This equation can subsequently be substituted into other equations to learn more about the mathematical system under investigation in the vicinity of that point. All resulting propositions, however, will be placed in that theory because they are not valid unless $x = 2.1284$ is also valid. Sibling theories may have similar investigations underway for other roots such as $x = -2.33006$ or $x = 0.20163$. It is impossible, however, for the user to mix these mutually exclusive situations without manually copying and pasting expressions.
Theories may be nested to any required depth. At the top level, the main theory encloses everything in the notebook.

7 Notebooks and Outlining

Theorist data is stored in "notebooks". Each notebook holds equations and graphs. In addition, notebooks can contain textual and picture comments provided by the user that are ignored by all mathematical processes. All such data is displayed in a window on the screen.
Theorist notebooks are organized with a topical outliner system. The user can open a topic to see its subtopics by double clicking on its icon (on the left). Many facilities in Theorist work with the outliner system to keep a notebook orgainized. For instance, all name declarations are hidden under the first proposition of a notebook, which is usually a comment. Another example is that when conclusions are generated from other equations, the conclusions are indented underneath the equation they were derived from, so that the dependency tree can be easily visualized. Whole trains of thought can be hidden with a single double-click or eliminated with a single deletion keystroke.

8 Limitations

Theorist runs on any Macintosh with at least one megabyte of memory. Implementations for other graphical user interfaces are planned for the future but there are no such versions at present.
For the most part, Theorist's symbolic capabilities are greater than PowerMath and Milo, roughly comparable to Derive, but inferior to Maple, Mathematica, Macsyma and Scratchpad II. For instance, it can invert matrices symbolically but it cannot find eigenvalues either symbolically or numerically (without the user going through the steps for finding roots of the characteristic polynomial). As is always the case, it is hard to compare systems and there are features in all of these programs that are unavailable on "more powerful" systems. In particular, many interactive features of Theorist increase the power of the user in unexpected ways. This article is merely a brief overview.
Theorist has no procedural programmability. Theorist can be programmed, though, in certain nonprocedural ways:

- algebraic rules can be added to its manipulation facilities
- graphical objects can be added to its graphs
- variables and functions can be defined for numerical calculations

9 Availability

Theorist is a commercial program published in the United States by:
Prescience Corporation
939 Howard St
San Francisco, CA 94103 USA
prescien@well.sf.ca.us
Phone: (415) 543-2252
Fax: (415) 882-0530
The retail price is US$ 400. A low-cost demonstration version is available, as is an advertising demonstration video and other information. The program is also available through most American mail-order retailers at a significant discount.
In German and French speaking countries, contact:
ALSYD
43, chemin du vieux
chêne
F-38240 Meylan
France
Phone: 76 41 85 05
Fax: 76 41 83 96

MuPAD

Multi **P**rocessing **A**lgebra **D**ataTool

Andreas Kemper, Oliver Kluge, Karsten Morisse,
Holger Naundorf, Waldemar Wiwianka, Benno Fuchssteiner
Universität – Gesamthochschule Paderborn
Fachbereich Mathematik

Der vorliegende Bericht gibt einen Überblick über das MuPAD Projekt. MuPAD steht für *MultiProcessing Algebra Datatool.* MuPAD ist ein neues Computer Algebra System, und wie es der Name schon ausdrückt, handelt es sich um ein System für Parallelrechner. Hinsichtlich des Anwendungsbereiches von MuPAD und dem Einsatz der Parallelverarbeitung in MuPAD machen wir keine Einschränkungen. MuPAD ist ein general – purpose Computer Algebra System, und die Parallelverarbeitung wird in alle Teilen des Systems einbezogen.

1 Designziele

Ein Hauptaugenmerk bei der Entwicklung von MuPAD liegt darin, daß MuPAD den Anwender in die Lage versetzen soll, auch mit sehr großen Datenmengen hantieren zu können. Daten, die auf Grund ihrer Größe mit anderen Computer Algebra Systemen nicht behandelt werden können, sollen durch MuPAD mit Hilfe eines Parallelrechners bearbeitet werden. Der zur Verwirklichung dieses Zieles eingesetzte Multiprozessor bleibt dem Benutzer jedoch verborgen. Weitere Designziele sind:

- Systemoffenheit: der Benutzer hat die Möglichkeit das System durch selbstdefinierte Datentypen zu erweitern
- strikte Modularisierung durch Trennung in Kern, Speicherverwaltung und Bibliothek
- keine Unterscheidung zwischen Programmkonstrukten und algebraischen Daten

2 Funktionalität

2.1 Datentypen

Neben den Zahlentypen *ganze Zahl, rationale Zahl, Gleitkommazahl*, die jeweils mit beliebiger Genauigkeit bzw. Länge zur Verfügung stehen, gibt es als weiteren atomaren Datentyp die *Zeichenkette.* Darüberhinaus steht eine Vielzahl zusammengesetzter Datentypen, genannt Kategorien, zur Verfügung. Dazu zählen *Tabelle, Liste, Menge, Feld, Prozedur, Ausdruck, Ausdruckssequenz, Anweisung, Anweisungssequenz.* Für die meisten Typen ist eine explizite Typdeklaration nicht nötig. Diese erfolgt implizit. Neben diesen, vom System vorgegebenen Typen, hat der Benutzer die Möglichkeit, eigene Kategorien zu definieren. Zu diesem Zweck stehen ihm Funktionen zur Verfügung, mit denen er die algebraische Struktur des neuen Typs beschreiben kann. Durch die Behandlung sämtlicher Programmkonstrukte als Datentypen ist die Generierung und Manipulation von Programmen möglich.

2.2 Funktionen

Parameter werden an Funktionen durch *Call by value* übergeben. Weiterhin besteht die Möglichkeit, Funktionen mit Attributen zu versehen. Zur Verfügung stehen *hold*, *remember* und *trace.* Bekanntlich ist der *remember* – Mechanismus eine sehr effektive Möglichkeit, die Geschwindigkeit rekursiver Funktionen zu steigern. Realisiert wird *remember* durch einen assoziativen Speicher.

2.3 Programmiersprache

Die Programmiersprache besitzt neben funktionalen Konstrukten auch Elemente objekt – orientierter Sprachen. Als Kontrollstrukturen stehen die üblichen Verzweigungs – und Schleifenkonstrukte zur Verfügung. Die Anweisungen können sowohl interaktiv, als auch im Rahmen von Prozeduren eingesetzt werden.

3 MuPAD, ein paralleles System

Die Zielarchitektur für MuPAD ist ein Multiprozessor mit geteiltem Speicher. Diese Rechner sind für das sehr heterogene Aufgabenfeld der Computer Algebra am besten geeignet, da sie die allgemeinste Form der Parallelverarbeitung erlauben. Die Implementation führen wir zunächst auf zwei verschiedenen Multiprozessoren durch. Zum einen auf einer Sequent Symmetry und zum anderen auf einem Netzwerk von Workstations. Die Sequent Symmetry ist ein System mit 10 Intel 80386 Prozessoren, die über einen System Bus auf ein oder mehrere Speicherbänke zugreifen können. Die Simulation eines Multiprozessors durch ein lokales Netzwerk von Workstations wird mit Hilfe der Cosmic Environment Software des Caltech durchgeführt. Mittels einer eigenen Speicherverwaltung wird der Speicher aller beteiligten Rechner auf einen logischen Adressraum abgebildet. Diese Speicherverwaltung ist die Hauptkomponente der Multiprozessorsimulation. Die Anpassung von MuPAD auf andere Architekturen geschieht durch den Austausch der Speicherverwaltung.

3.1 MuPAD Memory Management

Die Speicherverwaltung sorgt dafür, daß es für den Programmierer nicht sichtbar ist, welche Maschine tatsächlich verwendet wird. Für ihn handelt es sich immer um eine shared – memory Maschine. Zur Ersparnis von Speicherplatz und Laufzeit unterstützt die Speicherverwaltung auch die eindeutige Darstellung von Daten im Speicher. Falls es nötig ist, führt die Speicherverwaltung selbständig eine Verschiebung von Daten durch. Zum einen wird dadurch eine Speichersegmentierung vermieden, zum anderen ist dies bei der Netzwerkimplementation nötig, wenn Daten von einem Prozessor auf einen anderen aus Platzgründen ausgelagert werden. Bei dem Zugriff mehrerer Prozessoren auf ein Datum sorgt die Speicherverwaltung für die Synchronisation der Zugriffe und gewährleistet die notwendige Datenkonsistenz.

3.2 Interne Datentypen

MuPAD ist in der Programmiersprache C geschrieben. Alle Daten, die vom Benutzer in das System eingegeben werden, werden intern mit Typen versehen. Ihre Auswertung kann daher unabhängig vom Kontext geschehen. Die Speicherung erfolgt intern in einer Struktur, die aus einem statischen und einem dynamischen Teil besteht. Dadurch ist die Speicherung n-ärer Bäume mit konstantem Zugriff auf die Nachfolger möglich. Alle in Abschnitt 2.1 aufgeführten Typen werden in dieser einheitlichen Struktur gespeichert.

3.3 Parallelität

Auf Benutzerebene drückt sich die Parallelität durch die Kontrollkonstrukte *parbegin, for ... parallel* und durch die Funktionen zur Bindung von Daten und Prozessen aus. Auf interner Ebene wird ein *Problemheap* für die Umsetzung dieser Konstrukte eingesetzt. Dieser Problemheap sorgt für die gleichmäßige Arbeitsauslastung der Prozessoren. Jeder Prozessor, der zuviel Arbeit hat, schreibt diese in den Problemheap, und jeder Prozessor, der zuwenig Arbeit hat, entnimmt Aufgaben aus dem Problemheap. Mit diesem Mechanismus regelt sich das Problem der Lastbalancierung von selbst. Ein Großteil der verwendeten Parallelität steckt auf interner Ebene in der Speicherverwaltung.

4 Ausblicke

Das Computer Algebra System MuPAD steckt noch mitten in der Entwicklung, und wir bemühen uns ständig um eine Erweiterung und Verbesserung seiner Fähigkeiten. An dieser Stelle möchten wir einige Projekte erwähnen, die zur Zeit in Arbeit sind, bzw. in allernächster Zukunft in Angriff genommen werden.

- Entwicklung eines Graphik Paketes.
- Erweiterung der Libraries.
- In Kürze wird MuPAD spezielle Pakete für die Behandlung nichtlinearer dynamischer Systeme bereitstellen.
- Die Möglichkeiten zur automatischen Programmgenerierung werden erweitert und die Tools zur Manipulation von bestehenden Programmen werden vervollständigt.
- Zu einem späteren Zeitpunkt wollen wir Algorithmen zur automatischen Parallelisierung von Nutzerprogrammen bereitstellen.

Der Mathematik-Assistent–DERIVE Version 2

Bernhard Kutzler
Soft Warehouse Europe, A–4232 Hagenberg

1 Einleitung

Der Mathematik-Assistent DERIVE wurde entwickelt mit dem Ziel, Computeralgebra auf einfachster Hardware zur Verfügung zu stellen. Die Hardwareanforderungen, Bezugsquellen und weiterführende Literatur werden am Schluß dieses Artikels angegeben.
Die Autoren von DERIVE sind David Stoutemyer und Albert Rich von *Soft Warehouse, Inc.* (Honolulu, Hawaii). DERIVE ist eine Weiterentwicklung des Computeralgebra-Systems muMATH, es beinhaltet daher mehr als zehn Jahre Erfahrung in der Implementierung von Computeralgebra-Algorithmen auf einfachsten Rechnern: muMATH war bereits Anfang der 80er Jahre auf CP/M-Rechnern mit nur 64K Hauptspeicher verfügbar!
DERIVE Version 1 kam im November 1988 auf den Markt, die Version 2 folgte im November 1990. Bis Mitte Sommer 1991 ergab sich folgende Verbreitung:

- Weltweit sind etwa 30,000 Lizenzen im Einsatz, davon ca. 7,500 in Europa, davon ca. 5,500 in Deutschland und Österreich.
- Das österreichische Unterrichtsministerium hat DERIVE für alle Gymnasien angekauft. Ab Herbst 1991 ist DERIVE dort Standardwerkzeug im Mathematikunterricht.
- Die Militärakademie in Westpoint (USA) stattet jeden neuen Studenten mit einer Einzellizenz von DERIVE aus.
- Ein deutsches Handbuch gibt es seit Oktober 1989. Eine französische, eine italienische und eine holländische Übersetzung sind jeweils ab Herbst 1991 verfügbar.

Das Produkt ist als „elektronischer Schmierzettel“ konzipiert. Der Benutzer kann auf den Bildschirm Terme schreiben, ändern, auswerten, lösen, differenzieren, integrieren, plotten, löschen, umordnen, etc. Er oder sie kann mit DERIVE numerisch, algebraisch und graphisch arbeiten und programmieren. Die Benutzeroberfläche ist ähnlich wie die der Textverarbeitungssoftware WORD von Microsoft.
Die folgenden Abschnitte fassen die wichtigsten Merkmale des Systems zusammen. Es wurden bewußt einfachste Beispiele gewählt.

2 Ein paar Beispiele

Die Eingabe erfolgt durch Auswahl des Kommandos **Author**. Der Benutzer gelangt in einen Zeileneditor und wird zur Eingabe aufgefordert. Die Eingabe von `5(9-2)/3` bewirkt, daß im Arbeitsfenster das folgende Bild erscheint :

1: $\frac{5\ (9\ -\ 2)}{3}$

Der Term wird „zweidimensional“ ausgegeben und mit einer Termnummer versehen. Ausgerechnet (d.h. vereinfacht) wird der Term erst durch Wahl des Kommandos **Simplify**:

2: $\frac{35}{3}$.

Standardmäßig rechnet DERIVE im exakten Rechenmode (d.h. mit rationaler Arithmetik). So wird $\sqrt{45}$ vereinfacht zu

3: $3\ \sqrt{5}$

Auf Wunsch kann ein näherungsweiser Rechenmode eingestellt werden, wobei die Anzahl der signifikanten Stellen anzugeben ist. Mit 24-stelliger Arithmetik ergibt sich für $\sqrt{45}$ die Zahl

4: 6.70820393249936908922

Ebenso könnte man mit 250, 1400, oder sonst einer Zahl signifikanter Stellen rechnen. Das kostet allerdings Rechenzeit und Speicherplatz, womit sich eine praktische Beschränkung der größtmöglichen Stellenzahl ergibt. Eine Besonderheit von DERIVE ist, daß im näherungsweisen Rechenmode eine approximative Bruchzahlarithmetik verwendet wird. Diese ist zwar

etwas langsamer als die übliche Gleitkommaarithmetik, zeigt aber ein wesentlich besseres Fehlerfortpflanzungsverhalten.
Gibt man in DERIVE mit **Author** `(x-2y)^2` ein, so erscheint am Bildschirm

5: $(x - 2\ y)^2$

Das Ausmultiplizieren dieses Ausdrucks erreicht man mit dem Befehl **Expand**:

6: $x^2 - 4\ x\ y + 4\ y^2$

Terme können faktorisiert, auf gemeinsamen Nenner gebracht, nach einer Variablen gelöst werden, u.v.a.m. Möchte man einen Ausdruck integrieren, so gibt man zunächst den Term ein:

7: `x SIN (x)`

Der Befehl **Calculus** führt in ein Untermenü, in welchem man **Integrate** wählt. DERIVE erfrägt die Integrationsvariable (wir belassen die vorgeschlagene Variable x) und die Integrationsgrenzen (wir geben keine Grenzen ein, da eine Stammfunktion berechnet werden soll). DERIVE antwortet mit der formalen Anwendung des Integrationsoperators:

8: $\int$ `x SIN (x) dx`

Erst die Anwendung von **Simplify** bewirkt eine „tatsächliche Berechnung" des Integrals:

9: `SIN (x) - x COS (x)`

Im Untermenü **Calculus** stehen auch Befehle zum Differenzieren sowie zum Berechnen von Grenzwerten, Produkten, Summen und Taylorpolynomen zur Verfügung. Man muß also nicht die Namen der Funktionen wissen, mit denen man differenzieren, integrieren, etc. kann, und man muß auch nicht Anzahl, Art und Reihenfolge der nötigen Argumente kennen. Ein versierter Benutzer kann aber den vielleicht als zeitaufwendig empfundenen Menüdialoges durch Eingabe des Termes `INT(xSINx,x)` umgehen.
Die Anwendung von **Simplify** auf den Term

10: $\sqrt{(x\ y)} - \sqrt{x}\ \sqrt{y}$

bewirkt keine Vereinfachung. Viele würden hier 0 erwarten und sind vielleicht enttäuscht. Doch: Substituiert man für x und y je -1 (in DERIVE geht das mit dem Befehl **Manage Substitute**), so ergibt sich

11: $\sqrt{((-1)\ (-1))} - \sqrt{(-1)}\ \sqrt{(-1)}$

und weiter mit **Simplify**

12: 2

Der ursprüngliche Term ist nur dann equivalent zu 0, wenn x und y als nichtnegativ vorausgesetzt werden! Definiert man x und y mit dem Befehl **Declare Variable Domain Nonnegative** ebenso und vereinfacht dann den Term Nummer 10, so ergibt sich das Wunschergebnis

13: 0

DERIVE ist konservativ bei der Anwendung von Umformungsregeln – eine Eigenschaft, die ein gutes Computeralgebrasystem besitzen muß, denn kein Ergebnis ist besser als ein falsches Ergebnis! In diesem Punkt nimmt es DERIVE mit jedem seiner „großen Brüder" auf. Zum Beispiel wird der Ausdruck

$$\int_{\frac{\pi}{4}}^{\frac{3\pi}{4}} \frac{1}{2}\sqrt{2 - 2\cos(2x)}dx$$

von *Mathematica* zu 0 vereinfacht. Das ist falsch! DERIVE hingegen findet das korrekte Ergebnis, nämlich $\sqrt{2}$.

3 Eine Übersicht

Im Folgenden werden die weiteren Leistungsmerkmale des Systems stichwortartig aufgezählt.

▷ In DERIVE stehen zahlreiche Funktionen zur Verfügung sowie Know-How über Zusammenhänge und Vereinfachungsregeln: Exponentialfunktionen, Logarithmusfunktionen, trigonometrische Funktionen, trigonometrische Umkehrfunktionen, hyperbolische Funktionen, hyperbolische Umkehrfunktionen, stückweise stetige Funktionen, Funktionen aus der Wahrscheinlichkeitstheorie und der Statistik, Fehlerfunktionen und finanzmathematische Funktionen.

▷ DERIVE versteht sich auf das Rechnen mit Vektoren und Matrizen: Neben den Elementaroperationen wie Addition, Multiplikation mit einem Skalar, Skalarprodukt, Kreuzprodukt, Determinante, Transponierte, Inverse, etc., können auch Eigenwerte berechnet und differentielle und integrale Vektoranalysis betrieben werden.

▷ Neben dem numerischen und dem algebraischen Rechnen erlaubt DERIVE auch die Erstellung von 2- und 3-dimensionalen Graphiken. Die zu zeichnenden Kurven können dabei in kartesischen Koordinaten, in Polarkoordinat en oder in Parameterdarstellung gegeben sein. Zahlreiche Befehle zum Skalieren, zur Bildausschnittwahl, etc., stehen zur Verfügung.

▷ Das Arbeitsfenster von DERIVE kann beliebig unterteilt werden. Jedes dabei entstehende Fenster kann als Algebra-Fenster, als 2D-Plot-Fenster oder als 3D-Plot-Fenster definiert werden. Das nützt sowohl bei der Erstellung von Graphiken (Term und Schaubild sind gleichzeitig sichtbar) als auch beim Experimentieren mit Termumformungen: man kann zwei (oder mehr) Ideen in verschiedenen Fenstern verfolgen und die entstehenden Ergebnisse vergleichen. Das beste Resultat behält man. Alternativ dazu können Fenster auch übereinandergelegt werden.

▷ In DERIVE kann man auch eigene Funktionen definieren. Dabei können Iterationen, Rekursionen, IF-THEN-ELSE, logische Operatoren und alle DERIVE-Funktionen verwendet werden. Der Benutzer hat damit eine Art funktionale Programmiersprache zur Verfügung, die beliebige Erweiterungen des Systems erlaubt.

▷ DERIVE Version 2 beinhaltet eine Sammlung von 20 Dateien, in denen Funktionen zu zahlreichen Spezialthemen enthalten sind, wie zum Beispiel:
- Lösen nichtlinearer Gleichungssysteme,
- Numerische Differentiation und Integration,
- Anwendungen der Differentialrechnung,
- Anwendungen der Integralrechnung,
- Lösung gewöhnliche Differentialgleichungen,
- Lösung rekursiver Gleichungen,
- Wahrscheinlichkeitsfunktionen,
- Fresnelintegrale,
- Bessel- und Airyfunktionen,
- Hypergeometrische Funktionen,
- Elliptische Integrale,
- Orthogonale Polynome,
- Zeta-, Poly- und Dilogarithmusfunktionen,
- Plotten von Raumkurven und komplexen Kurven.

Diese Dateien können bei Bedarf geladen werden, womit die darin definierten Funktionen dann zur Verfügung stehen.

4 Hardwareanforderungen

DERIVE benötigt einen PC-kompatiblen Rechner mit 512K Hauptspeicher, MS-DOS Version 2.1 (oder höher) und ein $5\frac{1}{4}$" oder ein $3\frac{1}{2}$" Diskettenlaufwerk. Alle gängigen Graphikkarten werden unterstützt. DERIVE ist das einzige Computeralgebra-System, das auf einfachsten PC's läuft. Praktisch jeder PC-kompatible Rechner – Desktop, Laptop, ... – erfüllt obige Anforderungen. Insbesondere ist DERIVE auch auf den neuen Palmtop-PC's verwendbar:

- Der POQET PC paßt mit seinen 22.3 x 10.9 x 2.5 cm in die Westentasche und wiegt nur 550 Gramm. Zwei handelsüblichen AA Alkalibatterien erlauben bis zu 100 Betriebsstunden.
- Der in Abb. 1 gezeigte HP 95LX ist mit seinen 16.0 x 8.64 x 2.54 cm noch kleiner und mit seinen 303 Gramm noch leichter als der POQET. Auch der 95LX wird mit zwei AA Batterien betrieben. Der etwas kleinere Bildschirm mit nur 16 Zeilen und 40 Spalten hat eine kleine Adaption der Software nötig gemacht. Ab Herbst 1991 ist DERIVE-95LX auf ROM-Karten verfügbar.

Abbildung 1: HP 95LX

DERIVE auf den neuen Palmtop-PC's bedeutet eine „Revolution“, die noch sehr viel weitreichender ist, als es die Einführung des numerischen Taschenrechners war: jeder Techniker, jeder Ingenieur, jeder Wissenschafter, jeder Student, jeder Lehrer und jeder Schüler kann ab sofort einen leistungsfähigen Mathematik-Assistenten ständig bei sich tragen. Die sich daraus ergebenden Konsequenzen und Möglichkeiten sind heute noch gar nicht abzusehen.
In den Schulen werden immer heftigere Diskussionen zum Thema DERIVE geführt: Da DERIVE bis zu 80% der in heutigen Lehrbüchern enthaltenen Aufgaben „auf Knopfdruck“ löst, ist mit der Verfügbarkeit dieses neuen Werkzeuges eine drastische Änderung des Mathematikunterrichtes vonnöten. Das birgt eine große Chance, aber auch eine große Gefahr in sich: der Unterricht kann endlich weg von den handwerklichen Fähigkeit und hin zu den Konzepten gestaltet werden – doch würde genau damit der Mathematikunterricht auch entsprechend anspruchsvoller. Österreich ist das erste Land der Welt, in dem DERIVE Standardwerkzeug an allen Gymnasien ist. Die Verantwortlichen vieler Länder erwarten daher

mit Spannung die Ergebnisse des im Herbst 1991 in Österreich beginnenden „Großversuches.“

5 Bezugsquellen

DERIVE wird über Vertragshändler vertrieben. Eine Liste der europäischen Händler kann angefordert werden bei

Soft Warehouse Europe GmbH
Schloß Hagenberg
A-4232 Hagenberg, Österreich
Tel +43-7236-3297-81, Fax +43-7236-3297-30.

Die Adressen der Händler außerhalb Europas erhält man beim Entwickler

Soft Warehouse, Inc.
3615 Harding Avenue, Suite 505
Honolulu, Hawaii 96816, USA
Tel +1-808-734-5801, Fax +1-808-735-1105.

6 Literatur

Die wichtigste Quelle ist das ca. 250 Seiten starke Handbuch. In diesem sind auch alle bisher erschienenen Bücher über DERIVE angeführt.
Die Anfang 1991 gegründete *DERIVE User Group* gibt den viermal jährlich erscheinenden *DERIVE News Letter* heraus. Diese Publikation beinhaltet Leserbriefe und Artikel. Die Kontaktadresse ist

DERIVE User Group
att. Josef Böhm
D'Lust 1
A-3042 Würmla, Österreich
Tel +43-2275-8207.

CAYLEY - eine kurze Beschreibung

Gerhard J. A. Schneider
Institut für Experimentelle Mathematik, Universität Essen

Im Gegensatz zu den allgemeinen Computeralgebrasystemen richtet sich CAYLEY an den Nutzer, der Probleme aus speziellen Gebieten der Algebra und Zahlentheorie zu lösen hat. Hierfür bietet es eine Fülle von fertig implementierten, effizienten und ausgetesteten Algorithmen an, die sowohl einfach aufgerufen als auch über eine eigene Kommandosprache zu wirkungsvollen Programmen verknüpft werden können. Die dazugehörigen Datenstrukturen sind im System bereits vorhanden, so daß die Formulierung der Probleme auf eine für den Mathematiker natürliche Weise geschehen kann.

Das System selbst umfaßt mittlerweile ca. 7 MByte C-code und die kompilierte Version hat eine Größe von 1.9 – 2.7 MByte, je nach Hardware. CAYLEY ist für eine Vielzahl von Platformen verfügbar, darunter SUN3, SUN4, VAX/VMS, CADMUS und verschiedene IBM-Systeme, wie RS6000, RT6150, PS2(AIX) und VM/CMS. Die CADMUS- und IBM-Portierungen werden in Essen durchgeführt.

Neben dem eigentlichen System selbst wird eine Datenbank ausgeliefert, die eine Fülle von Beispielen enthält. So sind alle Gruppen der Ordnung bis 100, die der Ordnung 128 und 256 ebenso verfügbar wie alle einfachen Gruppen bis zur Ordnung 10^6, transitive Gruppen bis zum Grad 12, perfekte Gruppen, etc. Der Umfang dieser Datensammlung, die sehr leicht ins System eingelesen werden kann, umfaßt weitere 8 MByte.

Neben der Forschung kann CAYLEY auch erfolgreich in der Lehre eingesetzt werden, wie die Erfahrungen in Sydney, Essen und Zürich gezeigt haben. Es lassen sich Aufgaben formulieren und interaktiv bearbeiten. Somit kann der Teilnehmer zu einer richtigen und effizienten Lösung geführt werden und lernt quasi nebenbei die Benutzung eines großen Systems. Insbesondere wird man auf die Unterschiede zwischen mathematischer Denkweise und der computertechnischen Realisierung aufmerksam. Ein typisches Problem für Anfänger ist das Kommando `for each x in G do` - das Programm führt die Anweisungen wirklich für jedes einzelne Element durch, was bei großer Gruppenordnung zu gewissen Problemen führen kann (z. B. Laufzeiten von mehreren Jahrhunderten).

Aus Erfahrung kann ich sagen, daß die Grundlast, die CAYLEY vom Host anfordert, nicht allzu hoch ist. Bereits auf einer kleinen Workstation lassen sich mehrere Übungssitzungen gleichzeitig durchführen. Für die Forschung ist der verfügbare Hauptspeicher entscheidender. CAYLEY kann derzeit bis zu 64 MByte Datenspeicher verwalten, bescheidet sich aber auch mit weniger. Erfreulich ist, daß der Speicherbedarf nicht unkontrolliert
wächst, sondern vom Benutzer bei Bedarf erhöht werden muß. Damit ist sichergestellt, daß man nicht andere Jobs (ohne es zu wissen) stört.

Es gibt ein umfangreiches Handbuch zum System, das allerdings — ein häufig vorgebrachter Kritikpunkt — aufgrund zahlreicher Updates und Ergänzungen nicht mehr sehr systematisch aufgebaut ist. Im Durchschnitt erscheint alle 18 Monate ein neues Release. Derzeit ist Version 3.7 aktuell, 3.8 sollte in nicht allzu ferner Zukunft verfügbar sein. Außerdem werden in unregelmäßigen Abständen Benutzertreffen abgehalten. Das letzte Treffen fand Ende 1988 in Essen statt; die Proceedings sind Anfang dieses Jahres im SIGSAM Bulletin, vol 25, erschienen.

Ich möchte nun einige Aspekte des Systems anhand von Beispielen erläutern. Ich hoffe, daß damit das Interesse an weiteren Fragen bei den Demonstrationen auf der DMV-Tagung geweckt ist.

Zunächst wird eine Permutationsgruppe definiert, dann die Ordnung berechnet und eine Sylow-2-Gruppe bestimmt. Anschließend wird die Charaktertafel dieser Untergruppe berechnet und die Anzahl der Charaktere ausgegeben - aus Platzgründen wird auf die Ausgabe der ganzen Charaktertafel verzichtet

```
>G:PERMUTATION GROUP(12);
>G.GENERATORS:
> A = (1,2,3,4,5,6,7,8,9,10,11),
> B = (1,9)(2,6)(4,5)(7,8),
> C = (1,10)(2,5)(3,7)(4,8)(6,9)(11,12);
>print order(g);
95040
>s=sylow(g,2);
>ct=character table(s);
>print length(ct);
16
```

Ganz analog geht man vor, wenn eine Gruppe mit Erzeugern und Relationen einzulesen ist. Die Kommandos sind die gleichen, CAYLEY erkennt selbst, welche internen Algorithmen für die Struktur am besten geeignet ist.

```
>G:FREE(X,Y);
>G.RELATIONS:X^2=Y^4=(X*Y)^11=(X*Y^2)^6=
>(X*Y)^2*(X*Y^-1)^2*X*Y*X*Y^-1*X*Y^2*X*Y*X*Y^-1=1;
>print order(g);
  7920
>s=sylow(g,2);
```

Um nun die Kompositionsfaktoren der Darstellung der Gruppe $Co3$ auf 276 Ziffern über $GF(2)$ zu bestimmen, geht man am besten wie folgt vor:

```
>library co3
 Library module found as
/local/cayley/pergps/co3
  Conway's simple group co3 on 276 letters.
  Order:  495 766 656 000 = 2^10 * 3^7 * 5^3 * 7
* 11 * 23
  Base:  1,2,3,4,8,5; Basic Orbits:  276 * 275 *
162 * 105 * 64 * 6
  Stabilizer of set [1,2,...,23] is Mathieu
group M23
  Stabilizer of set [1,2,...,100] is Higman-Sims
group HS
  Group G; Generators:  A, B;
>v=vector space(276,field(2));
>p=permutation module(v,g);
>s=composition series(p);
>print s;
```

Aus der Darstellungstheorie sind neben dem Algorithmus zur Bestimmung der Charaktertafel auch die Meataxe implementiert. Diese Implementation arbeitet für eine beliebige Anzahl von Erzeugern über beliebigen Körpern; insbesondere wird auch der Basiswechsel dem Benutzer mitgeteilt.

Im letzten Release von CAYLEY sind auch eine Reihe von zahlentheoretischen Funktionen hinzugekommen. Neben der schon lange vorhandenen Langzahlarithmetik gibt es nun eine Reihe von Faktorisierungsalgorithmen und Primzahltests.

```
>n=pfactorisation(2^67-1;al:elliptic);
>print n;
    SEQ( 193707721, 1, 761838257287, 1 )
```

Die erste Faktorisierung dieser Zahl dauerte 20 Jahre, mit CAYLEY auf einer RS6000 ist das Ergebnis in wenigen Sekunden verfügbar.

CAYLEY selbst ist lizenzpflichtig; genauere Auskünfte hierzu erteilt Prof. J. Cannon, Dept of Pure Mathematics, University of Sydney, Sydney 2006, Australien.

Ich möchte diesen Überblick abschließen mit einer kurzen und allgemeinen Systembeschreibung, die J. Cannon selbst gegeben hat:

The CAYLEY *System for Discrete Algebraic and Combinatorial Computation*

CAYLEY is designed to support fast computation in important areas of modern algebra (*e.g.* groups, rings, modules), number theory and finite geometry. Rather than being an alternative to other Computer Algebra systems, CAYLEY provides coverage of new areas of mathematics and it supports a different style of computation. Features include:

- The system is designed around the concept of having the user explicitly define the specific algebraic structure with which he/she wishes to compute. Thus, the user may define a range of different types of algebraic structures such as finite and infinite groups, rings, fields and modules.
- Fast computation in important classes of algebraic structures. For example, it is possible to calculate with permutation groups of degree up to a million;
- Emphasis on obtaining global information about structures: order of a finite group, centre of a group, subgroups of small index in an infinite finitely presented group, composition series of a KG-module;
- A user programming language in which the data types correspond to the central concepts of modern algebra: algebraic structures, algebraic elements, sets, sequences, and mappings;
- A unique run-time environment whereby natural mappings of the form subalgebra → algebra and algebra → quotient algebra are handled automatically.
- Several hundred built-in functions capable of determining deep structural properties of groups and other objects. For example, one such function will display the composition factors of a permutation group.
- Data bases containing many useful examples of structures (mainly groups) are supplied with the system which further enhance its knowledge-base.

CAYLEY (currently running at 240 sites) has found wide application to problems arising in many branches of mathematical research including group theory, representation theory, topology, knot theory, finite geometry, number theory, combinatorial theory and graph theory. Further, the system has successfully solved problems arising in application areas such as complexity theory, coding theory, data encryption, communication network design, discrete fourier transforms, mathematical crystallography and solid state physics.

Macaulay — ein Computeralgebrasystem für Algebraische Geometrie und Kommutative Algebra

Kurt Behnke
Mathematisches Seminar, Universität Hamburg

1 Übersicht

Das Computeralgebrasystem Macaulay wurde von D. Bayer (Rutgers) und M. Stillman (Cornell) entwickelt. Derzeit liegt es in der Version 3.0 vor. Es ist frei verfügbar, und kann im C-Quellcode per ftp kopiert werden. Der Code läßt sich auf den meisten UNIX-Workstations ohne Probleme kompilieren — für die gängigsten Maschinen liegen makefiles bei. Daneben existiert eine kompilierte Version für den Apple Macintosh, die als Tool unter der MPW-Shell eingesetzt wird. Sie bietet im Vergleich zu UNIX-Installationen gesteigerten Bedienerkomfort.

Macaulay ist als Werkzeug für einen algebraischen Geometer oder kommutativen Algebraiker gedacht. Es erlaubt algebraische Manipulationen mit Polynomen, Matrizen, Idealen, Polynomringen, Moduln, Abbildungen zwischen Ringen und Moduln, und Komplexen von Moduln. Basisoperationen sind die Berechnung von Gröbner-Basen, Syzygienmoduln, endlichen freien Auflösungen, Hilbert-Polynomen. Auf der Basis dieser Operationen sind komplexere Aufgaben von Projektionen algebraischer Varietäten über Idealquotienten bis hin zur Berechnung von Garbencohomologie kohärenter Garben auf projektiven Varietäten auf komfortable Weise möglich.

Aufgaben wie das Lösen von Gleichungssystemen oder etwa das Faktorisieren von Polynomen gehören nicht zum Systemumfang. Es gibt keine Ganzzahlarithmetik und keine Graphik.

Für die wichtigsten Operationen müssen alle Objekte über einem geeignet positiv graduierten Polynomring homogen sein. Zusätzlich ist dieser Polynomring definiert über einem Körper der Charakteristik $p > 0$, mit $p = 31991$ voreingestellt. Für den „normalen" Computeralgebra-Anwender mag ein solches Programm nutzlos sein; in der Algebraischen Geometrie und der Kommutativen Algebra sind diese Einschränkungen akzeptabel. Dort auftretende algebraische Objekte sind häufig graduiert, speziell in der projektiven Geometrie. Invarianten wie Multiplizität, Codimension, Dimensionen von Cohomologiegruppen sind stabil bei Reduktion modulo geeigneter Primzahlen. In der Regel sind mit Macaulay erzielte Ergebnisse nach Charakteristik 0 übertragbar. Durch die genannten Beschränkungen erlaubten es, ein sehr leistungsfähiges System zu entwickeln. Mit Macaulay kann man auf einer PC-Plattform (Macintosh) Probleme bearbeiten, die bis heute außerhalb der Möglichkeiten der Gröbner-Packages von „General-Purpose" Computeralgebra-Paketen liegen. Macaulay unterstützt den theoretisch arbeitenden Mathematiker in seiner Arbeit. Es erlaubt das schnelle Testen von Vermutungen und Experimente mit komplizierten Polynomidealen.

2 Interaktion mit dem System

Macaulay zeichnet, falls es gewünscht wird, die aktuelle Sitzung in eine m Monitorfile auf. Ergebnisse von Berechnungen können in einem Standardformat in Dateien geschrieben und wieder gelesen werden. Das Kommando `putmat (Variable) >(Dateiname)` führt dies aus.

Macaulay-Berechnungen können vom Benutzer unterbrochen werden. Auf UNIX-Systemen geschieht das durch Eingabe von (CNTRL)-C, auf dem Macintosh genügt ein Mausklick. Danach ist Macaulay auf einem neuen Eingabelevel (angezeigt durch einen veränderten Prompt, s. u.), und das volle System steht zur Verfügung, um die Zwischenergebnisse zu untersuchen. Zum Beispiel kann man sich den Speicherplatzverbrauch (`space`), die Größe der bisher berechneten Standardbasis (`size (matrix)`) und die Zahl der noch abzuarbeitenden S-Polynome (`spairs`) angeben lassen. Bei Standardbasenkommandos und Auflösungen wird die Rechnung anschließend durch `continue` + (Enter) dort fortgesetzt, wo sie unterbrochen worden war. Durch `exit all` wird das Programm aus der Unterbrechung heraus gestoppt.

Einige Kommandos lassen sich anhalten, bevor sie vollständig ausgeführt sind, können aber anschließend nicht fortgesetzt werden.

3 MACAULAY-Funktionen

Da MACAULAY über 120 Kommandos verfügt, können hier nur einige der Funktionen beschrieben werden. Eine MACAULAY-Sitzung beginnt in der Regel mit der Definition eines Grundringes. Nach Eingabe von `ring` (Name) wird man auf den jeweiligen Prompt Grundcharakteristik, Anzahl und Namen der Variablen, Variablengewichte und die Monomiale Ordnung eingeben. (Enter) ohne einen Eintrag bestätigt jeweils die Voreinstellung.

MACAULAY gestattet eine weitgehend freie Vereinbarung von monomialen Ordnungen. Voreingestellt ist die graduiert invers-lexikographische Ordnung. Produktordnungen und die sogenannte Eliminationsordnung können auf einfache Weise eingestellt werden. Die Eliminationsordnung erhält man, indem man auf den Prompt nach der monomialen Ordnung `w` (für weight vector), und auf den nächsten Prompt einfach den Namen der letzten Variablen, die eliminiert werden soll, eingibt. Für die Berechnung von Standardbasen von Moduln kann man sich entscheiden, ob zunächst die lexikographische Ordnung der Basisvektoren oder die monomiale Ordnung der Koeffizienten ausgewertet werden soll.

Es können mehrere Ringe vereinbart werden, von denen zu jedem Zeitpunkt genau einer der *current ring* ist. Der *current ring* kann durch das `setring`-Kommando gewechselt werden.

Polynome, Ideale und Moduln (genauer: Untermoduln von freien Moduln) werden als Matrizen der Dimensionen 1×1, $1 \times m$ und $n \times m$ eingegeben. Die Erzeuger der Moduls sind dabei die Spalten der Matrix. Allgemeinere Moduln können dann natürlich mittels endlicher Präsentationen behandelt werden. Wo immer Ideale eingegeben werden, und Moduln als Parameter sinnvoll sind, sind sie auch zulässig. Mit Matrizen sind die üblichen algebraischen Manipulationen, wie Addition, Multiplikation möglich. Man kann zwei Matrizen gleicher Zeilenzahl zusammenfügen (`concat`), und man kann direkte Summen endlich vieler Matrizen herstellen. Bei den erzeugten Moduln entspricht das der Summenbildung von Untermoduln bzw. der direkten Summe.

Nach Berechnung einer Gröbner Basis (mit `std`) kann man z. B. die Poincaré-Reihe, geschrieben als rationale Funktion, abfragen. (Un)sinnigerweise erfolgt das mit dem Kommando `hilb`, das viel eher auf Hilbertfunktion oder Hilbertpolynom hindeutet. Das `hilb`-Kommando hat bei großen Standardbasen lange Laufzeiten und einen hohen Speicherplatzverbrauch. Daher gibt es schnelle Algorithmen, die die grundlegenden Invarianten Codimension und Grad (Multiplizität) bestimmen. Mit `codim m` sucht man z. B. nach einer minimalen Teilmenge der Variablen, die ein assoziiertes Primideal des von den Leitmonomen von `m` erzeugten Moduls bilden. Mit `lift` bzw. `reduce` werden die Erzeuger eines Untermoduls als Linearkombinationen der Standardbasiselemente dargestellt.

Das Kommando `res x c` berechnet die minimale freie Auflösung des Moduls `x` und schreibt sie als eine Folge von Matrizen `c.0, c.1,...`. Die numerischen Daten des Komplexes `c` kann man mit `numinfo c` bzw. `betti c` abfragen. Jeder Modul `c.i` hat automatisch eine Standardbasis. Mit dem Kommando `lift` wird dann einfach der Cokern einer Modulabbildung berechnet. Da der Kern einer Abbildung durch den 1. oder 2. Syzygienmodul dargestellt wird (je nachdem ob man Erzeuger oder eine endliche Präsentation haben möchte), bekommt man so ein Werkzeug zur Berechnung der Homologie eines Komplexes.

Nach Berechnung der Standardbasis eines Ideals kann man mit `qring` zum Quotientenring übergehen. Über Quotientenringen sind die gleichen Kommandos wie über vollen Polynomringen möglich — natürlich berechnet `std` die Gröbner-Basis eines Ideals in einem Quotientenring nicht in dem Ring selbst, sondern in dem zugehörigen vollen Polynomring, unter Hinzufügen des ausdividierten Ideals.

Mit Ringabbildungen (`rmap`) werden Objekte von einem Ring zum anderen transportiert (Variablensubstitution). Besitzen zwei Ringe den gleichen Satz von Variablen (wie ein Ring R und sein Quotientenring R/I), so kann man einfacher das Kommando `fetch` anwenden.

Beispiel: Berechnung des Conormalenmoduls I/I^2 für ein Ideal $I \subset R$. Es wird angenommen, daß R existiert, und I über R definiert ist.

:`res I c 1`! löst I bis zum 1. Syzygienmodul auf.

:`qring c.0 S`! Der Ring $S = R/I$ wird „current ring."

:`fetch c.1 P`! Die Matrix $c.1$ präsentiert über S den Conormalenmodul.

Mit `k-basis P B` kann man sich nun z. B. eine Vektorraumbasis ausgeben lassen (besser für jeden homogenen Grad einzeln); mit `mult c.0 D` erhält man eine minimale Menge von Erzeugern für I/I^2.

4 Command-Scripts

MACAULAY ist (in einem sehr bescheidenen Rahmen) programmierbar. „This ain't Lisp we're talking about here" (Manual, Abschnitt 11.6). Ein Command-

Script ist eine Datei, die eine Liste von MACAULAY-Kommandos enthält. Einem Script können dabei Parameter für die Platzhalter #1, #2,...übergeben werden. Als Parameter sind alle Macaulay-Objekte, ganze Zahlen und alle Ausdrücke, die ganze Zahlen ergeben, zulässig. Scripts werden aufgerufen durch `<script par1 par2` .... Scripts können andere Scripts aufrufen. Zu Beginn einer Sitzung befindet sich MACAULAY auf einem Level 0, angedeutet durch den Prompt %. Nach Aufruf eines Scripts (wie nach Unterbrechung einer Rechnung durch `(CNTRL)-C`) erhöht sich der Level um 1, um anschließend wieder auf den Wert vor Beginn des Scripts zurückzukehren. Durch Setzen der Systemvariablen `prlevel` kann man alle Scripts oberhalb eines bestimmten Levels „still" ablaufen lassen.
Es gibt keine lokalen Variablen. Um Variablenkonflikte zu vermeiden, können Variablennamen in Scripts das Zeichen verwenden. Bei Aufruf des Scripts wird es durch @nn@ ersetzt, wo nn die zweistellige Zahl ist, die den jeweiligen Level bezeichnet. Dadurch erhält eine Zählvariable i@ auf Level 1 den Namen i@01@, auf Level 2 dagegen i@02@.
Ein Script kann *Label* enthalten. Ein Label ist eine Zeichenkette, gefolg t von einem Doppelpunkt, am Beginn einer Zeile. Sprungbefehle (`jump label`), bedingte Verzweigung (`if i < j (label1) (label2)`) und einfache Schleifenkonstruktionen sind damit möglich.
Die Erstellung eines Scripts (außer mit einem Texteditor) erfolgt aus MACAULAY heraus nach dem Kommando >>(Script-Name). Die Inputs des Benutzers werden in die Datei (Script-Name) kopiert. Beim abschließenden Redigieren müssen lediglich die Variablen durch Platzhalter ersetzt werden, und die Objekte, die nur in dem Script benutzt werden, im obigen Sinne lokal gemacht werden. Anschließend steht die Datei als Script zur Verfügung. Das MPW Frontend auf dem Macintosh bietet noch die Möglichkeit, das MPW-Eingabefile in gleicher Weise als Script zu editieren (sehr komfortabel—).
Mit dem Programm werden z. Z. 81 Command-Scripts, geschrieben von D. Eisenbud und M. Stillman, geliefert. Sie erledigen zahlreiche Standardaufgaben aus Algebraischer Geometrie und Kommutativer Algebra. Zur Illustration seien hier vier von ihnen mit einer kurzen Funktionsbeschreibung vorgestellt.

:annihilator! berechnet den Annihilator $\mathrm{Ann}_R M$ eines Moduls über einem Ring R.

:cohomology! berechnet

$$\bigoplus_{n\geq 0} H^i(M(n))$$

für einen Modul M, aufgefaßt als kohärente Garbe auf dem projektiven Raum.

:double_dual! bestimmt zu einem Modul M den doppelt-dualen M^{**}.

:points! berechnet das Ideal der Formen, die auf einer endlichen Punktmenge des projektiven Raumes verschwinden.

5 Schlußbemerkungen

Wie schon eingangs erwähnt, liegt MACAULAY im C-Quellcode, zusammen mit einigen Makefiles zur Installation auf UNIX-Maschinen vor. Zu den mit makefiles bedachten Workstations gehören Sun3, Sun Sparc und DecStation. Unter Verwendung von makefile.sun und makefile.sparc gelingt z. B. die Installation auf HP Workstations oder der neuen IBM Risc/6000 ohne Mühe, während das ältere Modell 6150 RT/PC unter AIX erhebliche Schwierigkeiten macht. Mehrere Installationsversuche auf einem IBM PS/2 unter AIX sind gescheitert.
Die Macintosh-Version verträgt sich nicht gut mit jeglicher Art von Menu-Clock und mit gewissen Screen-Savern. Es ist eine gute Idee, auf dem Mac vor dem Start alle kritisch erscheinenden Inits und Kontrollfelddateien zu deaktivieren. Das gravierendste Problem auf dem Macintosh ist die Unverträglichkeit von MACAULAY mit dem Multifinder. Im Hinblick auf System 7 sollte hier schnellstmöglich Abhilfe geschaffen werden.
Der jeweils aktuelle MACAULAY Quellcode liegt auf dem Server zariski.harvard.edu als M3.tar im Directory ftp/Macaulay. Es handelt sich um ein ca. 1,6 MByte großes Archiv, das neben dem C-Code auch noch das Manual als LaTeX-source und als Postscript file enthält. Das MACAULAY-Tool für den Macintosh unter MPW ist aus der gleichen Quelle als BinHex file zu beziehen. Die Shell wird allerdings nicht mitgeliefert.
Für den Autor ist MACAULAY seit Jahren das Computeralgebrasystem der Wahl. Der Hauptgrund liegt natürlich im Arbeitsgebiet „Algebraische Geometrie/ Singularitäten". MACAULAY ist wohl das einzige System, das Polynome und Ideale der dort auftretenden Komplexität bewältigen kann. Dank seines geringen Speicherbedarfs und seiner Schnelligkeit ist es ideal zum Experimentieren, Testen von Vermutungen und zum Rechnen von Beispielen.

The PARI-GP package

Christian Batut, Dominique Bernardi,
Henri Cohen, Michel Olivier and Nils–Peter Skoruppa

1 The PARI system

Computations in number theory often have three distinctive characteristics. Firstly, they use a large amount of computer time, but the type of computations that are performed are usually well suited to the use of powerful workstations, as opposed for example to mainframe or supercomputers.
Secondly, the data types and structures used are constantly the same, for example bignums (multiprecise integers), bigfloats (multiprecise real numbers), fractions, complex numbers, polynomials, etc..., but these structures are not directly available on the computer, and their use is a nontrivial task.
Thirdly, currently available symbolic packages are too slow and too clumsy for extensive number-theoretic applications, while providing a huge number of functions which are useless to number theorists. The reason for this is not a mystery: these packages are targeted to the applied mathematics, engineering and physics market, and not to the tiny number-theoretic one.
A consequence of this situation was that each time a number-theorist needed to perform some nontrivial task, he had either to limit himself to the speed and functions of existing systems, or to write his own routines. Very often those routines were specialized, and received little publicity.
The goal of the first four authors in the last five years or so has been to develop a package with three primary goals in mind, essentially corresponding to the three observations above. The first goal was one of speed. The second goal was the ease of handling the standard data types occuring in number theory. The third goal was to put as many number-theoretic functions and procedures as possible, and this is of course a neverending task. We believe that those goals have been achieved. Evidently bugs continue to appear (at quite a slow rate now), and improvements and additions are done continuously, but the system, which was nicknamed PARI (for a reason which we do not want to explain here), is as fully operational as any other, and has been in constant use in our department as well as many other places around the world for two years now.
The PARI system is a large collection of programs. Although it can be used as an ordinary library, the most frequent use of the system is through an interpreted interactive calculator, called GP (Great Programmable calculator!).
As already indicated, a special feature of the PARI system is its speed. Experiments have shown that it is between 5 and 100 times faster than general purpose symbolic systems.

2 The PARI types

In addition to single precision integers, PARI has 18 different kinds of objects ("types"), which we now describe.
First, we have multiprecise integers and multiprecise reals. The precise coding scheme used is not important (see the user's manual for this), but a point to be made is that each object carries with it its precision or length.
Then we have rational numbers (in reducible or irreducible form), integermods (i.e. elements of $\mathbb{Z}/N\mathbb{Z}$), complex numbers and more generally quadratic numbers, p-adic numbers, and polymods (i.e. elements of $A[X]/P(X)A[X]$ where A is a base ring). Complex and quadratic numbers as well as polymods are *recursive* structures, a word which is fundamental in the PARI system. This means for example that the real and imaginary part of a complex number can be of any reasonable type, for example multiprecise integer or real, p-adic or integermod, although this may not make any mathematical sense. Polymods are especially useful since when the base ring A is the field $\mathbb{Q}$, and if the polynomial $P(X)$ is irreducible over $\mathbb{Q}$, then a polymod is an element in the number field defined by a root of P. Similarly, if A is the field $\mathbb{Z}/p\mathbb{Z}$ for p prime, this allows us to work in finite fields.
Finally we come to the completely recursive structures: polynomials, power series and Laurent series, rational functions, vectors and matrices, the components of these structures being any legal PARI object. For example, a polynomial in two variables X and Y

is simply represented as a polynomial in X whose coefficients are in the ring of polynomials in Y. This is of course a poor representation for multivariate polynomials, but we did say that PARI is not especially built for that.

In addition to this, binary quadratic forms of positive or negative discriminant have also been included, but they may disappear in future versions. Similarly, since all the routines exist, one could have added the type "point on an elliptic curve", but this would be straying from the main goal of the package. As it stands now the routines to handle such objects have simply a more complicated name than $+$ or $*$.

3 Hierarchical organization of the system

The PARI system can be considered as having three hierarchical levels. The first and most basic level which we will call the *basic kernel* mainly consists of the four standard operations (addition, subtraction, multiplication, division) between multiprecise integers, multiprecise reals, and single precision integers, as well as conversions between these objects and other related routines. It is absolutely crucial for the speed of the entire system that this basic kernel be as fast as possible, hence, when possible, at least a part has been written in assembly language. Of course this raises the issue of portability, which we will address in the next section.

The second level in PARI is what we call the *generic kernel*. It is an even larger collection of programs to deal with standard operations and conversions, this time between all the 18 kinds of PARI objects, in addition to single precision integers. Speed is of course still quite important here, but experience shows that the main thing is to use the best algorithms, and that the use of assembly language at this level is not worthwhile, except maybe for one or two very basic algorithms, such as integer gcd (extended or not). In our case, everything has been written in C.

The third level in PARI is an even larger collection of routines (also written in C) for performing work on the PARI objects, including linear algebra, transcendental functions, numerical analysis, input-output, plus many programs which are more specifically number-theoretical. For example primality and factoring in $\mathbb{Z}$, in $\mathbb{Z}[X]$, in $\mathbb{F}_p[X]$, the LLL algorithm [LLL], Ford's round 2 algorithm for finding integral bases in number fields, a polynomial reduction algorithm due to F. Diaz y Diaz and H. Cohen, an elliptic curve package including Tate's reduction algorithm, computation of Néron-Tate heights, of the number of points over $\mathbb{F}_p$, etc..., several p-adic transcendental functions, class number and regulator computations (for the moment only for imaginary and real quadratic fields).

At the same level, there is also a module which is the syntactical parser for the GP calculator, and a module which contains short on-line help as well as error messages. The GP calculator allows the use of all the functions of the library, and has been designed so as to be programmable, with the usual basic requirements of a programming language: control structures, variables, user-defined functions in particular.

4 Implementation and portability issues

We come back to the question of the basic kernel. The first operational version of PARI was written on a Sun 3 computer, a 68020 based machine. For several reasons, this microprocessor is well suited for multiprecision operations, and the design decisions were made with this processor in mind. In particular, the most efficient version on PARI available until January 1991 was the version in which the entire basic kernel was written in 68020 assembly language (around 6000 lines of assembly code after removing the comments). Only since the use of PARI on a 28 Mips Sparcstation 2, have we achieved a definite improvement (about a factor of 2) compared to a 4 Mips Sun 3/260. Of course, had we used machines based on any other RISC processors than the Sparc, we would have had hardware or microcoded multiply and divide, and the gain would be much larger.

With the advent of very good optimizing compilers, it becomes less important to have such a large assembly language module. However, as long as a C compiler is not able to use directly existing machine instructions for doing 32x32 bit multiplies, giving a 64 bit result, and 64/32 bit divide giving 32 bit quotient and remainder, as well as simpler operations such as dealing with the carry bit in integer operations, at least *some* assembly will be required. This allows the system to be mostly portable, since the number of 32 bit microprocessor types in wide use is not that large (MC68020, R2000, MC88000, RS6000, 80386, Sparc). This is the choice that we have made for the Sparc processor, for which we have written a kernel mostly in C with in addition less than 500 lines of Sparc assembly. This C kernel can be used with any processor, at the cost of writing the equivalent of

these 500 lines of assembly language. Note that 500 may seem a large number, but this is entirely due to the fact that Sparc does not have multiply and divide. For any other processor, the size would be much smaller.
Finally, if nothing else is available, one can use a kernel entirely written in C, at an additional expense of speed.

5 What can PARI do for you

As is already implicit from the existing types PARI enables you to do first of all the standard operations of adding, multiplying and dividing in finite fields, algebraic number fields, p-adic fields, polynomial rings, in rings of formal power series with coefficients in any of the former structures.
At the next level PARI knows most of the higher transcendental functions. Usually all these functions operate on any object that makes sense, i.e. they accept as arguments complex numbers, matrices, vectors and some accept also p-adics.
Especially useful for the number theorist is the huge list of arithmetical functions and arithmetical functionalities built in in PARI including a gcd routine, divisor sums, kronecker symbols, class numbers etc.. PARI factors integers, of course, but it also factors polynomials over $\mathbb{Z}$, $\mathbb{Z}/p\mathbb{Z}$ or $\mathbb{Q}_p$.
You can do numerical analysis like numerical integration, root finding, eigenvalues and eigenvectors. There are tools for doing linear algebra like, e.g., routines for finding normal forms of matrices, for computing kernels, determinants, characteristic polynomials, intersection of subspaces, and, of course, for lattice reduction, for checking linear and algebraic dependence.
There are packages for more specialized number theoretic topics. There is a package for doing computations on elliptic curves. Another one does computations in algebraic number fields: you give an (irreducible) polynomial over $\mathbb{Q}$ and PARI will give you a smaller one, which generates the same field, and an integral basis for the field in questions. There is a "Galois"-package which determines the Galois group of a given rational polynomial and which computes Galois conjugates. There is a package which computes everything (so to speak) for quadratic number fields.
Besides all this there exist a lot of standard (and not that standard) functions for converting types, like "trunc", "ceiling", "floor", but also "lift", for lifting an integermod, polymod etc. to an integer, polynomial, etc..
Finally, there are functions like "sum", "product", "divsum", "prodeuler" whose names explain their effect. There is "suminf", "prodinf" and related functions. If you ask for

```
\precision=37\end{verbatim
}
and then for
\begin{verbatim}4*sumalt(n=1,kro(-4,n)/n)
```

you obtain the answer

```
3.141592653589793238462643383279502885
```

(PARI-GP provides of course easier and faster ways for having this number appear on the screen). If you are working under GP you have available several control statements for writing a program, and you have the possibility to define functions yourself.

6 Concluding remarks

The best way to compare PARI to other systems is to try it. If you have a sun work station, Sparc station, MacII or Mac SE30 at your disposal and if you are connected to Internet it should take you not more than half an hour or so to get PARI started. You can pick up the necessary sources (including a TEXcode for a manual) for free using *ftp* at the following address:

```
192.33.148.32 mizar.greco-prog.fr
```

(type *ftp* 192.33.148.32, *login* as "anonymous", *password*: your name). If the french address does not work try

```
128.97.64.16 math.ucla.edu.
```

For the MacIntosh there exist two versions of the GP-calculator, a stand-alone version and a version which requires the MPW-C shell.

7 References

F.Q. Gouvêa: A Look at PARI and GP, to appear in the Notices of the AMS

C. Batut: Thèse Université Bordeaux I, 1990

C. Batut, D. Bernardi, H. Cohen, M. Olivier: User's Guide to PARI-GP, 1991

SAC-2 in ALDES – Ein Werkzeug für die Algorithmenforschung

R. Bündgen, G. Hagel, R. Loos, S. Seitz, G. Simon, R. Stübner, A. Weber
Computeralgebra-Gruppe, Fakultät für Informatik, Universität Tübingen

1 Forschungsprobleme

Die ältere Bezeichnung „Symbolisches Rechnen“ für das, was man heute „Computeralgebra“ nennt, hat in den letzten Jahren neue Aktualität gewonnen: Die formale Behandlung symbolischer Ausdrücke steht im Mittelpunkt von Termersetzungssystemen, automatischen Beweisern und etwa auch des logischen Programmierens. Für den Informatiker ist dabei eine Fülle überraschender Querverbindungen von Interesse. So läßt sich z. B. der Gröbnerbasen-Algorithmus durch den sehr viel allgemeineren Vervollständigungs-Algorithmus nach Knuth-Bendix simulieren. Vollständige Reduktionssysteme lassen effiziente Beweise in Termalgebren, die gleichungsmäßig gegeben sind, zu. Sogar Beweise mit Induktion werden möglich, die herkömmlichen Methoden mit Theorembeweisern an Effizienz weit überlegen sind. Auch Implementierungen konstruktiver Typsysteme bedienen sich Termersetzungssystemen, die in vorläufiger Form auch bei benutzerdefinierten Berechnungsregeln bei vielen Computeralgebra-Systemen ansatzweise vorhanden sind.

Nach einer Periode sehr intensiver Forschung auf dem Gebiet von Polynom-Algorithmen, insbesondere zur Berechnung des größten gemeinsamen Teilers und der Sylvesterschen Resultanten konzentriert sich die Forschung in der Computeralgebra heute hauptsächlich auf drei Gebiete: Gröbnerbasen, Quantorenelimination und Differentialgleichungen.

Sowohl im symbolischen als auch im algebraischen Rechnen bestehen in allen genannten Gebieten, trotz der enorm gestiegenen Rechnerleistungen, inhärente Grenzen im Zeitaufwand der verfügbaren Algorithmen. Deswegen konzentrieren sich die Forschungen in der Computeralgebra auf die Entwicklung besserer zentraler algebraischer Methoden, sowie auf die softwaretechnische Ausnutzung verteilten und parallelen Rechnens.

Als wichtigste Werkzeuge haben sich eine vollständige Dokumentation der Algorithmen, eine aussagekräftige Rechenzeitanalyse und die freie Verfügbarkeit aller Quellen erwiesen. Es ist nicht sehr einfach, die erstaunlichen Rechnerleistungen moderner selbst kleiner Maschinen in Computeralgebra-Systemen einzusetzen. Deswegen erscheint es uns wichtiger, zentrale algebraische Algorithmen in der Lehre zu Allgemeingut zu machen, als sie in kommerzieller Software undokumentiert zu verstecken. Die Informatik, als Wissenschaft von den Algorithmen verdankt viele Einsichten nichttrivialen algebraischen Algorithmen und herkömmlicher mathematischer Praxis und Notation, die bisher nur höchst inadäquat den Weg zum Rechner fanden.

2 Effizientes Symbolisches und Algebraisches Rechnen

2.1 Symbolisches Rechnen

Im Bereich des symbolischen Rechnens ist unser Hauptziel das Beweisen in algebraischen Strukturen mit Hilfe von Termersetzungssystemen (TES). Dabei stehen (induktive) Vervollständigungsverfahren und der Zusammenhang zwischen symbolischem und algebraischem Rechnen im Mittelpunkt. So wurden schnelle Algorithmen zur Analyse der initialen Algebren von TES entwickelt und es konnte gezeigt werden, daß sich der Buchbergeralgorithmus auf Termvervollständigung modulo AC zurückführen läßt. Weitere Interessen gelten der Verbesserung der Termvervollständigung durch Konfluenzkriterien für kritische Paare und starke Termordnungen, sowie der Anwendung von TES zur Typinferenz.

Die erwähnten Verfahren wurden im ReDuX-System ([Bün91]) implementiert. ReDuX bietet eine umfangreiche Sammlung von Algorithmen zum Arbeiten mit TES und eine Reihe von Beweisprogrammen (basierend auf Knuth-Bendix-, Peterson-Stickel- und induktiver Vervollständigung). Das ReDuX-System ist eine Erweiterung des TC-Systems von Küchlin und benutzt die Listen- und Symbolverarbeitung des SAC-2 Systems.

2.2 Algebraisches Rechnen

2.2.1 u-Resultanten

Sei $F_1, F_2, \ldots, F_n$ ein homogenes Polynomsystem F in den Unbestimmten $x_1, x_2, \ldots, x_{n+1}$. Führt man ein lineares Polynom $U = u_1x_1 + u_2x_2 + \ldots + u_{n+1}x_{n+1}$ mit Unbestimmten $u_1, u_2, \ldots, u_{n+1}$ ein und bildet den ggT der Polynome des Resultantensystems von F und U, so erhält man ein homogenes Polynom $R(u_1, u_2, \ldots, u_{n+1})$, die *u-Resultante* von F. Sie zerfällt über dem Körper der komplexen Zahlen vollständig in Linearfaktoren.
Mit Hilfe von u-Resultanten ist man in der Lage, Gleichungssysteme von Polynomen zu lösen. Damit kann man ein einfach-exponentielles Entscheidungsverfahren für die existentielle Theorie der reellen Zahlen angeben ([Ren89]).
Die Algorithmen zur Berechnung von u-Resultanten wurden in ALDES unter Verwendung fast aller SAC-2-Module implementiert.

2.2.2 Praktikable Quantorenelimination

Im Zusammenhang mit der Implementierung des Collins'schen Quantoreneliminationsalgorithmus [Col82] für die reelle Arithmetik (D.S. Arnon, H. Hong, L. Langemyr [Lan90]) treten Probleme auf, die bisher nur ansatzweise als gelöst gelten können. Hierzu gehört eine effiziente Arithmetik algebraischer Erweiterungskörper und die Vereinfachung der voluminösen Formelausgabe. Auch die Frage effizienter Darstellungen für algebraische Zahlen und von Datenstrukturen für zylindrische Zerlegungen hat großen Einfluß auf die Praktikabilität. Besonders wichtig erscheinen Spezialfälle wie Systeme von Polynomgleichungen (u-Resultanten), strikten Polynom-Ungleichungen oder von linearen geordneten Körpern (V. Weispfenning, [Wei88]).
L. Langemyr hat in Fortführung seiner Dissertation in Tübingen zahlreiche neue Algorithmen zur Arithmetik algebraischer Erweiterungen und Mehrfacherweiterungen in SAC-2 implementiert. Darauf basierend kann er Bäume als Datenstruktur für zylindrische Zerlegungen benutzen, was zu einer neuen Implementierung des Collins'schen Algorithmus führte. Außerdem liegt eine Implementierung des Weispfenningschen Algorithmus zur Quantorenelimination in linearen geordneten Körpern vor, bei der jedoch Verbesserungen bei der Formel-Simplifikation wünschenswert sind.
Benutzt werden fast alle Module des SAC-2-Systems, wobei durch die neuen Algorithmen fast eine Verdoppelung der Algorithmenanzahl eintritt.

2.3 Effizienz

Die Rechenzeiten von Computeralgebra-Programmen kontrastieren gelegentlich mit deren qualitativer Leistungsfähigkeit und stellen so auch ein ernstes Problem für die Praktikabilität dar. Deshalb wird der Effizienz besondere Aufmerksamkeit gewidmet. Grundsätzlich existieren zur Leistungssteigerung zwei Strategien: Man kann versuchen, die Komplexität der Algorithmen zu reduzieren oder auf der Basis der Parallelisierung die Implementierung an innovativen Rechensystemen vornehmen. Im letzten Fall sind die kommerzielle Verfügbarkeit der benutzten Rechnersysteme und die konservative Implementierung des Computeralgebra-Systems wichtige Gesichtspunkte, d.h. die Adaption an die neue Rechnerarchitektur darf nur minimale Änderungen des SAC-2–Systems erfordern. Die Portabilität muß weiterhin im Vordergrund stehen.

2.3.1 Ein iterativer Karatsuba-Algorithmus

In Computeralgebra–Systemen spielt die Effizienz der Langzahlarithmetik eine wesentliche Rolle. Man muß besonders auf die Implementierung der zentralen Algorithmen des Systems achten, die am häufigsten zum Einsatz kommen. Mit zunehmender Länge der zu multiplizierenden Zahlen haben spezielle Multiplikations-Algorithmen, wie z.B. der von *Karatsuba* oder *Schönhage-Strassen* Vorteile gegenüber der gewöhnlichen Multiplikation. Um die Effizienz der Langzahlarithmetik des SAC-2–Systems zu erhöhen, wurde ein iteratives Karatsuba-Verfahren entwickelt, bei dem die Ziffernfolgen der zu multiplizierenden Zahlen und die Zwischenergebnisse nach einem bestimmten Schema in den Speicher abgelegt werden. Die ausgewählte Anordnung der Daten ermöglicht eine effiziente Implementierung des Algorithmus in Assembler. Der so erzielte Assembler-Modul wurde in das SAC-2–System integriert. Die empirischen Resultate zeigen die Vorteile dieses Verfahrens im Vergleich zur gewöhnlichen Multiplikation und zum rekursiven Karatsuba-Algorithmus.

2.3.2 Verteiltes Rechnen

Die Sprache ALDES wurde als formales Werkzeug zur Beschreibung von Algorithmen entworfen. Für ihre Funktion als Programmiersprache ist ein Konzept zum verteilten Rechnen entwickelt und implementiert worden, dessen Komponenten bis auf einen systemspezifischen Teil in ALDES implementiert wurden ([Sei90]). Das Modell ist eine spezielle Anwendung der dazu entwickelten Erweiterung von AL-

DES um Konzepte des Nachrichtenaustauschs (message passing) in einem lokalen Netz. Der Programmierer veranlaßt analog zum Aufruf von Prozeduren die Erzeugung von *Aufträgen* zur Laufzeit, die aus einem Algorithmennamen und Argumenten bestehen. Die Aufträge werden auf einem von mehreren *Algorithmenservern* ausgeführt, die auf Maschinen mit möglicherweise unterschiedlicher Architektur an einem lokalen Netz laufen. Bis zur Verfügbarkeit der Ergebnisse arbeitet die den Auftrag erzeugende Prozedur mit *Repräsentanten* der erwarteten Werte. Das Modell ist besonders für Algorithmen geeignet, die Resultate nach der modularen Methode berechnen.
Die Implementierung des Modells erfolgte auf der Basis der Kernmodule (Basis-, Listen- und Symbol-System) von SAC-2. Die Systemabhängigkeit der Implementierung beschränkt sich auf die Ein/Ausgabe-Schnittstelle eines ALDES-Programms mit dem Betriebssystem. Zur Kommunikation in einem lokalen Netz wurde die Bedeutung der Units zur Ein- und Ausgabe erweitert und das in vielen UNIX Implementierungen vorhandene Konzept der *Sockets* benutzt.

2.3.3 Vektorielles Rechnen

Die im Hochschulbereich verfügbaren Supercomputer[1] legen eine vektorielle SAC-2–Implementierung nahe. Um den Vektorprozessor effizient einzusetzen, benötigt man statt der gestreuten Speicherung, die aus der üblichen Listenrepräsentation resultiert, „dicht im Speicher liegende“ Daten ([Stü89]). Das wird durch einen Kernmodul *vp* erreicht, der funktional dem Listenverarbeitungsmodul entspricht und dem SAC-2–System angefügt wird, ohne die übrige Funktionalität einzuschränken. Für die Konvertierung Liste–Vektor stehen Konvertierungsroutinen zur Verfügung. Ansonsten werden, wegen des modularen Aufbaus von SAC-2, die Algorithmen höherer Module nicht beeinträchtigt. Auf der Grundlage vektoriellen Datenstruktur können beispielsweise Algorithmen der Langzahlarithmetik und der modularen Arithmetik effizient implementiert werden.

2.4 Sprache und Implementierung

2.4.1 ALDES – die Implementierungsprache von SAC-2

Das SAC-2–System ist in der Sprache ALDES geschrieben. ALDES wurde in der 70-er Jahren speziell für SAC-2 entwickelt, wobei man bei dem Entwurf gewisse Randbedingungen des schon teilweise vorhandenen Systems beachten mußte. ALDES ist mit Listenverarbeitung und dynamischer Speicherverwaltung ausgestattet und hat eine effiziente Langzahlarithmetik. Die zahlreichen Implementierungen von SAC-2 sind durch unterschiedliche Zielsetzungen motiviert. Die zwei maßgebenden Überlegungen waren jedoch Effizienz und Portabilität. Dabei wurde ALDES nach Assembler (IBM 370, Siemens 7000) oder aus Portabilitätsgründen nach ANSI-FORTRAN übersetzt. Mittlerweile existieren weitere SAC-2 Implementierungen, die LISP, Modula-2 oder C benützten. Um zwischen Effizienz und Portabilität einen vernünftigen Kompromiß zu finden, haben wir einen sogenannten „retargetable“ ALDES Übersetzer entwickelt ([Sim91]). Als Entwicklungswerkzeug wurde der *Amsterdam Compiler Kit* verwendet. Die Architektur der Zielmaschine wird durch Tabellen beschrieben und der gewünschte Übersetzer anhand dieser Tabellen mit Hilfe von Programmgeneratoren erzeugt. Das Front-End ist für alle Übersetzer der gleichen Sprache gleich. Der so generierte Übersetzer verfügt über lokale und globale Optimierungsmöglichkeiten, erzeugt kompakten, effizienten Code und hat kurze Übersetzungszeiten.

2.4.2 Typproblem

Ein umfangreiches Computeralgebra-System wie SAC-2 umfaßt zahlreiche Algorithmen für viele unterschiedliche Datentypen, wie z. B. algebraische Zahlen, Matrizen, Polynome über verschiedenen Zahlbereichen oder auch prädikatenlogische Formeln über verschiedenen Sprachen 1. Stufe.
Eine Benutzersprache für SAC-2 existiert zur Zeit nicht. Ziel wäre es, die Flexibilität traditioneller mathematischer Notation möglichst getreu zu realisieren, eventuell auch zweidimensional. Hierzu müssen einige Probleme als derzeit ungelöst eingestuft werden. Ein zentrales Problem ist es beispielsweise, ein geeignetes Typensystem zu finden. Aus der Sicht des Benutzers ist es wünschenswert, daß das System den Typ der Daten weitgehend selbständig rekonstruieren kann. Um aber umfangreiche Berechnungen weiterhin effizient durchführen zu können, sollte diese *Typinferenz* bereits zur Compilierzeit erfolgen und nicht erst von einem Interpreter vorgenommen werden.
Während beim Entwurf neuer funktionaler Programmiersprachen, wie z. B. **ML**, beim Problem der Typinferenz in den letzten Jahren große Fortschritte erzielt wurden, sind bei den bestehenden Computeralgebra-Systemen Mechanismen zur Typinferenz während der Compilierzeit kaum vorhanden.
In unserer Arbeitsgruppe wird an Problemen, die in

[1] Diese SAC-2–Implementierung erfolgte an der VP-S600/20 des Rechenzentrums der Universität Karlsruhe, mit dem wir über BELWÜ verbunden sind.

diesem Zusammenhang auftauchen, gearbeitet, insbesondere mit dem Ziel einer flexiblen Benutzersprache.

3 SAC-2–Module

3.1 Algebraische Module

Vereinfachend lassen sich alle von G. E. Collins entwickelten Computeralgebra-Systeme im Zusammenhang des einen großen Problems sehen, zu einer effizienten Implementierung Tarskis elementarer Algebra zu gelangen. Das SAC-1-System (Symbolic and Algebraic Computation, Version 1) implementierte auf der Basis einer Langzahlarithmetik und effizienter Arithmetik in endlichen Körpern zahlreiche Polynomalgorithmen, die Forschungsergebnisse von 1965–1975 realisierten: reduzierter und Subresultanten-Polynom-ggT-Algorithmus, Berlekamp-Hensel-Algorithmus zur Faktorisierung ganzzahliger Polynome, Bareiss-Algorithmus für lineare Systeme.
Sie finden sich im SAC-2 Polynom-ggT- und Resultanten-Modul, im Polynom-Faktorisierungs-Modul und Lineare-Algebra-Modul implementiert. Untermodule behandeln kombinatorische Algorithmen, Arithmetik ganzer, rationaler, modularer und reell-algebraischer Zahlen. Letztere werden nach einem modifizierten Verfahren von J. V. Uspensky durch Isolation reeller Nullstellen erzeugt. In diesen Modulen sind viele Doktorarbeiten von Collins Arbeitsgruppe realisiert, die weitgehend von ihm selbst, z. T. von SAC-1 aus, implementiert und dokumentiert wurden.

3.2 Kernmodule

Aus Gründen der Portabilität ist die Kommunikation zwischen einem ALDES Programm und dem Betriebssystem im SAC-2–System auf drei systemabhängige Prozeduren zum Lesen und Schreiben von Datensätzen und zum Lesen der Systemzeit beschränkt.
Das *Basis System* ist eine Sammlung einfacher Algorithmen, welche die Ein- und Ausgabe elementarer Objekte wie Zeichen und einfach genauer ganzer Zahlen unterstützt. Dieses System bildet die Grundlage für alle höheren Systeme.
Das *Listen System* stellt den Datentyp „Liste“ sowie Algorithmen zur Manipulation von Objekten zur Verfügung, welche durch Listen implementiert werden (z.B. Mengen). Die Verwaltung der Listenzellen und die Freispeichersammlung ist für den Programmierer transparent.
Im *Symbol System* sind Algorithmen zum Umgang mit symbolischen Bezeichnern implementiert. Die Übersetzeroption der Ablaufverfolgung von ALDES Programmen ist mit Hilfe dieser Algorithmen implementiert.

3.3 Software-Werkzeuge

Die Erstellung und Publikation von ALDES-Programmen wird u.a. durch graphische Schnittstellen zu SUNVIEW und dem X Window System, sowie durch „Pretty Print“ (LaTeX) unterstützt.

4 Erhältlichkeit

Das SAC-2–System wird nicht kommerziell vertrieben. Es ist für wissenschaftliche Zwecke innerhalb Europas erhältlich bei: Prof. Dr. R. Loos, Universität Tübingen, Fakultät für Informatik, Sand 13, W-7400 Tübingen.
Viele der erwähnten Algorithmen sind mit Laufzeitanalyse in der Literatur veröffentlicht.

[ALDES] *ALDES–Report, revised version.* Wilhelm-Schickard-Institut für Informatik, Tübingen.

[Bün91] Reinhard Bündgen. *Term Completion Versus Algebraic Completion.* Dissertation, Universität Tübingen, D-7400 Tübingen, Mai 1991.

[Col82] George Collins. Quantifier elimination for real closed fields: a guide to the literature. In Bruno Buchberger, George Collins, and Rüdiger Loos, Hrsg., *Computer Algebra*, Springer-Verlag, 1982.

[Lan90] Lars Langemyr. *An Implementation of the Cylindrical Algebraic Decomposition Algorithm with Multiple Algebraic Extensions.* Interner Bericht 90–13, Wilhelm-Schickard-Institut für Informatik, Tübingen, 1990.

[Ren89] J. Renegar. *On the Computational Complexity and Geometry of the First-Order Theory of the Reals, Part I, II, III.* Technical Reports 853, 854, 856, Cornell University, Ithaka, NY, 1989.

[Sei90] Steffen Seitz. *Verteiltes Rechnen in SAC-2.* Interner Bericht 90–09, Wilhelm-Schickard-Institut für Informatik, Tübingen, 1990.

[Sim91] Gábor Simon. *A Retargetable ALDES Compiler.* PhD thesis, Universität Tübingen, D-7400 Tübingen, Germany, 1991. (eingereicht).

[Stü89] Rudolf Stübner. *Vektorrechner, Funktionsweise, Perspektive und ihre Anwendung im Bereich der Computer Algebra.* Interner Bericht 89–07, Wilhelm-Schickard-Institut für Informatik, Tübingen, 1989.

[Wei88] Volker Weispfenning. The complexity of linear problems in fields. *Journal of Symbolic Computation*, 5:3–27, 1988.

GAP
Groups, Algorithms and Programming

Martin Schönert, RWTH Aachen

GAP ist ein System für algorithmische Gruppentheorie, welches seit 1986 am Lehrstuhl D für Mathematik der RWTH Aachen entwickelt wird.
GAP besteht aus drei Teilen, der Programmiersprache, der Library von Funktionen sowie der Programmierumgebung.
Die Programmiersprache wurde speziell entwickelt, um das Schreiben von Programmen für die Gruppentheorie zu erleichtern. Dazu bietet sie spezielle Datentypen wie z.B. endliche Körper, Matrizen und Permutationen.
Die Library ist eine umfangreiche Sammlung von Funktionen, welche viele der bekannten Algorithmen implementieren.
Die Programmierumgebung hilft beim Entwickeln von Algorithmen, bei der Fehlersuche sowie beim Auffinden der zeitkritischen Teile eines Algorithmus.
Zu GAP gibt es eine Library von Gruppendaten, in der z.B. alle transitiven Gruppen vom Grad kleiner 13, alle Gruppen, deren Ordnung 256 teilt, sowie die Charaktertafeln aller Gruppen des ATLAS verfügbar sein werden.
GAP wird gegen Erstattung der Sachkosten abgegeben. In Zukunft wird es auch elektronisch per 'ftp' frei zu erhalten sein. Dieser Artikel beschreibt die Version 3.0, welche zum Ende dieses Jahres verfügbar sein sollte.

1 Die Sprache

Natürlich gibt es *ganze Zahlen* sowie *rationale Zahlen*. Wie gewöhnlich in algebraischen Programmsystemen sind die ganzen Zahlen in ihrer Größe nicht beschränkt.

```
gap> Factorial( 32 );
263130836933693530167218012160000000
gap> 1/2 + 2/3 + 3/4 + 4/5;
163/60
```

Ungewohnt ist dagegen die Unterstützung von **Kreisteilungskörpern**, also $Q(e^{2\pi i/n})$. Für $e^{2\pi i/n}$ schreibt GAP 'E(¡n¿)' und stellt die Elemente eines solchen Körpers als Polynom darin dar, wobei automatisch in den kleinstmöglichen Körper konvertiert wird.

```
gap> E(24) + 1;
E(24)-E(24)^8-E(24)^16
gap> last * (E(24) - 1);
E(12)^4-E(12)^7+E(12)^8
gap> last+E(9)^2-E(9)^3;
E(36)^8-E(36)^21+E(36)^24
```

Auch alle **endlichen Körper** bis zur Ordnung 2^{16} werden unterstützt. Dem Element `E`(n) oben entspricht hier `Z`(n), das ein Erzeuger der multiplikativen Gruppe von $GF(n)$ ist. Die Elemente des Körpers werden als Potenzen davon dargestellt. Auch hier wird automatisch in den kleinsten Körper konvertiert.

```
gap> GF(16).root^2 + 1;
Z(2^4)^8
gap> last * Z(16)^2;
Z(2^2)^2
gap> last * (Z(8)+1);
Z(2^6)^6
```

Als ersten Typ von Gruppenelementen gibt es in GAP **Permutationen** also bijektive Abbildungen einer endlichen Menge, welche in GAP immer die Form $\{1..n\}$ hat. Permutationen werden in Zykelschreibweise geschrieben. Prinzipiell werden Gruppenelemente mit dem * Operator verknüpft. Der ^ Operator angewendet auf ein Gruppenelement und eine ganze Zahl ergibt immer die entsprechende Potenz des Elements. Ist der rechte Operand eine negative Zahl, so wird das Gruppenelement natürlich invertiert. Der ^ Operator angewendet auf eine Zahl und eine Permutation ergibt das Bild der Zahl unter der Permutation.

```
gap> p := (1,2)(3,4,5);
(1,2)(3,4,5)
gap> (2,3,4) * p;
(1,2,4)(3,5)
gap> 2 ^ last;
4
```

Matrizen können in GAP über jedem der oben genannten Körper definiert werden. Man beachte, daß Matrizen per Multiplikation von rechts auf Vektoren operieren, auch diese Operation kann mit ^ geschrieben werden.

```
gap> [ [ Z(4), Z(4)^0 ],
       [ Z(4)^2, Z(4)^0 ] ] ^ 5;
[ [ Z(2)^0, 0 ],
  [ 0, Z(2)^0 ] ]
gap> [ [ 1, 2 ],
       [ 2, 1 ] ] ^ -1;
[ [ -1/3, 2/3 ],
  [ 2/3, -1/3 ] ]
gap> [ 1, 1 ] ^ last;
[ 1/3, 1/3 ]
```

Schließlich gibt es noch **Worte** in abstrakten Erzeugenden. Die Multiplikation ist für Worte in abstrakten Erzeugern einfach durch Hintereinanderschreiben mit Kürzen definiert.

```
gap> a := Word("a");
a
gap> a ^ Word("b");
b^-1 * a * b
gap> Comm(a,b) / last;
a^-1
```

In dem Beispiel oben sieht man noch eine weitere Definition des ^ Operators: sind beide Operanden Elemente aus derselben Gruppe, so ist das Ergebnis das mit dem rechten Element Konjugierte des linken Elementes.
Worte in auflösbaren Gruppen benutzen, daß es für jede auflösbare Gruppe eine Präsentation gibt, welche es ermöglicht, jedes Wort in den Erzeugern in eine eindeutige Normalform zu bringen.

Natürlich gibt es auch Datentypen die aus anderen Sprachen gut bekannt sind. Der wichtigste ist die **Liste**. Um Elemente zu einer Liste zusammenzufassen, schreibt man sie einfach durch Kommas getrennt in eckige Klammern. Auf die Elemente einer solchen Liste kann man dann mit *liste*[*index*] zugreifen.

```
gap> l := [5,2,3,11,2,7];
[ 5, 2, 3, 11, 2, 7 ]
gap> l[4];
11
gap> l[10] := [13];;  l;
[ 5, 2, 3, 11, 2, 7,,,, [ 13 ] ]
```

Am letzten Beispiel sieht man, daß Listen automatisch erweitert werden, Löcher enthalten können sowie Elemente unterschiedlicher Typen enthalten können.

Mengen werden in GAP auch durch Listen dargestellt. Damit wir eine Liste aber Menge nennen, darf sie keine Löcher haben, sie muß sortiert sein, und sie darf kein Element mehrfach enthalten.

```
gap> s := Set(l);
[ 2, 3, 5, 7, 11, [ 13 ] ]
gap> 4 in s;
false
gap> Union( s, [4,9,8] );
[ 2, 3, 4, 5, 7, 8, 9, 11, [ 13 ] ]
```

Wer findet, daß Listen den weiter oben eingeführten Vektoren und Matrizen sehr ähnlich sehen, hat ganz recht. In der Tat sind **Vektoren** in GAP nichts anderes als Listen, von denen wir fordern, daß sie keine Löcher enthalten und daß alle Elemente in einem gemeinsamen Körper liegen. **Matrizen** sind dann einfach Listen von Vektoren.

```
gap> v := [1,-1,1,-1];
[ 1, -1, 1, -1 ]
gap> v + [1/2,0,0,0];
[ 3/2, -1, 1, -1 ]
gap> v * v;
4
```

Eine Liste, die nur aufeinanderfolgende ganze Zahlen enthält, nennen wir einen **Range**. Für Ranges gibt es die Kurzschreibweise [*Anfang* .. *Ende*]. Anhand von Ranges kann man sehen, daß obige Einteilung nicht ausschließlich ist. Ein Range ist gleichzeitig eine Liste, eine Menge und ein Vektor.

```
gap> r := [2..5];
[ 2 .. 5 ]
gap> IsSubset( s, r );
false
gap> r + v;
[ 3, 2, 5, 4 ]
```

Records dienen ebenfalls dazu, mehrere Objekte zu einem gemeinsamen Objekt zusammenzufassen. Man kann auf die Komponenten eines Records mit *Record.name* zugreifen.

```
gap> r := Gcdex( 123, 66 );
rec( gcd := 3,
  coeff1:=7,   coeff2:=-13,
  coeff3:=-22, coeff4:=41 )
gap> r.coeff3;
-22
```

Records werden in GAP hauptsächlich verwendet, um strukturierte Mengen wie z.B. Gruppen, Körper und Vektorräume darzustellen, wie im folgenden Beispiel die Mathieu–Gruppe auf 11 Punkten:

```
gap> G := M(11);
rec(
  identity   := (),
  degree     := 11,
  generators := [
    (1,2,3,4,5,6,7,8,9,10,11),
    (3,7,11,8)(4,10,5,6) ] );
```

Die Programmiersprache selber ist PASCAL ähnlich. Das folgende Beispiel sollte ohne Probleme zu verstehen sein.

```
  gap> Orbit := function ( G, pnt )
 local  orb, p, g;
 orb := [ pnt ];
 for p  in orb  do
   for g  in G.generators  do
     if not p^g in orb  then
       Add( orb, p^g );
     fi;
   od;
 od;
 return orb;
end;
```

Die Programmiersprache wird vom GAP–Kern interpretiert. Sie ist allerdings so geplant, daß eine zukünftige Version, um höhere Geschwindigkeit zu bieten, auch einen Compiler haben könnte.

Speicherverwaltung geschieht automatisch. Der Programmierer muß also für seine Objekte nicht selber Speicher bereitstellen oder diesen nach Gebrauch der Objekte selber wieder frei machen. Dies geschieht automatisch durch eine Garbage Collection.

Variablen haben in GAP keinen statischen Typ wie etwa in PASCAL. So kann man obige Funktion verwenden, um die Bahn eines Punktes unter einer Permutationsgruppe, die Bahn eines Vektors unter einer Matrixgruppe oder die Konjugiertenklasse eines Elementes unter einer beliebigen Gruppe zu berechnen.

2 Die Library

Die Library ist der inzwischen größte Teil von GAP. Sie enthält eine große Zahl von Funktionen, welche der Benutzer entweder verwenden kann, um direkt sein gegebenes Problem zu lösen, oder als Bausteine, um mit diesen kompliziertere Programme zu schreiben.

Die Library läßt sich untergliedern in generelle Funktionen für die eingebauten Datentypen, Funktionen für beliebige Gruppen, Permutationsgruppen, auflösbare Gruppen und endlich präsentierte Gruppen sowie zum Umgang mit Charakteren.

Einige Beispielfunkionen für die eingebauten Datentypen mögen hier genügen.

```
gap> IsPrime( 2^61-1 );
false
gap> Root( 2*10^60 );
1414213562373095048801688724209
gap> Position( Primes, 541 );
100
gap> Sum( [1..10], i->i^2 );
385
gap> Determinant( [[1,2],[2,1]] );
-3
gap> m := [[1,2,3],[4,5,6],[7,8,9]];;
gap> Nullspace( m );
[ [ -1, 2, -1 ] ]
gap> ElementaryDivisors( m );
[ 1, 3, 0 ]
gap> ProductPol( [1,-2,1], [1,1] );
[ 1, -1, -1, 1 ]
gap> FactorsPol( Z(2)*[0,1,0,0,1] );
[ [0,Z(2)^0], [Z(2)^0,Z(2)^0],
  [Z(2)^0,Z(2)^0,Z(2)^0] ]
```

Der wichtigste Teil der Library sind natürlich die **gruppentheoretischen Funktionen**. Da Beispiele mit diesen Funktionen jeweils ziemlich umfangreich wären, begnüge ich mich hier damit, nur einige zu nennen.

Für alle Gruppen, auch solche die aus vom Benutzer selbst definierten Elementen bestehen, stehen Funktionen wie `DirectProduct`, `Elements`, `CommutatorSubgroup`, etc. bereit.

Auch für kompliziertere Aufgaben, z.B. zur Berechnung des Untergruppenverbandes oder der Burnside'schen Markentafel stehen (teilweise interaktive) Funktionen zur Verfügung.

Wann immer eine Gruppe auf einer endlichen Menge **operiert**, kann man die Funktionen `Orbit`, `Blocks`, `Operation`, `Stabilizer`, etc. verwenden.

Für **Permutationsgruppen** gibt es unter anderem die Funktionen `Base`, `Stronggens`, `Centralizer`, `Normalizer`. Dabei können diese auch für sehr große Gruppen eingesetzt werden, da ihre Laufzeiten hauptsächlich vom Grad und nicht von der Ordnung abhängen.

Für **auflösbare Gruppen**, die durch eine geeignete Präsentation gegeben sind, gibt es `Centralizer`, `ConjugacyClasses`, `HallSubgroups` und vieles (170 Funktionen) mehr. Auch dabei können die beteiligten Gruppen sehr groß werden, da die Laufzeit im allgemeinen nur von der Länge einer Kompositionsreihe abhängt.

Für **endlich präsentierte Gruppen** gibt es z.B. die Funktionen: `OperationCosets`, `Index`, `PresentationSubgroup`.

Schließlich gibt es noch eine Fülle von Funktionen, mit denen man mit **Charaktertafeln** endlicher Gruppen rechnen kann, z.B. Funktionen zum Bestimmen aller Tafelautomorphismen, zum Untersuchen von Fusionen, zum Berechnen von Skalarprodukten, zum Symmetrisieren und zum Suchen nach Permutationscharakteren.

3 Das Environment

GAP besitzt eine interaktive Programmierumgebung. Wie in allen Beispielen oben zu sehen, fordert GAP den Benutzer mit einem Prompt `gap>` zu einer Eingabe auf, liest dann den Befehl, führt ihn aus und druckt das Ergebnis dieser Berechnung.
Es gibt eine umfangreiche **On-line Help**. In dieser sind alle verfügbaren Funktionen sowie die Sprache und weitere Möglichkeiten von GAP beschrieben. Sie dient zugleich als Basis für das **Handbuch**.
Die Umgebung unterstützt den Benutzer bei der **Fehlersuche**. Tritt ein Fehler in einer Berechnung auf, so wird ein sogenannter **Break loop** betreten. In diesem kann sich der Benutzer die Werte aller Variablen anschauen, verändern und dann entweder die Berechnung fortsetzen oder in die Hauptschleife zurückkehren.
Letztlich gibt es die Möglichkeit, sich eine genaue Aufstellung darüber geben zu lassen, wo bei einer Berechnung die Zeit verbraucht wird. Damit kann man die zeitkritischen Stellen seines Programmes finden und gegebenenfalls verbessern.

CoCoA: A User-Friendly System for doing Computations in Commutative Algebra

Alessandro Giovini & Gianfranco Niesi

CoCoA is a small special-purpose system for doing **Co**mputations in **Co**mmutative **A**lgebra, which runs on any computer of the Macintosh family provided with at least 1Mb of RAM (and now also on IBM/PC or compatibles with at least 512K of RAM).

Several motivations led us to the development of an entirely new system for symbolic computations. We noticed that mathematicians working in the fields of commutative algebra and algebraic geometry, and which do not have much experience of computers, do not use very frequently (or effectively) existing general purpose powerful systems; in out opinion the main reason is that these packages run essentially on microcomputers or mainframes and their use implies an account on a machine and some knowledge of its operating system; very often instead one needs a software system for sophisticated but reasonably simple tasks for which a microcomputer (not to say a mainframe) is really oversized; we noticed instead that many mathematicians use frequently a computer of the Macintosh family due to its very cleverly designed user interface. Finally, we felt the need of a system entirely developed within our research group to test and check algorithms and theoretical ideas.

In late 1987 we started to work on the prototype of a small system capable of handling multivariate polynomial rings, embodying algorithms which use the most recent algebraic techniques and completely integrated within the Macintosh environment. At the beginning of 1989 a first version of the system was ready to be given to -external users-; since then the system has been presented, among other places, at the COCOA II meeting (Genova, May 29—June 3, 1989), at the Computers & Mathematics 1989 conference (MIT, Cambridge, Massachusetts, June 13—17, 1989), at the DISCO 90 conference (Capri, Italy, April 10—12, 1990) and, the last release (1.5), at the Meeting on Computational Algebraic Geometry and Commutative Algebra (Cortona, Italy, June 17—21, 1991).

The main characteristics of CoCoA can be in our opinion so summarized:

- it is a small system, and we make any effort in keeping it small; it runs on a machine one can have on the desktop, and even at home, but takes advantage of additional hardware provided;
- it uses consistently the classical Macintosh user interface and it is completely integrated within that environment;
- it gives the user great freedom in organizing the computations and in writing commands and expressions in a very intuitive way while being very forgiving w.r.t. errors;
- it implements advanced algorithms and strategies without supposing the user knowing them;
- and, yes!, it is reasonably fast.

Presently CoCoA is being used in several research centres in Canada, England, France, Germany, Holland, Italy, Japan, Nigeria, Spain, Sweden and USA.

CoCoA is capable of performing both simple and sophisticated operations on multivariate polynomial rings and on various data connected with them (essentially rational functions, ideals, modules and matrices); polynomials may have coefficients either in the field $\mathbf{Q}$ of rational numbers[1] or in the residue ring $\mathbf{Z}_p$.

Every operation is performed within a current ring, which is determined by the ring name, the field of coefficients, the names (and weights) of the variables and the term-ordering. The default ring is $\mathbf{Q} : t, x, y, z$ with all weights set to 1 and with the degree reverse lexicographic term-ordering. At any time the user can easily modify this ring by just pulling down a menu and editing some values.

The advanced user has also the possibility of changing the values of some parameters affecting the way in which some specialized algorithms work. The values to which the system defaults should however suffice in most cases for an optimum use of the system.

The user can open standard text-editing Macintosh windows in which data can be entered in a very simple

[1] Currently, a proper subset of $\mathbf{Q}$, since both numerators and denominators must not exceed $2^{32} - 1$

format. Several kinds of computations can then be performed on the entered data and the results can be assigned to variables for later use. If the user modifies the ring, then the already entered or computed data can be easily transferred to the new ring (whenever the operation makes sense).
The system is capable of performing basic operations on the data, as for example the following ones:

- sums, products, powers, derivatives, gcd, lcm of polynomials,
- sums, products, powers, derivatives of rational functions,
- sums, products, powers of ideals,
- sums of modules,
- sums, products, powers, determinants of matrices,
- operations between heterogeneous values, like the product of a polynomial and an ideal, etc.

and more advanced operations like the following ones:

- Gröbner bases of ideals and modules;
- the normal form of a polynomial w.r.t. an ideal;
- syzygies of ideals and modules;
- standard bases and tangent cones of ideals.
- elimination and substitution of variables;
- homogeneization of ideals;
- intersection and division of ideals and modules;
- inclusion or equality test for ideal and modules;
- minimalization of a system of generators of an ideal or a module;
- minimal free resolutions of ideals and modules;
- Poincaré series and Hilbert function of an ideal;
- dimension and multiplicity of an ideal;
- computing particular classes of ideals (for example, the ideals of the minors of a matrix, or the ideal of a finite set of points of the projective space).
- resultant of two polynomials;

The syntax for the expressions that the system can evaluate has been chosen to be as close as possible to the usual mathematical notation; the system displays the exponents as superscripts and the indexes as subscripts, taking advantage of the graphics capabilities of the Macintosh. For example, to eliminate the indeterminate t from the ideal generated by the three polynomials $t^{31} + t^6 - x$, $t^8 - y$ and $t^{10} - z$ one has simply to evaluate the following expression (whose result is a value of type ideal):

$$\mathrm{Elim}(t, \mathrm{Ideal}(t^{31} + t^6 - x, t^8 - y, t^{10} - z)).$$

The core of the system is an implementation of Buchberger's algorithm for computing the Gröbner basis of an ideal; the algorithm (and an adaptation to the case of modules) has been optimized in several senses and it is used as a building block for some of the operations that the system is capable of doing; for most uses the user can however completely ignore the theory of Gröbner bases and even their existence: the system will do all the necessary Gröbner stuff in the background. However, for an optimum use of the system (and of some system parameters) some knowledge of the theory may be useful.
Requests for the current release of the system (1.5) and suggestions can be sent directly to the authors or to the following email address:

`cocoa@igecuniv.bitnet` .

Das Algorithmen-System SIMATH in der Zahlentheorie

Ursula Schneiders, Pascale Serf

SIMATH ist ein Computer-Algebra-System mit Schwerpunkt in der algebraischen Zahlentheorie. Es basiert auf der Sprache C und besteht aus Algorithmenpaketen für Listenverwaltung und -bearbeitung, für die Arithmetik über $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{Z}/m\mathbb{Z}$, endlichen Körpern und globalen Körpern, für das Rechnen mit Polynomen bzw. Matrizen und Vektoren über diesen Strukturen sowie für die Theorie der elliptischen Kurven. Zusätzlich ist das System mit einem Calculator namens simcalc ausgestattet, der die meisten Funktionen im Dialog zur Verfügung stellt.

SIMATH, was für SInix MATHematik steht, wurde ursprünglich im Rahmen eines Kooperationsvertrages zwischen der Universität des Saarlandes und der Siemens AG auf dem Siemens PC MX2 entwickelt. Inzwischen ist es auch für den PC MX2+ sowie für SUN3- und SUN4-Workstations und für Apollo--Workstations DN3000 und DN4500 verfügbar.
Da das System vollständig in C geschrieben ist, bereitet seine Anwendung keine Schwierigkeiten, denn der Benutzer kann einfach die SIMATH-Funktionen in seinen C-Programmen aufrufen. Konvertierungsroutinen machen es zusätzlich möglich, die SIMATH-Funktionen in Fortran-Programmen zu benutzen.
SIMATH ist als offenes System konzipiert, d.h. dem Benutzer stehen auch die Quellbibliotheken zur Verfügung, so daß er leicht allgemeine Algorithmen seinen speziellen Bedürfnissen anpassen kann bzw. eigene Algorithmen an jeder Stelle des Systems hinzufügen kann.
Die Verbindung zwischen SIMATH und dem Betriebssystem SINIX bzw. UNIX wird durch eine Benutzeroberfläche hergestellt. Diese erleichtert die Verwaltung von Quell- und Modulbibliotheken, Compilieren, Drucken usw. Sie bietet aber z.B. auch die Möglichkeit, alle Funktionen zu einem bestimmten Schlüsselwort auflisten zu lassen oder die Dokumentation jeder beliebigen SIMATH-Funktion auf dem Bildschirm ausgeben zu lassen.
Die Graphik am Ende des Artikels verdeutlicht den Systemaufbau.

1 Basissystem

Das Basissystem besteht hauptsächlich aus Listenfunktionen, d.h. Funktionen zur

- Listenverwaltung
- zum Sortieren
- zum Suchen,

enthält aber auch Ein- und Ausgabeprogramme sowie Funktionen zur Systemverwaltung und -initialisierung. Die Systemverwaltung liegt allerdings unter der Benutzeroberfläche, so daß der Anwender damit keine Probleme hat. Z.B. wird der dynamisch aufgeteilte Speicherplatz von einem automatischen garbage-collector verwaltet.

2 Arithmetiksystem

Das Arithmetiksystem umfaßt zur Zeit ca. 370 Funktionen für $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{Z}/m\mathbb{Z}$ und $\mathbb{F}_q$, wobei m eine natürliche Zahl ist und $\mathbb{F}_q$ ein endlicher Körper mit q Elementen, sowie für algebraische Zahl- und Funktionenkörper und für elliptische Kurven über $\mathbb{Q}$ und algebraischen Zahlkörpern.
Es stehen sowohl Programme zur Grundarithmetik als auch höhere Algorithmen zur Verfügung wie z.B.

- Bestimmung von ggT und kgV
- Chinesischer Restsatz
- Primzahltest
- Faktorisierung

} in $\mathbb{Z}$,

- Diskriminantenberechnung
- Norm und Spur
- Zerlegungsgesetz
- Ganzheitsbasis

} für Zahlkörper,

- Führer
- Reduktionstyp
- lokal/global min. Modell

} für ell. Kurven.

3 Polynomsystem

Polynome in SIMATH können beliebig viele Variablen haben und Koeffizienten aus $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{Z}/m\mathbb{Z}$, $\mathbb{F}_q$ und algebraischen Zahlkörpern.
Für die interne Darstellung kann man zwischen sparse und dense wählen.
Das Polynomsystem enthält etwa 390 Programme

- zur Standardarithmetik
- zum Chinesischen Restsatz
- zum Ableiten und Integrieren
- zum Auswerten und Substituieren
- zur Berechnung von Resultanten und Diskriminanten
- zur Faktorisierung.

4 Matrizen-/Vektorsystem

Matrizen und Vektoren sind über $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{Z}/m\mathbb{Z}$ und $\mathbb{F}_q$ sowie über Polynomringen und rationalen Funktionenkörpern zugelassen. Das Matrizen-Vektor-Paket, das aus ca. 320 Funktionen besteht, ist mittlerweile fast vollständig. Neben der Standardarithmetik umfaßt es z.B.

- Linearkombination und Skalarprodukt für Vektoren
- für Matrizen:
  - Determinantenberechnung
  - Inversenbildung
  - Hermite-Normalform
  - Elementarteilerform
  - charakteristisches Polynom
  - Eigenwerte über $\mathbb{Z}$, $\mathbb{Q}$ und $\mathbb{Z}/m\mathbb{Z}$
- Lösen von Gleichungssystemen.

5 Calculator

Der Calculator simcalc arbeitet interaktiv. Er stellt die meisten SIMATH-Funktionen für Zahlen (ganze Zahlen, rationale Zahlen, Elemente aus $\mathbb{Z}/m\mathbb{Z}$), Polynome, Matrizen und Vektoren sowie für elliptische Kurven zur Verfügung. Besonders geeignet ist er für Benutzer mit wenig Programmiererfahrung.
Die zu berechnenden Ausdrücke gibt man so ein, wie man sie auf dem Papier aufschreiben würde. Man kann z.B.

- Ergebnisse Variablen zuweisen
- mit Variablen wie mit Zahlen rechnen
- Ausdrücke beliebig verschachteln
- Variablen in Polynomen, Matrizen und Vektoren substituieren (sogar rekursiv)
- Daten aus Dateien einlesen
- ein Protokoll auf Datei mitlaufen lassen
- Rechenzeit- und Speicherplatzverbrauch auflisten lassen
- ins Betriebssystem verzweigen.

Eingabefehler werden abgefangen. Außerdem ist es möglich, die letzten 10 Befehle auflisten zu lassen, evtl. zu korrigieren und dann noch einmal auszuführen. Vielfältige help-Funktionen helfen in den meisten Fällen weiter.

6 Höhere Algorithmen

Nun sollen 2 Beispiele höherer Algorithmen aus der konstruktiven Zahlentheorie angegeben werden, die in SIMATH enthalten sind.
Sei $f(x) \in \mathbb{Z}[x]$ ein normiertes separables Polynom vom Grad n, $A_f := \mathbb{Q}[x]/(f(x)) = \mathbb{Q}[\omega]$ für $\omega := x \bmod f(x)$ und O_f der ganze Abschluss von $\mathbb{Z}$ in A_f.
Der Ford-Zassenhaus-Algorithmus ORDMAX berechnet Polynome $g_1(x), \ldots, g_n(x) \in \mathbb{Q}[x]$, die eine $\mathbb{Z}$-Modul-Basis der Maximalordnung O_f bilden, d.h. O_f läßt sich in der Form $O_f = \sum_{i=1}^{n} \mathbb{Z} \cdot g_i(\omega)$ schreiben.
Falls das Polynom $f(x)$ zusätzlich irreduzibel ist, ist die Algebra A_f ein algebraischer Zahlkörper K und $g_1(\omega), \ldots, g_n(\omega)$ eine Ganzheitsbasis von $K/\mathbb{Q}$.
Dieser Algorithmus kann auch auf halb-einfache Polynomalgebren über $\mathbb{F}_p(y)$ angewendet werden, wobei p Primzahl ist. Wenn dann das definierende Polynom $f(x) \in \mathbb{F}_p[y][x]$ außer normiert und separabel zusätzlich irreduzibel ist, wird die Algebra $A_f = \mathbb{F}_p(y)[x]/(f(x))$ ein Kongruenzfunktionenkörper.
Das SIMATH-System enthält diesen Algorithmus sowohl über $\mathbb{Q}$ als auch über $\mathbb{F}_p(y)$ und zusätzlich - als Anwendung - Programme zur Berechnung des Zerlegungsgesetzes in Zahlkörpern und Kongruenzfunktionenkörpern.
Sei $k = \mathbb{F}_p$ endlicher Primkörper, $K = k(x)$ der rationale Funktionenkörper in einer Variablen über k und F eine quadratische Erweiterung von K, also

$F = k(x, \sqrt{f(x)})$, wobei $f(x)$ quadratfreies Polynom über k ist.
Das Arithmetiksystem enthält für solche F/k, sog. quadratische Kongruenzfunktionenkörper, Programme zur Berechnung von

- Klassenzahl und Klassengruppe
- Einheitengruppe
- Regulator.

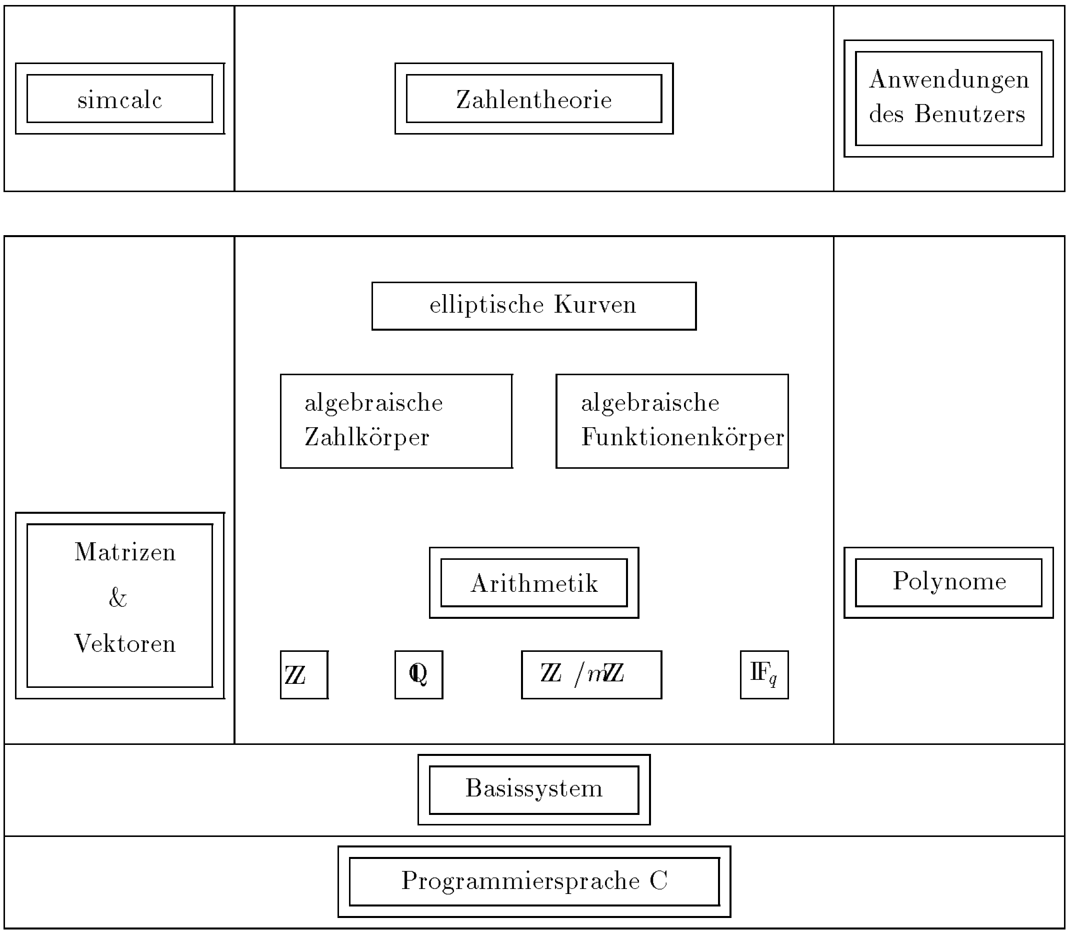
Zugrunde liegen dabei der Kettenbruchalgorithmus von E. Artin und der baby step / giant step Algorithmus von D. Shanks.

7 Anwendungen

Vor allem im Bereich der elliptischen Kurven ist SIMATH bisher erfolgreich angewendet worden, z.B. zur Berechnung von

- Kurven hohen Rangs über Zahlkörpern
- Torsionsgruppen über Zahlkörpern
- Höhen über lokalen und globalen Körpern
- L-Reihen über Zahlkörpern
- Kongruenzzahlserien und -tabellen.

Die meisten dieser Algorithmen werden in nächster Zukunft in das System eingebaut.



SIMATH-Gruppe
Fachbereich 9 Mathematik
Universität des Saarlandes
D 6600 Saarbrücken

Prof. Dr. H.G. Zimmer
Dipl.-Math. U. Schneiders
Dipl.-Math. P. Serf

KANT - ein Software-Paket für zahlentheoretische Anwendungen

Michael E. Pohst, Nicole Schulte

1 Was ist KANT?

KANT ist ein in ANSI Fortran 77 geschriebenes Software Paket (über 600 Haupt- und Unterprogramme) und wird seit 1987 an der Heinrich-Heine-Universität Düsseldorf entwickelt. Es wurde auf verschiedenen Systemen (Siemens 7.580-S, Atari, APOLLO Workstations, SUN 3/60) implementiert. KANT wird ständig weiter entwickelt, das heißt, es werden laufend neue Unterprogramme hinzugefügt, bestehende werden verbessert.
Der Name **KANT** entstand aus

Computational **A**lgebraic **N**umber **T**heory,

wobei **K** als Hinweis auf die deutsche Herkunft zu verstehen ist.
Die wesentlichen Gründe für die Entstehung KANTs sind:

- Entwicklung und Erprobung zahlentheoretischer Algorithmen
- Erstellen großer Tabellen mit Zahlkörperdaten
- Untersuchungen in Zahlkörpern, um Vermutungen zu unterstützen oder zu widerlegen
- Lösen nichtlinearer diophantischer Gleichungen

KANT gliedert sich in sechs Bereiche:

1. *Basic tools*
   - Ein- und Ausgabe-Programme
   - grundlegende Rechenarithmetik, wie zum Beispiel Vektor- und Matrixoperationen
2. *Higher level tools*
   - Nullstellenberechnung von Polynomen
   - Determinanten und Inverse von Matrizen
   - Hermite- und Smith- Normalform
   - LLL-Reduktion
   - Modifizierte LLL-Reduktion
   - Auszählen von Gitterpunkten in Ellipsoiden
   - Berechnung kurzer und kürzester Gittervektoren
   - Ganzzahlige Relationen zwischen reellen Vektoren
   - Polynomfaktorisierung modulo einer Primzahl (Berlekamp- sowie Cantor /Zassenhaus-Verfahren)
   - Irreduzibilitätstest für ganzzahlige Polynome
3. *Number field tools*
   - Arithmetik für algebraische Zahlen
   - Ideal-Arithmetik ($\mathbb{Z}$ -Basis, 2-Element-Normaldarstellung)
   - Regulatorabschätzungen
4. *Internal tools*
   - Systemzeit
   - Verwaltung von globalen Variablen
5. *Extended arithmetics tools*
   - Intervall-Arithmetik
   - Polynom-Arithmetik
   - lange Arithmetik für ganze und reelle Zahlen
6. *Hauptprogramme zur Berechnung von:*
   - Ganzheitsbasen
   - unabhängigen Einheiten
   - Grundeinheitensystemen
   - Torsionseinheiten
   - Lösungen von Normgleichungen
   - Klassengruppe und Klassenzahl
   - Lösungen von Indexformgleichungen
   - Lösungen von Thue-Gleichungen
   - Körperisomorphismen
   - reduzierten Gitterbasen

2 Arbeiten mit KANT

Es gibt zwei Möglichkeiten, KANT zu nutzen:

1. Anwenden der bestehenden Hauptprogramme
2. Entwickeln eigener Hauptprogramme unter Verwendung der Unterprogramme aus KANT

Fast alle bestehenden Hauptprogramme benötigen Zahlkörperdaten. Ein Zahlkörper $F = \mathbb{Q}(\rho)$ wird von einer Wurzel ρ eines normierten, irreduziblen Polynoms $f(t) \in \mathbb{Z}[t]$ vom Grad n erzeugt. Die meisten Algorithmen führen dann Berechnungen im Ring der ganzen Zahlen o_F von F aus. Dieser Ring hat eine $\mathbb{Z}$ -Basis $\omega_1, \ldots, \omega_n$, wobei die ω_i $(1 \leq i \leq n)$ bis auf einen gemeinsamen Nenner ganzzahlige Linearkombinationen von
$1, \rho, \ldots, \rho^{n-1}$ sind.
Diese Daten müssen in einem bestimmten Format vorliegen. Man kann sie entweder selber als Eingabedatei anlegen oder sie werden von entsprechenden Hauptprogrammen erzeugt. Eine typische Eingabedatei könnte so aussehen:

```
(F=)
  0, -9, 4, 9, -3, -1
(ITB=)
1,R,R2,R3,R4,(-8+2R-R2+2R3+7R4+R5)/19
(FUNIT=)
-16,-27,-34,  53,  35, -92
 58, 38,  1, -63, -38, 112
 13, -7, 21, -18, -14,  34
-24, -2, 13,  25,  15, -50
  2,  0,  0,  -1,  -1,   4
(END=)
```

Dies bedeutet:

- $f(t) = t^6 - 9t^4 + 4t^3 + 9t^2 - 3t - 1$,
- o_F hat eine $\mathbb{Z}$ $-$Basis $\omega_1, \ldots, \omega_6$ mit
  $\omega_i = \rho^{i-1} \quad (i = 1, 2, 3, 4, 5)$,
  $\omega_6 = \frac{-8+2\rho-\rho^2+2\rho^3+7\rho^4+\rho^5}{19}$,
- die Einheitengruppe von o_F ist
  $< -1 > \times < \eta_1 > \times \ldots \times < \eta_5 >$ mit
  $\eta_1 = -16\omega_1 - 27\omega_2 - 34\omega_3 + 53\omega_4 + 35\omega_5 - 92\omega_6$
  $\eta_2 = 58\omega_1 + 38\omega_2 + \omega_3 - 63\omega_4 - 38\omega_5 + 112\omega_6$
  $\eta_3 = 13\omega_1 - 7\omega_2 + 21\omega_3 - 18\omega_4 - 14\omega_5 + 34\omega_6$
  $\eta_4 = -24\omega_1 - 2\omega_2 + 13\omega_3 + 25\omega_4 + 15\omega_5 - 50\omega_6$
  $\eta_5 = 2\omega_1 - \omega_4 - \omega_5 + 4\omega_6$

Die Ganzheitsbasis und ein Grundeinheitensystem kann auch mit KANT nach Eingabe der Polynomkoeffizienten ausgerechnet werden.
KANT ist in der Lage, erzeugte Ausgabedaten wieder als Eingabedaten zu verwenden. Dies geschieht mit den oben erwähnten Ein- und Ausgaberoutinen. Außerdem liegen schon Daten über Zahlkörper vor (darunter allein über eine halbe Million Körper vom Grad ≥ 4), die man als Eingabe verwenden kann.
Im allgemeinen kann aufgrund der Struktur der Ganzheitsbasis und der Einheitengruppe ganzzahlig gerechnet werden. Dadurch werden computerabhängige Rundungsfehler vermieden. Wenn sich das Rechnen mit reellen Zahlen nicht vermeiden läßt (etwa bei der Berechnung der Nullstellen eines Polynoms), kann entweder lange Arithmetik oder Intervall-Arithmetik verwandt werden. Die Intervall-Arithmetik berechnet für jede Operation zwischen reellen Zahlen den entstehenden absoluten Fehler. Wird dieser absolute Fehler zu groß, geben die entsprechenden Unterprogramme eine Fehlermeldung aus. Dies ist zum Beispiel dann der Fall, wenn bei Division oder Logarithmenbildung die Null im betreffenden Intervall liegt.
Einige Programme, wie zum Beispiel das Programm zur Lösung von Normgleichungen führen die absoluten Fehler jeder Zahl mit. Sie schalten automatisch auf lange Arithmetik, falls dieser zu groß wird. Die benötigten Daten werden dann mit einer hinreichend grossen Genauigkeit ausgerechnet. Danach wird wieder auf Intervall-Arithmetik geschaltet.
Lange Arithmetik ist nach Möglichkeit zu vermeiden, da sie sehr rechenaufwendig wird.
Viele Programme können ihre Berechnungen mit doppelter Genauigkeit oder Intervall- oder langer Arithmetik ausführen. Dies läßt sich durch einen Schalter steuern.
Eine Benutzeroberfläche für KANT ist in Entwicklung. Mit dieser lassen sich dann die Hauptprogramme von KANT im Dialog benutzen.

Prof. M.E. Pohst
Dipl.-Math. N. Schulte
Mathematisches Institut
Heinrich-Heine-Universität Düsseldorf
W-4000 Düsseldorf
E-mail: pohst@dd0rud81.bitnet

LiE, A software package for Lie group computations

Computer Algebra Group
c/o M. A. A. van Leeuwen
CWI
P. O. Box 4079,
1009 AB Amsterdam
The Netherlands

```
LiE LiE     LiE LiE     LiE LiE LiE LiE
LiE LiE     LiE LiE     LiE LiE LiE LiE
LiE LiE     LiE LiE     LiE LiE
LiE LiE     LiE LiE     LiE LiE
LiE LiE     LiE LiE     LiE LiE LiE LiE
LiE LiE     LiE LiE     LiE LiE LiE LiE
LiE LiE     LiE LiE     LiE LiE
LiE LiE                 LiE LiE
LiE LiE LiE LiE LiE     LiE LiE LiE LiE
LiE LiE LiE LiE LiE     LiE LiE LiE LiE
```

version 2.0

LiE is a specialised computer algebra package for computations concerning Lie groups and algebras, and their representations. LiE is developed by the Computer Algebra Group of CWI, Amsterdam. The package offers a large number of algorithms performing computations related to Lie groups and algebras, involving amongst others root systems, the Weyl group and its action on the root/weight lattice, symmetric group characters, semisimple elements and their centralisers, highest weight modules and other representations, their characters, tensor product decompositions and branching to subgroups, symmetric, alternating and full tensor powers, and more generally arbitrary plethysms. Furthermore it provides an interactive programming environment with variables and control structures, which allow the user to customise the package by defining new functions and extending existing ones. In addition to this LiE gives on-line help and background information on Lie groups.

LiE provides the user with a few simple data types: integers, integer vectors and matrices, (multivariate) polynomials with integer coefficients and values describing types of reductive complex Lie groups (or algebras). With these objects however, a large number of concepts relating to the study of Lie groups or algebras can be described. For instance, LiE can perform calculations in roots systems, computing Cartan matrices, sets of positive roots, inner products, closures of subsets of roots, etcetera, and here roots are represented by integer vectors, and sets of roots by matrices. LiE may also compute in the Weyl group, finding for instance reduced expressions and elements of minimal length in (double) cosets for Weyl subgroups, and in this case vectors represent expressions for Weyl group elements as product of simple reflections. Alternatively Weyl group elements may be used to act upon the root- or weight lattice, in which case they can be represented by square matrices. In the study of the symmetric groups (which is needed for instance to define plethysms) vectors may stand for permutations, partitions or Young tableaux, and matrices may be used to represent symmetric group characters. Finally, in the operations pertaining to the representation theory of reductive Lie groups—the computationally hardest type of algorithms provided by LiE—the predominant types types are vector (representing weights) and polynomial (for sets of weights with multiplicities, representing either reducible modules as decomposed into highest weight modules, or characters of modules). Having so many algorithms related to Lie groups and their representations available in one package, and with the flexible programming interface provided by the simple but useful data types, LiE provides a powerful tool for studying Lie groups, both for students and for academic researchers in the field.

Version 1.3 of LiE was released in Januari 1990, and is being used in many academic environments all over the world. An improved version 2.0 is to be released in summer 1991. While version 1.3 already contains many of the key algorithms needed for Lie algebraic computations, the new version features a large number of improvements, and additional algorithms. The most notable differences with the earlier version are the following:

- introduction of the polynomial data type, which is especially suited to represent characters of modules or decompositions of reducible modules,
- new functions implementing Young tableau algorithms, notably tensor product computation for the groups SL_n by means of the Littewood-Richardson rule and computation of the Robinson-Schensted correspondence,
- new functions for computing the Bruhat order on the Weyl group, and computing the Kazhdan-Lustig polynomials,
- improvement and extension of many of the existing functions, in particular generalising them to the case of non-simple groups and reducible modules,
- improved documentation, both in the form of an extensive manual, providing a complete description of all aspects of the package, explanation of the theoretical background needed to formulate problems in the language of LiE and many instructive examples, as well as improved on-line help faciltilies.

The package is written in portable ‚C', and can be made available on any computer system that provides a C-compiler. Currently LiE has been made available on the following computers:

- VAX (under BSD UNIX and under VAX/VMS)
- SUN3 workstations
- SUN4 workstations and Sparcstations (SunOS)
- IBM RT (AIX 2.2.1)
- DECstations (DEC3100 under Ultrix)
- IBM PC compatibles (MS DOS)
- Apple Macintosh
- For users of other computers who are willing to perform the porting

to their computer themselves, a (C-)source version of LiE 2.0 will be made available from a mailserver.
The package is provided on several media; the ordinary distribution for the VAX version is on tape, and for the other versions either (for workstation versions) on cartridge or on floppy disk. The price of LiE depends on the medium: it is *fl*150 for the diskette versions, *fl*175 for the tape version and *fl*230 for the cartidge versions (*fl*1 $\approx$ \$0.55). The price includes the manual and shipping costs (but no VAT).
LiE can be ordered at below address, mentioning the desired version and medium. Enquiries for more information can also be directed to this address.

Background information

The theory of (reductive) Lie groups and algebras and their representations is very well suited for a computational approach, due to the classification of these groups and the uniform and explicit description of their representations, and many results such as character formulae and rules for computing tensor product decompositions. The combinatorial complexity of the computations involved makes them very time consuming when performed in a general purpose computer algebra system such as Maple. On the other hand, the complexity of the data structures needed for the computations is relatively low when compared to other typical applications of symbolic computation: in most cases vectors and matrices with integer coefficients suffice, supplemented by Laurent polynomials in a fixed number of indeterminates (viz. the rank of the Lie group or algebra). Based on these ideas the software package LiE has been developed, in which the time consuming basic algorithms, such as computation and decomposition of characters, and traversal of Weyl group orbits are performed by algorithms which are directly compiled into the package, and are therefore executed very efficiently. To provide the user with a friendly interface to these functions, of which there are about 100 different ones, LiE is equipped with an interpreter which provides a high level mathematical language from which they are directly accessable. This language allows the user to manipulate the following types of objects: integers (of arbitrary size), vectors, matrices, (multivariate Laurent) polynomials (all these with integer coefficients), groups (e.g., D_4, E_8 or A_3T_1) and character strings. These types are provided with the usual arithmetic operations such as addition and multiplication wherever applicable, and moreover with a number of additional operations that facilitate programming with these objects. Furthermore input, output and editing facilities are available.

MAS – Modula–2 Algebra System

Heinz Kredel, Universität Passau

Abstract: MAS is an experimental computer algebra system combining imperative programming facilities with algebraic specification capabilities for design and study of algebraic algorithms. MAS views mathematics in the sense of universal algebra and model theory and is in some parts influenced by category theory. We give an overview of system design and the current state of the MAS project and the available algorithms.

Starting point for the development of MAS was the requirement for a computer algebra system with an up-to-date language and design which makes the existing ALDES / SAC–2 algorithm libraries available. At this time there have been about 650 algorithms in ALDES / SAC–2 and in addition I had 450 algorithms developed on top of ALDES / SAC–2. The tension of reusing existing software in an interactive environment with specification capabilities contributes most to the evolution of MAS.
The resulting view of the software has many similarities with the model theoretical view of algebra. The abstract specification capabilities are realized in a way that an interpretation in an example structure (a model) can be denoted. This means that is is not only possible to compute in term models modulo some congruence relation, but it is moreover possible to exploit a fast interpretation in some optimized (or just existing) piece of software.
In the sequel we give a short summary of the design, the system components and current state of MAS.

1 Design Overview

MAS (Modula–2 Algebra System) is an experimental computer algebra system combining imperative programming facilities with algebraic specification capabilities for design and study of algebraic algorithms. The new things are: MAS brings together specifications of abstract data types, imperative programs, controlled polymorphism, compiled libraries and term rewriting. The **goal** of the MAS system is to provide:

1. an *interactive* computer algebra system
2. comprehensive algorithm *libraries*, including the ALDES/SAC–2 system [Collins 82],
3. a familiar program *development* system with an efficient compiler,
4. an algebraic *specification* component for data structure and algorithm design
5. algorithm *documentation* open to the users.

Key **attributes** of the MAS system are:

1. portability (it is portable to a computer during a student exercise 'Praktikum'), machine dependencies isolated in a small kernel,
2. extendability (it is possible to add and interface to external algorithm libraries), open system architecture,
3. transparent low level facilities: storage management (garbage collection is provided without user cooperation), stable error handling (no system break down on misspelled expressions and runtime exceptions), input / output with streams (no changes are required to existing libraries to redirect I/O).
4. effectivity (critical parts can be compiled and still be accessed interactively)
5. expressiveness (possibility to specify abstract algebraic concepts like rings or fields)

The goals and attributes have been achieved by the following main **design concepts**:
MAS replaces the ALDES language [Loos 76] and the FORTRAN implementation system of SAC–2 by the Modula–2 language [Wirth 85]. Modula–2 is well suited for the development of large program libraries; the language is powerful enough to implement all parts of a computer algebra system and the Modula–2 compilers have easy to use program development environments.
To provide an interactive calculation system, a LISP interpreter is implemented in Modula–2 with full access to the library modules. For better usability a Modula–2 like imperative (interaction) language was

defined, including a type system and function overloading capabilities. To increase expressiveness, high-level specification language constructs have been included together with conditional term rewriting capabilities. They resemble facilities known from algebraic specification languages like ASL [Wirsing 86].

Theoretical design issues are:

MAS views mathematics in the sense of universal algebra and model theory and is in some parts influenced by category theory. The MAS language and its interpreter has no *knowledge of mathematics* and mathematical objects; however it is capable to describe (specify) and implement mathematical objects and to use libraries of implemented mathematical methods. Further the imperative programming, the conditional rewriting and function overloading concepts are separated in a clean way. There exists a denotational semantics description of the MAS language in [Kredel 91].

More precisely, a *specification* construct defines a formal language L from sorts and operations Σ. Included are the capability to join specifications and to rename sorts and operations during import of specifications. This allows both the specification of abstract objects (rings, fields), concrete objects (integers, rational numbers) and concrete objects in terms of abstract objects (integers as a model of rings). Specifications can be parameterized in the sense of λ abstraction.

The *semantics* of a specification is defined by a class of Σ–structures. It can be described either by implementations, axioms or models. The *implementation* part describes (imperative) procedures and data representations. The semantics for procedures is fixpoint semantics of the respective λ–terms.

The *axioms* part describes conditional rewrite rules which define a reduction relation on the term algebra generated by the sorts and operations of the specification. The semantics is therefore the class of models of the term algebra modulo the (congruence) relation. Currently there are no facilities to solve conditional equations.

The *model* part describes the association between a specification and (several) algebraic structures (models). Models may be given by native implementations in Modula–2 libraries, imperative functions or term rewrite rules and can be associated to suitable formal languages defined by Σ. The semantics is the interpretation of the Σ functions in the associated model. The desired model for interpretation is selected according to the types of the actual function parameters.

At this point one sees a limitation of universal algebra: the functions in the models must all have the same arity as the Σ functions. This prohibits for example the usage of finite fields (of some characteristic p) as models for fields, since the functions for finite fields need an additional parameter for the characteristic. This limitation is not present in category theory, so more precisely the model part describes a functor between two categories (here between a Σ_1 and a Σ_2 structure). Additional information required for parameters can be supplied from the 'descriptors' of the actual parameters.

Since functional terms entered into the system may involve subterms in several semantics, their *evaluation* order is defined as follows: If there is a model in which the function has an interpretation and a condition on the parameters is fulfilled, then the interpretation of the function in this model is applied to the interpretation (values) of the arguments. If there is an imperative procedure, then the procedure body is evaluated in the procedure context. If the unification with the left hand side of a rewrite rule is possible and the associated condition evaluates to true, then the right hand side of the rewrite rule is evaluated. Otherwise the functional term is left unchanged.

Related systems:

In contrast to other computer algebra systems (like Scratchpad II [Jenks 85]), the MAS concept provides a clean separation of computer science and mathematical concepts. The programmer / user has complete control over all operator overloadings. MAS specifications provide multiple inheritance.

In contrast to functional programming languages (like SML [Appel 88]) which implement typed lambda calculus the types of operations are not deduced from the program text but must be explicitly defined in the specification of an operation, in a variable declaration or in a typed string expression.

Compared to pure specification systems, MAS provides the possibility that operations in algebraic structures can be implemented by fast (imperative and/or library) procedures and not only by rewrite rules in term models.

A weak point in the current MAS design is that the language is only interpreted. This is actually not a handicap in execution speed since compiled libraries can be used, but in a too weak semantic analysis of the specifications. This means that certain errors in the specifications are only detected during actual evaluation of an expression.

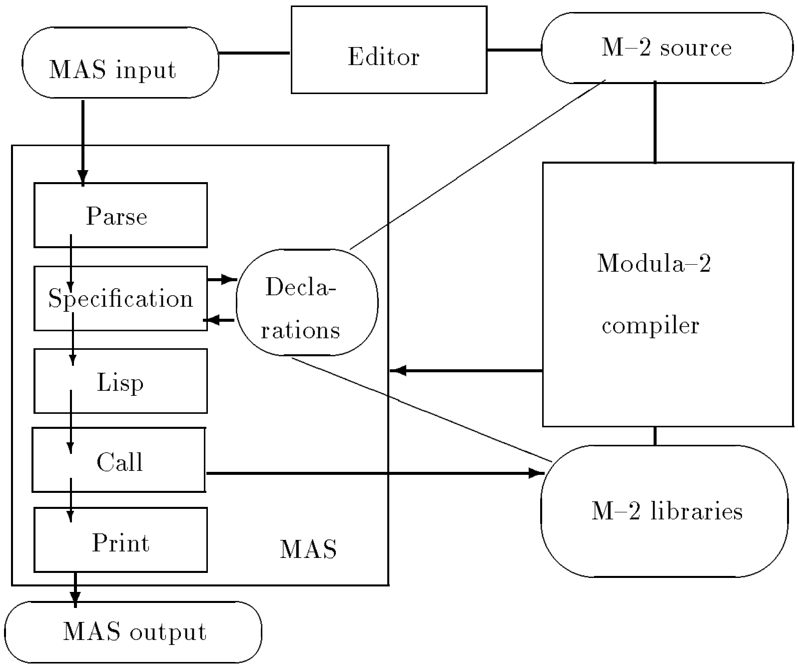


Abbildung 1: System Components

2 System components

The MAS system components are identified in figure 1. Active components (programs) are enclosed in square boxes and passive components (data) are enclosed in oval boxes. Arrows indicate flow of data and lines between boxes show that the boxes are related in some way.

As mentioned in the overview section MAS itself is a Modula–2 program. Thus the MAS program can be recompiled and linked together with other symbolic and numerical libraries by a suitable Modula–2 compiler. This is shown as an arrow from the compiler box on the right to the enclosing MAS box on the left.

On the top line the editor box both acts on the Modula–2 source code (on the right) and the MAS input data (on the left). The input is processed by the following internal components:

1. The parser for the MAS language (Parse box): character strings in concrete syntax are transformed into abstract syntax trees. Static syntax check together with variable scope analysis is performed.
2. The specification processor (Specification box) with an attached data base of declarations (Declarations box): declarations are extracted from the parse tree and stored in the declaration base, information is retrieved during interpretation. The declarations reflect the Modula–2 source code and the library structure.
3. The LISP interpreter (LISP box): according to the type or the function name of an S–expression innermost (that is eager) evaluation is performed.
4. The interface to the compiled library procedures (Call box): if external functions are encountered, then compiled procedures are restored and called with the appropriate parameters.
5. Finally the results are displayed by the (pretty) printing part (Print box).

3 Achievements and Current State

The steps towards the MAS system have been reported in [Kredel 88] (from ALDES / SAC–2 to

Modula–2, storage management); in [Kredel 90] (on the implementation of a LISP interpreter with access to compiled Modula–2 procedures, the interaction language) and in [Kredel 91] (specification component, denotational semantics). Further there is a parser for the ALDES language and the MAS interpreter is now able to evaluate ALDES statements (although with low performance). Versions of the MAS system are running on Atari ST (TDI and SPC Modula–2 compilers), IBM PC/AT (M2SDS and Topspeed Modula–2 compilers) and Commodore Amiga (M2AMIGA compiler).
The ALDES / SAC–2 libraries have been implemented including

- the Arithmetic and Polynomial Systems
- the Polynomial Factorization System
- and the Real Root Isolation System.

From the DIP system [Gebauer Kredel 83], [Kredel 88]

- the Buchberger Algorithm System and the
- Ideal Decomposition and
- Ideal Real Root System have been implemented.

Gröbner Bases are also available for non-commutative polynomial rings of solvable type [Kandri-Rodi Weispfenning 86]. The combination of the MAS programs with numerical Modula–2 libraries has been tested. The mathematical libraries have been enlarged by a package for linear algebra.
Current developments include an implementation of parametric (comprehensive) Gröbner bases [Weispfenning 90] and modules ofsyzygies.

Availability: MAS (Version 0.6x) is available on electronic networks (internet) via anonymous ftp from: `alice.fmi.uni-passau.de` = `123.231.10.1` or from the author.

References: (selected)
[Collins 82] G.E. Collins, R. Loos: *ALDES/SAC-2 now available*, SIGSAM Bulletin 1982, and several reports distributed with the ALDES/SAC-2 system.
[Kredel 90] H. Kredel: *MAS Modula–2 Algebra System*, Proc. DISCO 90 Capri, LNCS 429, pp 270-271, Springer, 1990.
[Kredel 91] H. Kredel: *The MAS Specification Component*, Proc. PLILP 91, University of Passau, August 1991.

Das Darstellungstheorie–Programmpaket CREP

Peter Dräxler

Das Programmpaket CREP (Combinatorial REPresentation theory of finite-dimensional algebras) wird gegenwärtig in der Arbeitsgruppe „Darstellungstheorie endlich-dimensionaler Algebren“ der Universität Bielefeld entwickelt. Der Fortschritt dieses mathematischen Teilgebiets in den letzten Jahren legte nahe, ein solches Projekt in Angriff zunehmen. Es hatte sich nämlich gezeigt, daß die Untersuchung interessanter Klassen endlich-dimensionaler Algebren und ihrer Darstellungen auf kombinatorische Probleme führen, für deren Lösung geeignete Algorithmen ausgearbeitet wurden. Stichworte sind etwa Köcher mit Relationen, Darstellungen teilweise geordneter Mengen und ganzzahlige quadratische Formen mit ihren Wurzeln. Ziel ist es, für diese Algorithmen Programme zu implementieren, bzw. die an verschiedenen Orten bereits erstellten Programme, wenn möglich, aufzunehmen. Bisher sind PASCAL-Programme zu folgenden Problemkreisen vorhanden:

a) Berechnung präprojektiver Komponenten.

b) Bestimmung des Darstellungstyps endlicher geordneter Mengen.

c) Berechnung einiger Invarianten (z.B. Cartan-Matrix, Coxeter-Matrix, Wachstumszahl) erblicher Algebren.

d) Test, ob eine gegebene Algebra in einer der bekannten großen Listen ausgezeichneter Algebren (zahme verkleidete Algebren, minimal wilde verkleidete Algebren, aufrichtige darstellungsgerichtete Algebren) enthalten ist.

e) Überprüfung des Darstellungstyps (endlich, zahm, wild) einer vollständig trennenden Algebra mit Hilfe ihrer geordneten Mengen dünner Startmoduln.

f) Bestimmung der Vektorraumkategorie (verallgemeinerter) Einpunkterweiterungen.

Unsere Arbeit konzentriert sich momentan auf den weiteren Ausbau des Pakets, während die Ausgestaltung der Benutzeroberfläche und die Bereitstellung einer ausführlichen Dokumentation erst in einem späteren Stadium geplant sind.

Die Aufnahme von Programmen zu folgenden Problemkreisen ist für die nähere Zukunft vorgesehen:

a) Test, ob eine gegebene ganzzahlige quadratische Form schwach positiv oder schwach nichtnegativ ist.

b) Berechnung von Hall-Polynomen.

c) Test, ob eine gegebene Algebra tubular bzw. iteriert tubular ist.

d) Klassifikation darstellungsendlicher selbstinjektiver Algebren.

e) Bestimmung der universellen Überlagerung eines Köchers mit Relationen.

f) Berechnung der fast zerfallenden Folge, die in einem gegebenen unzerlegbaren Modul endet.

g) Untersuchung des Kohomologierings einer endlich-dimensionalen Algebra.

h) Klassifikation aufrichtiger schwach positiver ganzzahliger quadratischer Formen.

CREP ist so konzipiert, daß es auf allen herkömmlichen Rechnern problemlos implementiert werden kann.
Ferner ist bereits ein Paket von C-Programmen zur Berechnung projektiver Auflösung vorhanden, das von Prof. E. L. Green (Virginia Polytechnic Institute and State University Blacksburg, Virginia) zur Verfügung gestellt wurde.

SOGOS und SPAS

Volkmar Felsch

SOGOS ist ein Programmsystem zum Rechnen in endlichen auflösbaren Gruppen. Es enthält u. a. Programme für die Berechnung von Untergruppen, Faktorgruppen, Konjugiertenklassen, Zentralisatoren, Normalisatoren, verschiedenen Normalreihen und Subnormalreihen, speziellen Untergruppen, Durchschnitt und Erzeugnis von Untergruppen, Übertragung von Eigenschaften zwischen Gruppen, vollständigem Untergruppenverband oder der Burnside'schen Markentafel für auflösbare Gruppen sowie eine ganze Reihe weiterer Algorithmen für nilpotente Gruppen, etwa zur Berechnung von charakteristischen Untergruppen, Komplementen, Supplementen oder Darstellungen. Darüber hinaus ist in SOGOS auch ein wesentlicher Teil des stand-alone „Nilpotent Quotient" Programms aus Canberra enthalten.

SPAS (Subgroup Presentation Algorithms System) ist ein kleines Programmsystem, das eine Reihe von verschiedenen Programmen zum Umgang mit endlichen Präsentationen zusammenfaßt, die teilweise von verschiedenen Autoren aus verschiedenen Orten stammen und meist schon vorher längere Zeit, etwa als stand-alone Programme, benutzt wurden. Dazu gehören

(1) das bekannte „Todd-Coxeter" Programm aus Canberra zur Restklassenabzählung,

(2) das ebenfalls aus Canberra stammende „Tietze Eliminations" Programm,

dann drei alternative Methoden zur Bestimmung einer Präsentation für eine Untergruppe von endlichem Index in einer endlich präsentierten Gruppe, nämlich

(3) das „Reidemeister-Schreier" Programm von G. Havas,

(4) das in St. Andrews entwickelte sogenannte „Modified Todd-Coxeter" Verfahren und

(5) der „Reduced Reidemeister-Schreier" Algorithmus von J. Neubüser,

und schließlich

(6) der sogenannte „Low Index Subgroups" Algorithmus von C. Sims zur Bestimmung von Untergruppen von kleinem Index.

Für alle diese Programme bietet SPAS eine einheitliche Ein- und Ausgabe an. Überdies kann SPAS Dateien erstellen, die direkt von zwei weiteren größeren stand-alone Programmen aus Canberra, dem „Integer Matrix Diagonalization" Programm und dem „Nilpotent Quotient" Programm, gelesen werden können.

SOGOS und SPAS sind zwei ältere, im wesentlichen in Fortran geschriebene Programmsysteme. Beiden liegt das in Sydney von J. Cannon entwickelte Speicherverwaltungsprogramm „Stackhandler" zugrunde, und beide benutzen zur Dateneingabe die Sprache „Slang". Mit den Versionen SOGOS 5.0 und SPAS 2.5 liegt jeweils die endgültige Fassung vor, denn da das derzeit in Aachen entstehende neue gruppentheoretische Programmsystem GAP in absehbarer Zukunft alle ihre Aufgaben übernehmen kann, werden sie nicht mehr erweitert oder verbessert. Ihre Wartung beschränkt sich ausschließlich auf die Korrektur von Fehlern.

Unix-Fassungen von SOGOS und SPAS sind für Maschinen vom Typ SUN 3/60, Hewlett-Packard 9825 oder Masscomp 5000 verfügbar. Außerdem gibt es eine Version, die auf der VAX 11/780 unter VMS läuft. Die Systeme sind erhaeltlich vom Lehrstuhl D für Mathematik, RWTH Aachen, Templergraben 64, D-5100 Aachen. Eine Benutzungsgebühr wird nicht erhoben.

FELIX - ein Computeralgebra–System für konstruktive Algebra

Joachim Apel, Uwe Klaus

1 Einleitung

Das Computeralgebra–System *FELIX* dient dem Rechnen in und mit algebraischen Strukturen. Ursprüngliches Anliegen des Systems war die Arbeit in nichtkommutativen Ringen mit polynomähnlichen Elementen und in deren Quotientenschiefkörpern. Die hier beschriebene Version enthält in gleichberechtigter Weise auch kommutative Polynomringe und Moduln. Zu den ursprünglich vorgesehenen Operationen zwischen Elementen algebraischer Strukturen sind nun auch Operationen zwischen algebraischen Strukturen, insbesondere zwischen Idealen und Untermoduln hinzugekommen. Weiterhin ist auch die Implementierung von Abbildungen zwischen den Strukturen vorgesehen.

FELIX besteht aus drei Ebenen. Die unterste Ebene ist vollständig in Assembler geschrieben. Sie stellt den maschinenabhängigen Teil von *FELIX* dar. Gegenwärtig ist das Programmsystem nur auf IBM-PC-kompatiblen Rechnern unter dem Betriebssystem DOS verfügbar. Der Vorteil dieser Version liegt darin, daß sie sogar auf Rechnern der XT-Klasse lauffähig ist. Der entscheidende Nachteil ist die Speicherbeschränkung auf 640 KByte. Mit der Portierung des Systems unter Benutzung einer DOS-Erweiterung wurde bereits begonnen. Die zweite Ebene ist ein Kommandointerpreter, der sich von der ersten Ebene im Wesentlichen durch seine übersichtlichere Syntax unterscheidet. Die Trennung der beiden Ebenen bewirkt eine große Flexibilität in der dem Nutzer zur Verfügung gestellten Syntax und ermöglicht die relativ einfache Schaffung neuer Sprachkonstruktionen. Die dritte Ebene ist eine Sammlung höherer algebraischer Algorithmen, die dem Nutzer über eingebaute Funktionen und Unterprogramme zur Verfügung gestellt werden, welche selbst in der *FELIX*- Sprache geschrieben sind. Das Hinzufügen neuer Funktionen und deren Zusammenspiel mit den Systemfunktionen ist daher völlig unproblematisch.

Die Nutzung des Systems erfolgt, wie in den meisten Computeralgebra–Systemen üblich, interaktiv. Die Datenstruktur von *FELIX* ist vorwiegend listenorientiert. Darüberhinaus sind spezielle Datentypen, wie Exponentenvektoren und Bitketten, integriert, die insbesondere der Darstellung algebraischer Objekte dienen. Obgleich die Syntax der Sprache prozedural erscheint, liegt ihr eine LISP-artige Philosophie zugrunde. Insbesondere werden Operatordefinitionen selbst als Daten angesehen und sind manipulierbar. Ihre Abarbeitung erfolgt daher interpretativ. *FELIX* besitzt jedoch auch einen Kompiler. Dieser ermöglicht es, während der interaktiven Arbeit Operatordefinitionen zu kompilieren und als Maschinenprogramme in Modulen zusammenzufassen, welche später bei Bedarf eingebunden werden können. Kompiler und Interpreter des Systems sind semantisch äquivalent.

Aufgrund der Tatsache, daß wir uns in allen bisherigen Veröffentlichungen auf die Beschreibung der dritten, der algebraischen Schicht des Systems beschränkt haben (vgl. [1], [2], [3]), konzentrieren wir uns in dem vorliegenden Artikel auf eine Einführung in die Programmiersprache *FELIX*.

2 Die Programmierebene

In der Programmierebene ist es möglich, Operatordefinitionen vorzunehmen. Ein Operatoraufruf gibt stets einen Wert zurück, er kann aber auch Nebeneffekte erzielen. In Abhängigkeit davon, welche der beiden Eigenschaften überwiegt, könnte man auch von Funktionen oder Unterprogrammen sprechen. Eine syntaktische Unterscheidung nach diesem Gesichtspunkt gibt es in *FELIX* jedoch nicht. Eine Operatordefinition erfolgt durch Aufruf einer **operator**-Anweisung. Diese hat die Syntax:

$\langle$ Name $\rangle$ **operator**
        $\langle$ Pliste $\rangle$ **(** $\langle$ Nliste $\rangle$ **)(** $\langle$ Nliste $\rangle$ **)**
    $\langle$ Anweisungsfolge $\rangle$
endo

$\langle$ Name $\rangle$ steht für einen Bezeichner, den Operatornamen. Das System kennt zwei für den Nutzer interessante Klassen solcher Bezeichner. Die erste enthält

die Namen, die aus Buchstaben und Ziffern gebildet sind und mit einem Buchstaben beginnen. Die Bezeichner der zweiten Klasse sind aus Sonderzeichen zusammengesetzt.
$\langle$ Pliste $\rangle$ steht für die Parameterliste, sie kann entweder ein Name sein oder die Form **(** $\langle$ Nliste $\rangle$ **)** haben. Eine Namensliste $\langle$ Nliste $\rangle$ ist eine (möglicherweise leere) Liste von durch Kommata getrennten Namen. Schließlich wird noch der Begriff der Anweisungsfolge benötigt. $\langle$ Anweisungsfolge $\rangle$ steht für (nichtleere) Folgen von durch Semikola getrennten Anweisungen. Eine Anweisung kann ein Ausdruck oder eine Steueranweisung sein. Was darunter zu verstehen ist, wird in den Abschnitten 2.2. und 2.3. beschrieben.

2.1 Variablenbindung

Besteht $\langle$ Pliste $\rangle$ nur aus einem Namen, so kann der Operator mit beliebiger Parameteranzahl aufgerufen werden, er ist n-stellig. Beim Aufruf wird die Liste seiner Aktualparameter an den Namen gebunden. Liegt dagegen eine geklammerte Namensliste vor, so muß bei einem späteren Aufruf des Operators die Anzahl der Aktualparameter mit der Länge dieser Liste der Formalparameter übereinstimmen. Die Aktualparameter werden ausgewertet und entsprechend ihrer Reihenfolge an die Formalparameter gebunden. Stimmen die Parameteranzahlen nicht überein, so erscheint eine Fehlerausschrift.
Die zweite Liste der Operatordefinition enthält die Namen der lokalen Variablen des Operators mit statischem Gültigkeitsbereich. Sie sind nur innerhalb des Operatorkörpers, d.h. der Anweisungsfolge, sichtbar. Im Körper aufgerufene Operatoren können nur über ihre eigenen Aktualparameter auf diese Variablen zugreifen.
In der dritten Liste werden lokale Variablen mit dynamischer Bindung vereinbart. Sie wirken wie globale Variablen auf innerhalb des Operatorkörpers aufgerufene Operatoren. In der Operatordefinition auftretende Variablenbezeichner, die in keiner der drei Listen deklariert wurden, werden als global angesehen. Ihre Bindung wird erst beim Aufruf des Operators festgestellt. Dazu wird die Aufrufkette solange zurückverfolgt, bis zum erstmalig ein Operator erscheint, der den Variablennamen in seiner Liste der Variablen mit dynamischer Bindung enthält. Tritt kein derartiger Operator auf, so wird die Variable als global im Gesamtsystem betrachtet.

2.2 Ausdrucksdefinition

Wir werden die Menge der Ausdrücke rekursiv definieren. Die zentrale Rolle nehmen dabei die Ausdrücke ein, die den Aufruf eines Operators bewirken. Hat der aufzurufende Operator keine speziellen Syntaxregeln, so erfolgt sein Aufruf durch seinen Operatornamen, gefolgt von einer in runden Klammern eingeschlossenen (möglicherweise leeren) Liste von durch Kommata getrennten Ausdrücken. Die Liste besteht aus den Aktualparametern des Aufrufes. Es ist zu beachten, daß das Klammerpaar auch bei leerer Parameterliste nicht weggelassen werden darf, da ein Name in *FELIX* gleichzeitig eine Variable und einen Operator bezeichnen kann.
Diese syntaktische Regel für einen Operatoraufruf kann durch Setzen von Eigenschaften geändert werden. Der Parser wird dann nach dem Einlesen eines Namens mit solchen speziellen Eigenschaften eine Spezialfunktion zum Einlesen des Operatorausdruckes rufen und mittels dieser den Ausdruck generieren, der an den Interpreter weitergeleitet wird. Es bestehen die folgenden Möglichkeiten, einem Operator eine spezielle Eingabesyntax zuzuordnen.

- Infix-Operator
  setinfix($\langle$ NAus $\rangle$, $\langle$ ZAus $\rangle$)
  setinfix($\langle$ NAus $\rangle$, $\langle$ NAus $\rangle$, $\langle$ ZAus $\rangle$)
- Präfix-Operator
  setprefix($\langle$ NAus $\rangle$, $\langle$ ZAus $\rangle$)
  setprefix($\langle$ NAus $\rangle$, $\langle$ NAus $\rangle$)
- Postfix-Operator
  setpostfix($\langle$ NAus $\rangle$, $\langle$ ZAus $\rangle$)
  setpostfix($\langle$ NAus $\rangle$, $\langle$ NAus $\rangle$)
- n-stelliger Operator
  setnary $\langle$ NAus $\rangle$

$\langle$ NAus $\rangle$ ist ein Ausdruck, dessen Wert ein Name ist. Oft wird an dieser Stelle ein quotierter Name verwendet. Analog ist $\langle$ ZAus $\rangle$ ein Ausdruck, der eine positive ganze Zahl als Wert hat. Der erste Parameter gibt den Namen des Operators an, der die Syntaxregel erhalten soll. Der Zahlenparameter wird, so weit er vorhanden ist, als Präzidenz gedeutet. Ein zweiter Name in der Parameterliste kann einen Operator definieren, der die weitere Eingabe des Ausdrucks übernimmt. Darauf soll an dieser Stelle nicht weiter eingegangen werden. Als Beispiel sei auf die Beschreibung der verzweigten Anweisung im nächsten Abschnitt verwiesen. Präfix- und Postfixoperatoren müssen genau einen Parameter besitzen, dieser darf ohne die einschließenden runden Klammern hinter bzw. vor den Operatornamen geschrieben werden. Man beachte, daß **setnary** selbst die Präfix-Eigenschaft hat. Ein Infix-Operator muß entweder zweistellig sein, in diesem Falle wird der Operatorname zwischen die Argumente geschrieben, oder er muß im Sinne der Operatordefinition n-stellig sein und auch die entsprechende Syntaxdeklaration aufweisen. Die nary-Eigenschaften dürfen nur Infix-Operatoren erhalten. Diese Deklaration ist insbesondere für assoziative Operatoren geeignet.

Nach diesen Bemerkungen zur Syntax eines Operatoraufrufes sind wir in der Lage, eine Ausdrucksdefinition vorzunehmen. Ausdrücke 0-ter Stufe sind Namen, ganze Zahlen, (in Doppelapostrophe eingeschlossene) Zeichenketten und Aufrufe nullstelliger Operatoren. Ein Ausdruck (n+1)-ter Stufe ist ein Ausdruck n-ter Stufe oder ein Operatoraufruf, dessen Parameter Ausdrücke n-ter Stufe sind. Die Menge aller Ausdrücke ist die Vereinigung der Ausdrücke n-ter Stufe über alle natürlichen Zahlen n (einschließlich Null).

2.3 Steueranweisungen

Die prozedurale Syntax unserer Sprache legt es nahe, die Steueranweisungen getrennt von den Ausdrücken zu behandeln. Semantisch sind beide sehr ähnlich, insbesondere aufgrund der Tatsache, daß jede Anweisung in *FELIX* einen Wert hat. Ein Hauptgrund der getrennten Behandlung besteht auch darin, daß Steueranweisungen nur innerhalb einer Operatordefinition erlaubt sind, wogegen Ausdrücke selbstverständlich auch im Toplevel verwendet werden dürfen. Im Einzelnen stehen folgende Steuerkonstrukte zur Verfügung.

- Verbundanweisung
  begin ⟨ Anweisungsfolge ⟩ **endb**
- Schleifenanweisung
  loop ⟨ Anweisungsfolge ⟩ **endl**
- bedingte Anweisung
  when ⟨ Ausdruck ⟩ **;** ⟨ Anweisungsfolge ⟩ **endw**
- Verzweigungsanweisung
  if ⟨ Ausdruck ⟩ **then** ⟨ Anweisungsfolge ⟩
  [**else** ⟨ Anweisungsfolge ⟩] **endi**
- Rücksprung
  return(⟨ Ausdruck ⟩ **)**

Die Verbundanweisung bewirkt eine sukzessive Abarbeitung der Anweisungsfolge. Bei der Schleifenanweisung wird diese Abarbeitung zyklisch wiederholt. Falls in der bedingten Anweisung der Ausdruck den Wert **false** hat, so wird die Anweisung verlassen und die Abarbeitung mit der darauffolgenden Anweisung fortgesetzt. Andernfalls wird die Anweisungsfolge innerhalb der bedingten Anweisung sukzessive abgearbeitet. Danach wird die Steuerung nicht an die der bedingten Anweisung folgende Anweisung übergeben, sondern es erfolgt ein Rücksprung aus der aktuellen Blockebene heraus. Eine solche Blockebene kann eine Verbundanweisung, eine Schleifenanweisung, eine Verzeigungsanweisung oder eine Operatordefinition sein.

Die Verzweigungsanweisung arbeitet im Falle, daß der Ausdruck den Wert **false** hat, die auf das Schlüsselwort **else** folgende Anweisungsfolge ab, fehlt dieser Zweig, so wird die Anweisung verlassen. Hat der Ausdruck einen von **false** verschiedenen Wert, so wird die erste Anweisungsfolge abgearbeitet. Nach Abarbeitung der Verzweigung erfolgt in jedem Fall, außer wenn eine Rücksprunganweisung ausgeführt wurde, die Fortsetzung der Abarbeitung hinter der Verzweigung auf unveränderter Blockebene.

Die Rücksprunganweisung bewirkt ein Verlassen des aktuellen Operatoraufrufes, unabhängig von der gegenwärtigen Tiefe der Blockebene. Sie übergibt den Wert ihres Argumentes als Wert des Operatoraufrufes an die aufrufende Ebene.

Wie bereits erwähnt wurde, hat jeder Aufruf einer Steueranweisung einen Wert. Dieser ist jeweils der Wert ihrer zuletzt berechneten Anweisung. Die explizite Benutzung einer Rücksprunganweisung ist dementsprechend nur dann erforderlich, wenn direkt aus einer Schachtelung von Anweisungen zurückgesprungen werden soll. Jedes Konstrukt besitzt ein eigenes Schlüsselwort zum Kennzeichnen seines Endes. Diese Unterscheidung dient der Verbesserung der Syntaxkontrollmöglichkeiten des Interpreters.

Die geringe Anzahl von Sprachkonstrukten mag als Nachteil erscheinen. Mancher Leser wird verschiedenartige Schleifen (wie **for-**, **repeat-** oder **while-**Konstruktionen) oder auch Mehrfachverzweigungen vermissen. Die relative Armut der Sprache beruht darauf, daß hier mit Ausnahme der Verzweigung genau die Konstrukte vorliegen, die auf der untersten, der Assemblerebene, programmiert sind. Mit Hilfe dieser wenigen Steuerkonstrukte lassen sich alle komfortableren Anweisungen aufbauen. Das kann und sollte in maschinenunabhängiger Form in der *FELIX*-Programmiersprache geschehen.

In der Tat ist die verzweigte Anweisung bereits auf diese Weise erstellt wurden. Der Operator **if** ist ein Präfix-Operator mit spezieller Einlesefunktion, er generiert einen für den Interpreter verständlichen, semantisch äquivalenten Ausdruck der Bauart

begin
  when ⟨ Ausdruck ⟩ **;** ⟨ Anweisungsfolge ⟩ **endw ;**
  ⟨ Anweisungsfolge ⟩
endb

Es könnte erstaunen, daß keine Zuweisung unter den Anweisungen zu finden ist. Selbstverständlich existiert eine solche. Wir fassen sie jedoch als Ausdruck auf (als solcher ist sie auch auf dem Toplevel anwendbar). Sie hat die Syntax

$\langle$ Name $\rangle$ **:=** $\langle$ Ausdruck $\rangle$

und wird intern in die Form

assign(quote($\langle$ Name $\rangle$**),**$\langle$ Ausdruck $\rangle$**)**

umgewandelt.

3 Der Kompiler

Ein *FELIX*-Kompileraufruf hat die Gestalt

compile($\langle$ NListAusd $\rangle$**,** $\langle$ NListAusd $\rangle$ **,**
$\langle$ Zeichenkette $\rangle$**,** $\langle$ Zeichenkette $\rangle$**)**

$\langle$ NListAusd $\rangle$ ist ein Ausdruck, dessen Wert eine Liste von Namen ist. Der Operator **compile** bewirkt das Generieren von Maschinenprogrammen zur Abarbeitung der Operatoren, deren Namen in den beiden Namenslisten angegeben sind. Diese werden in einer Datei abgelegt, deren Name durch die erste Zeichenkette angegeben wird. Die in der zweiten Liste aufgeführten Operatoren sind nur lokal innerhalb des Moduls sichtbar, d.h. auf sie kann später nur von Operatoren zugegriffen werden, die sich im selben Modul befinden. Diese Vorgehensweise hat verschiedene Vorteile. Hilfsfunktionen können unsichtbar gemacht werden, sie können daher nicht versehentlich umdefiniert werden oder andere Funktionen zerstören. Außerdem können verschiedene Module Hilfsfunktionen gleichen Namens besitzen.
Die zweite Zeichenkette gibt einen Kommentar an, der im gesetzten Zustand des Schalters **link** auf dem Bildschirm erscheint, sobald der Modul eingebunden wird. Dieses Einbinden erfolgt mittels des Aufrufes

link($\langle$ Zeichenkette $\rangle$ **)** ,

wobei die Zeichenkette den Namen des einzubindenden Moduls angibt. Das Einbinden kompilierter Operatoren bringt folgende Vorteile mit sich:

- Verkürzung der Rechenzeit (ca. auf ein Drittel)
- Entlastung des Datenbereiches, da Module in eigenen Segmenten abgespeichert werden
- Verminderung des Garbage collection - Aufwandes, da kompilierte Operatoren dabei nicht berücksichtigt werden müssen
- Extreme Verkürzung der Zeit zum Einlesen von Operatordefinitionen

Standardmäßig erfolgt keine Ausgabe der Linkerinformation, sie kann durch Eingabe von

on link $

bewirkt werden.

4 Die Algebra-Ebene

Die Algebra-Ebene erlaubt die Definition solcher algebraischer Strukturen, wie kommutative Polynomringe, nichtkommutative Algebren, Körper, Schiefkörper und Moduln. Dabei werden diese Strukturen selbst als Daten betrachtet, sie können in Ausdrücken zur Bildung neuer komplizierterer Strukturen, wie z.B. Restklassenringe oder Moduln, verwendet werden.
Neben dem Rechnen mit Elementen aus den algebraischen Strukturen sind bisher insbesondere Operationen zwischen ein- und zweiseitigen Idealen und Untermoduln möglich. In Zukunft werden dem System auch Abbildungen zwischen den Strukturen, sowie Operationen mit ihnen hinzugefügt werden.
Die Ideal- und Untermoduloperationen basieren vorwiegend auf Buchberger's Algorithmus zur Berechnung von Standardbasen. Dieser kann zum einen direkt angewandt werden. Bei Bedarf gibt das System statistische Auskünfte über Zwischenschritte und den Gesamtverlauf der Standardbasisberechnungen. Es besteht die Möglichkeit, der freien Wahl der Termordnung und einer Auswahlstrategie für die Behandlung kritische Paare. Beides kann entscheidenden Einfluß auf den Speicher- und Rechenzeitbedarf haben. Zum anderen ist der Algorithmus u.a. in Funktionen zur Berechnung von Minimalbasen, Idealdurchschnitten, Idealquotienten, Syzygienketten, freien Auflösungen und Eliminationsidealen eingebaut.

Literatur

[1] J. Apel, U. Klaus, Implementation aspects for non-commutative domains. Erscheint in : Proc. IV.Int.Conf.Computer Algebra in Physical Research 1990, JINR Dubna, Moskau, 1990.

[2] J. Apel, U. Klaus, *FELIX* - an assistant for algebraists. Erscheint in : Proc. ISSAC'91, Bonn, 1991.

[3] J. Apel, U. Klaus, *FELIX* - User's manual (Draft version). Unveröffentlicht, Leipzig, 1991.

Symmetriesuche per Computer

Benno Fuchssteiner und Waldemar Wiwianka

1 Symmetrien

Überall, wo echte Wechselwirkung von dynamischen Variablen beschrieben wird, hat man es mit nichtlinearen Systemen zu tun. Im Gegensatz zu linearen Systemen beschreiben nichtlineare Systeme also die interessanteren Phänomene. Solche Systeme treten, je nach Modellierung, in vielfältiger Form auf; als nichtlineare partielle Differentialgleichungen, oder auch als nichtlineare Flüsse auf Punktgittern, manchmal aber auch als komplizierte quantenmechanische Evolutionen, bei denen die Hamiltonoperatoren hochgradig nichtlinear von den Feldvariablen abhängen. Leider aber sind nichtlineare Systeme kompliziert. Dies liegt nicht nur daran, daß bewährte Methoden, wie zum Beispiel die Spektraltheorie, versagen, sondern hat strukturelle Ursachen. Wegen der Kompliziertheit ist man häufig schon zufrieden, wenn man spezielle Lösungen kennt und deren Bedeutung versteht. Bei partiellen Differentialgleichungen findet man spezielle Lösungen meist dadurch, daß man die entsprechenden Systeme auf *invarianten* Untermannigfaltigkeiten betrachtet. Diese muß man allerdings erst bestimmen, ihre Konstruktion setzt die Existenz invarianter Tensoren voraus.

Die einfachsten Beispiele für invariante Tensoren sind *infinitesimale Generatoren* einparametriger Symmetriegruppen. Im Gegensatz zu linearen Systemen, die, abhängig von der Dimension der Mannigfaltigkeit, immer viele Symmetriegruppengeneratoren besitzen, braucht es solche bei nichtlinearen Systemen überhaupt nicht zu geben. Hat jedoch ein nichtlineares System eine *große* abelsche Symmetriegruppe, so führt dies im allgemeinen dazu, daß man ganze Klassen von Lösungen explizit bestimmen kann.

Bei endlichdimensionalen Hamiltonsystemen kann man diese etwas vage Aussage durch den Satz von Liouville-Arnold präzisieren: Hat ein Fluß auf einer $2N$-dimensionalen Mannigfaltigkeit N unabhängige Symmetriegruppengeneratoren, die durch hamiltonische Vektorfelder gegeben sind, und vertauschen diese Generatoren untereinander, so ist der Fluß linearisierbar. Dabei heißt linearisierbar, daß man eine Transformation der Mannigfaltigkeit auf den $\mathbb{R}^{2N}$ oder auf einen $2N$-dimensionalen Torus finden kann, auf welchem die Evolution dann linear ist.

Der sich in diesem Satz ausdrückende Sachverhalt gilt, mit gewissen Abstrichen, auch auf unendlichdimensionalen Mannigfaltigkeiten, also zum Beispiel auf einer Mannigfaltigkeit geeigneter Funktionen $u(x)$, auf denen ein Fluß, durch eine nichtlineare partielle Differentialgleichung, etwa die Korteweg-de Vries-Gleichung

$$u_t = u_{xxx} + 6u_x u \tag{1}$$

beschrieben wird. Denn auch bei nichtlinearen unendlichdimensionalen Flüssen stellt die Angabe einer großen Symmetriegruppe einen wichtigen, oft den entscheidenden Schritt bei der Untersuchung des Systems dar.

Abstrakt lassen sich Symmetriegruppengeneratoren leicht charakterisieren: ihre Vektorfelder müssen in der Lie-Algebra aller Vektorfelder mit dem Ausgangsvektorfeld, welches das dynamische System beschreibt, vertauschen. So einfach dieses Kriterium auch sein mag, die Angabe eines Symmetriegruppengenerators, oder selbst die Verifikation daß ein vorgegebenes Vektorfeld Symmetriegruppengenerator ist, ist häufig eine schwierige Angelegenheit.

Bei unendlichdimensionalen Systemen beschreibt man in der Praxis große Symmetriegruppen, indem man eine Rekursionsformel für die entsprechenden Generatoren angibt. Dies geschieht zum Beispiel durch die Angabe eines jeweils einfach kokontravarianten Tensors, der invariant unter dem vorgegebenen Fluß ist. Ein solcher Tensor definiert im Tangentialbündel der Mannigfaltigkeit eine lineare Abbildung, welche Symmetriegruppengeneratoren in ebensolche überführt. Zum Beispiel ist

$$\Phi = D^2 + 2u + 2DuD^{-1}, \quad D := \frac{d}{dx}$$

ein solcher Tensor für die Gleichung (1). In diesem Fall läßt sich leicht zeigen, daß die auf diese Weise durch den Tensor Φ erzeugten Generatoren untereinander vertauschen. In vielen Fällen sind aber solche Aussagen mit riesigen Rechnungen verbunden, mitunter sogar mit dem Nachweis, daß ein Integrodifferentialoperator von vielen Milliarden Termen identisch verschwindet. Trotz solcher rechneri-

scher Schwierigkeiten ist die Angabe eines entsprechenden invarianten Tensors, der häufig die nahezu vollständige Lösung eines beliebigen Anfangswertproblemes erlaubt, eine höchst wünschenswerte Angelegenheit.
Für den Übergang von der Angabe des entsprechenden invarianten Tensors bis zur Lösung des allgemeinen Anfangswertproblems gibt es eine Reihe von Standardverfahren, zum Beispiel die *Inverse Streumethode*. Dabei faßt man den Operator Φ als Integrodifferentialoperator auf, für den durch Gleichung (1) ein *isospektraler* Fluß definiert ist. Die Funktion $u(x,t)$ spielt dabei die Rolle eines Potentials. Eine praktische Lösung des Anfangswertproblemes besteht dann darin, daß man zu gegebener Anfangsvorgabe $u(x,t_0)$ die Streudaten berechnet, die sich aus dem durch diesen Operator definierten Streuproblem ergeben. Wegen der Invarianz des Operators unter dem gegebenen Fluß findet man für die zeitliche Entwicklung der Streudaten meist eine lineare Evolution, was bedeutet, daß man aus den gegebenen Anfangsstreudaten in einfacher Weise die Streudaten zu beliebigen Zeitpunkten ausrechnen kann. Man muß also nur noch eine inverse Transformation anwenden, nämlich aus den Streudaten zu beliebiger Zeit t auf das Potential $u(x,t)$ zurückrechnen, um die Lösung zu finden.
Die Möglichkeit der Anwendung der Inversen Streumethode zur Lösung eines allgemeinen Anfangswertproblems ist also ebenfalls an die Existenz "vielerëinparametriger Symmetriegruppen gebunden.
Gleichungen, die große Symmetriegruppen besitzen, haben noch weitere interessante Eigenschaften. Sie weisen zum Beispiel Solitonlösungen auf, und man findet für sie häufig eine Schar von Auto-Bäcklund-Transformationen.

2 Suche per Computer

Vor einiger Zeit haben wir uns die Aufgabe gestellt für vorgegebene dynamische Systeme computerisierte Tests für die Existenz von Symmetriegruppen zu entwickeln. Und nicht nur dies, wir wollten auch in der Lage sein, die möglichen Rekursionsformeln mit dem Computer zu bestimmen. Bei gewissen Klassen von dynamischen Systemen ist uns dies inzwischen gelungen. Als entscheidendes Hilfsmittel haben sich dabei die *Mastersymmetrien* herausgestellt.
Mastersymmetrien sind Vektorfelder, die zwar nicht invariant unter dem gegebenen Fluß sind, aber bei denen der Kommutator mit dem Vektorfeld des Flusses zu einer Invarianten führt. Bei vollständig integrablen Flüssen lassen sich aus der Angabe einer solchen Mastersymmetrie weitere invariante Vektorfelder durch sukzessive Kommutatorbildung bestimmen. Für die Gleichung (1), die ein Hamiltonsystem definiert, ist zum Beispiel das Feld

$$\tau_1 := x(u_{xxx} + 6uu_x) + 4u_{xx} + 2u_x J(u) + 8uu$$

eine Mastersymmetrie. J steht dabei für Integration von $-\infty$ bis x. Interessant ist noch, daß durch die Angabe der Mastersymmetrie auch der invariante Tensor Φ leicht konstruiert ist. Man nimmt einfach eine Hamiltonformulierung des Systems, bildet die Lie-Ableitung des Hamiltonoperators in Richtung der Mastersymmetrie und erhält einen invarianten Tensor, der zweifach kovariant ist. Kombination des Hamiltonoperators mit diesem neuen Tensor führt dann zu Φ.
Auf der theoretischen Grundlage dieser Mastersymmetrien haben wir ein Programmpaket entwickelt, welches für nichtlineare partielle Differentialgleichungen in zwei unabhängigen Variablen, bei denen die Vektorfelder von polynomialem Typ sind, Algorithmen bereitstellt, die diese Systeme auf vollständige Integrabilität testen. Bei positivem Ausgang des Tests berechnet das Programmpaket die unendlichdimensionale Symmetriegruppe, und damit implizit die Erhaltungsgrößen, sowie die Winkel- und Wirkungsvariablen des dynamischen Systems.
Die Algorithmen wurden bisher in dem Formelmanipulationssystem MAPLE implementiert. Tests zeigten, daß dieses System in bezug auf Geschwindigkeit, aber auch in bezug auf die Ökonomie der Speicherverwaltung anderen Systemen überlegen ist.
Zur optimalen Implementierung der Algorithmen war eine Reihe schwieriger Probleme zu lösen. Zum Beispiel war ein effizienter Normalformenalgorithmus für die Elemente einer Integrodifferentialalgebra zu entwickeln. Dies entsprach der algorithmischen Lösung eines nicht ganz einfachen kontextsensitiven Sprachproblems. Die Algorithmen beruhen auf einer subtilen Gradierung, die ein sukzessives Abarbeiten endlichdimensionaler Projektionen von Lie-Algebren erlaubt. Durch die Gradierung war es möglich, einen Algorithmus zu konstruieren, der die Lösung von Kommutatorgleichungen erlaubt. Dies war ein nichttriviales Problem, da die zugrunde liegenden Lie-Algebren eine Vielzahl von Nullteilern, nämlich unendlich viele, enthalten.
Ein analoges Programmpaket, allerdings auf ganz anderen mathematischen Grundlagen basierend, wurde für nichtlineare dynamische Systeme auf mehrdimensionalen Gittern von Zhang Hongwei entwickelt. Ein weiteres Programmpaket für die Untersuchung vollständig integrabler Quantensysteme

wurde von Uwe Falck geschrieben. Es erlaubt die Untersuchung von Spin-Wechselwirkungen (im Falle von Spin=1/2), die polynomial von den Spinmatrizen abhängen.

3 Beispielsitzung

Wir wollen die Wirkungsweise unserer Werkzeuge für die Symmetriesuche erläutern. Als Beispiel wählen wir die Korteweg-de Vries-Gleichung (1) aus. Das zu untersuchende Vektorfeld K ist also

$$K_1 := u_{xxx} + 6uu_x.$$

Unser Programmpaket stellt ein Werkzeug GHO zur Verfügung, welches auf den Aufruf *GHO(A,K,R)* hin, ein Vektorfeld X bestimmt, dessen Term höchster Ordnung mit dem von R übereinstimmt, und dessen Kommutator mit K das Vektorfeld A ergibt. Die Unbestimmtheiten des dadurch definierten Divisionsproblems in der gewählten Projektion der Vektorfeld-Liealgebra (einer Algebra mit Nullteilern) drückt sich in den freien Konstanten des Ergebnisses aus. Wir erhalten den nächsten Symmetriegruppengenerator des Systems, sofern er existiert, also durch:

$$K_2 := GHO(0, K_1, u_{xxxxx}).$$

Die darin enthaltene Kenntnis über den Term höchster Ordnung u_{xxxxx} folgt leicht aus elementaren Strukturüberlegungen, oder auch durch den systematischen Einsatz des beschriebenen Programmpakets. Das Ergebnis ist in unserem Fall

$$K_2 := u_{xxxxx} + 10uu_{xxx} + 20u_{xx}u_x + 30uuu_x.$$

Zur Sicherheit rechnen wir den Kommutator mit K_1 aus. Dieser wird durch den Aufruf *COM*(K_1, K_2) berechnet und liefert natürlich 0. Will man einen weiteren Symmetriegruppengenerator erhalten, fährt man mit $K_3 := GHO(0, K_1, u_{xxxxxxx})$ fort und erhält

$$\begin{aligned} K_3 := {} & u_{xxxxxxx} + 14uu_{xxxxx} + 42u_{xxxx}u_x + 70uuu_{xxx} \\ & +70u_{xxx}u_{xx} + 70u_xu_xu_x + 280uu_xu_{xx} + 140uuuu_x. \end{aligned}$$

So könnte man nun fortfahren, um dann beliebig viele Generatoren zu erhalten. Dieses Verfahren liefert aber noch keine Rekursionsformel für die eventuell vorhandene unendlich-dimensionale Symmetriegruppe. Zum Auffinden einer solchen Rekursionsformel versuchen wir nun ein Vektorfeld τ_1 zu bestimmen, welches mit K_1 kommutiert, das Feld K_2 ergibt. Im Falle der vollständigen Integrabilität des Ausgangsfeldes wissen wir dann, daß τ_1 eine Mastersymmetrie ist. Aus allgemeinen Strukturüberlegungen weiß man weiterhin, daß τ_1 die Translationssymmetrie linear bricht. Also muß der Aufruf $\tau_1 := GHO(K_2, K_1, xu_{xxx})$ zum Erfolg führen. In der Tat er liefert

$$\tau_1 := x(u_{xxx} + 6uu_x) + 4u_{xx} + 2u_xJ(u) + 8uu.$$

Damit hat man nun die Generatoren der Symmetriegruppe rekursiv als

$$[\tau_1, K_1], \quad [\tau_1, \quad [\tau_1, K_1]], \quad [\tau_1, [\tau_1 \quad [\tau_1, K_1]]], \ldots$$

bestimmt. Ein gewisser Trick des Pakets liegt darin, daß die Arbeitsweise der Programme die Annahme der vollständigen Integrabilität voraussetzt; eine Annahme, die dann während der Laufzeit überprüft wird. Diese Strategie ist dafür verantwortlich, daß sehr schnelle Laufzeiten erzielt werden.
Interessiert man sich nun für das Auffinden weiterer invarianter Tensoren, so arbeitet man am besten mit einer einfachen Hamilton-Formulierung der Gleichung. Diese ist durch

$$u_t = D\nabla \int_{-\infty}^{+\infty} \frac{1}{2}(u_x^2 + u^3)dx$$

gegeben. Wobei D als zweifach kontravarianter Tensor, also als Operator vom Cotangentialbündel auf das Tangentialbündel angesehen wird. Die Lie-Ableitung dieses Tensors nach τ_1 ergibt dann

$$D^3 + 2Du + 2uD.$$

Wegen der Mastersymmetrieeigenschaft von τ_1 ist dies ebenfalls ein zweifach kontravarianter Tensor, der invariant unter dem Fluß der Gleichung (1) ist. Bildet man nun das Operatorprodukt dieses Tensors mit dem Inversen von D so erhält man einen invarianten Operator vom Tangentialbündel in das Tangentialbündel, dessen Lie-Ableitung in der Richtung von K_1 ebenfalls verschwinden muß. Dies ist natürlich unser eingangs erwähnter Operator, der dann in kanonischer Weise zu einem inversen Streuproblem führt. Das Streuproblem läßt sich sogar leicht in ein Streuproblem übersetzen, welches durch den üblichen Schrödinger-Operator definiert wird.
Beim Einsatz auf die Korteweg-de Vries-Gleichung liefern unsere Computeralgebrawerkzeuge natürlich keine neuen Erkenntnisse, hier war alles schon vorher bekannt. Wenn man sich aber für Vektorfelder höherer Ordnung interessiert, wird man *SymmPAD* als unschätzbares Werkzeug bei der Herleitung neuer Ergebnisse und beim Auffinden neuer vollständig integrabler Systeme kennenlernen.

Minimale aufspannende Bäume in der Euklidischen Ebene mit graphischem Optimalitätsbeweis

Michael Jünger und William R. Pulleyblank

Das Bild zeigt einen aufspannenden Baum minimaler Gesamtlänge auf 20 Punkten in der Euklidischen Ebene und gleichzeitig einen „graphischen Beweis" der Minimalität dieses Baumes.

Wir betrachten n durch kartesische Koordinaten gegebene Punkte $p_1 = (x_1, y_1), p_2 = (x_2, y_2), \ldots, p_n = (x_n, y_n)$ in der Euklidischen Ebene und identifizieren diese Punkte mit den Knoten eines vollständigen ungerichteten Graphen $G = (V, E_n)$ mit Knotenmenge $V = \{p_1, p_2, \ldots, p_n\}$ und Kantenmenge $E_n = \{\{p_i, p_j\} \mid i, j \in \{1, 2, \ldots, n\}, i \neq j\}$. Zusätzlich definieren wir für jede Kante $\{p_i, p_j\} \in E$ als Länge die Euklidische Distanz $d(p_i, p_j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$ ihrer Endpunkte. Für eine Kantenmenge $E \subseteq E_n$ definieren wir $V(E) = \bigcup_{e \in E} e$.

Ein Baum ist ein kreisfreier zusammenhängender Graph. Ein V aufspannender Baum ist ein Subgraph $G_T = (V, T)$ von G, der ein Baum ist und alle Knoten miteinander verbindet, d. h. $T \subset E_n$ und $V = V(T)$. Offensichtlich hat ein solcher aufspannender Baum $n - 1$ Kanten. Die Gesamtlänge des aufspannenden Baumes $G_T = (V, T)$ ist $z_T = \sum_{\{u,v\} \in T} d(u, v)$. Die Aufgabe, einen aufspannenden Baum minimaler Gesamtlänge zu bestimmen, ist effizient lösbar, sogar wenn man beliebige „Längen" für die Kanten zuläßt. Eine Möglichkeit ist die Verwendung eines Algorithmus von Kruskal aus dem Jahre 1956, der für einen beliebigen zusammenhängenden Graphen $G = (V, E)$ mit beliebigen Kantengewichten die Laufzeit $O(|E| \log |E|)$ hat. In unserem speziellen (Euklidischen) Fall können wir, ohne die Zeitkomplexität zu erhöhen, diesen Algorithmus so erweitern, daß er einen „graphischen Beweis" der Minimalität der Gesamtlänge gleich mitliefert.

Dazu wählen wir einen beliebigen Knoten $\hat{v} \in V$ und betrachten das folgende Paar zueinander dualer Linearer Optimierungsprobleme.

$$
\begin{array}{lll}
\text{(LP)} & \max \displaystyle\sum_{S \subseteq V,\, \hat{v} \notin S} 2\, w_S & \\
(1) & \displaystyle\sum_{S \subseteq V,\, |S \cap \{u,v\}| = 1} w_S \leq d(u, v) & \text{für alle } u, v \in V,\ u \neq v \\
(2) & \displaystyle\sum_{S \subseteq V,\, v \in S} w_S = \alpha & \text{für alle } v \in V \\
& w_S \geq 0 & \text{für alle } S \subseteq V \\
& \alpha \quad \text{unbeschränkt} &
\end{array}
$$

$$
\begin{array}{lll}
\text{(DLP)} & \min \displaystyle\sum_{u,v \in V,\, u \neq v} d(u, v) x_{\{u,v\}} & \\
(3) & \displaystyle\sum_{u,v \in V,\, u \neq v,\, |S \cap \{u,v\}| = 1} x_{\{u,v\}} + \sum_{v \in S} y_v \geq \begin{cases} 2 & \text{falls } \hat{v} \notin S \\ 0 & \text{falls } \hat{v} \in S \end{cases} & \text{für alle } S \subseteq V \\
& \displaystyle\sum_{v \in V} y_v = 0 & \\
& x_{\{u,v\}} \geq 0 & \text{für alle } u, v \in V,\ u \neq v \\
& y_v \ \text{unbeschränkt} & \text{für alle } v \in V
\end{array}
$$

Das Problem (DLP) ist das Problem der Bestimmung eines aufspannenden Baumes minimaler Gesamtlänge in einer etwas ungewöhnlichen Formulierung. Kruskal's Algorithmus mit leichten Modifikationen konstruiert Optimallösungen für beide Lineare Optimierungsprobleme.

Zunächst betrachten wir (LP). Kruskal's Algorithmus startet mit den n Bäumen, die aus jeweils einem der Knoten aus V bestehen. Beginnend mit der kürzesten Kante wird dann iterativ die kürzeste verbleibende Kante, die keinen Kreis produziert, hinzufügt, bis $n-1$ Kanten gewählt wurden. Wenn eine Baumkante t hinzugefügt wird, verbindet sie zwei Bäume G_{T_1} und G_{T_2}, die zuvor gebildet wurden.

Während dieses Prozesses konstruieren wir eine Lösung w für (LP). Am Anfang sei $w = 0$. In jeder Iteration gilt für alle Knoten v jedes konstruierten Baumes G_T die Beziehung $\sum_{v \in S} w_S = \alpha(G_T)$ wobei $\alpha(G_T)$ die halbe Länge einer längsten Kante in G_T ist, bzw. $\alpha(G_T) = 0$, falls G_T nur aus einem Knoten besteht.

Wenn wir eine Kante t hinzufügen, die die Knoten $u_1 \in G_{T_1}$ und $u_2 \in G_{T_2}$ verbindet, setzen wir $w(S_i) = \frac{1}{2}d(u_1, u_2) - \alpha(G_{T_i})$ für $i = 1, 2$, wobei $S_i = V(T_i)$. (Nach Definition sind diese Werte nichtnegativ.) G_T sei der neuproduzierte Baum, und wir setzen $\alpha(G_T) = \frac{1}{2}d(u_1, u_2)$. Offensichtlich erfüllt w bei Terminierung (1) und (2), wobei $\alpha = \alpha(G_T)$ gleich der halben Länge einer längsten Kante des minimalen aufspannenden Baumes G_T ist. Die Lösung w erfüllt (1) für jede Kante von G_T mit Gleichheit.

Nun konstruieren wir eine zulässige Lösung für (DLP). Sei $x_{\{u,v\}} = 0$, falls $\{u, v\}$ keine Kante von G_T ist, und sei $x_{\{u,v\}} = 1$, falls $\{u, v\}$ eine Kante von G_T ist. Wir orientieren alle Kanten von G_T in Richtung $\hat{v}$. Für jedes $v \in V$ sei y_v gleich der Anzahl der v verlassenden Kanten (in G_T) minus der Anzahl der in v endenden Kanten (in G_T). Durch Induktion auf T können wir beweisen, daß wir auf diese Art eine zulässige Lösung für (DLP) erhalten. Da $x_{\{u,v\}} = 1$ nur für Kanten in T gilt, ist (1) für jede solche Kante mit Gleichheit erfüllt. Außerdem können wir durch Induktion zeigen, daß (3) für alle $S \subseteq V$ mit Gleichheit erfüllt ist, falls $w_S > 0$.

Die auf diese Art konstruierten Lösungen von (LP) und (DLP) erfüllen also die Komplementären Schlupfbedingungen für Optimalität. Man kann die Optimalität der beiden Lösungen auch einfach mit Hilfe des Schwachen Dualitätssatzes der Linearen Optimierung nachweisen, indem man überprüft, daß die oben konstruierten Lösungen zulässig für (LP) bzw. (DLP) sind und die Zielfunktionswerte übereinstimmen.

Im Bild ist die Lösung w von (LP) so dargestellt, daß für jeden Knoten $v \in V$ mit $w_{\{v\}} > 0$ eine Kreisscheibe mit Radius $w_{\{v\}}$ um v gezeichnet wird und für jede Knotenmenge $S \subseteq V$, $|S| > 1$, mit $w_S > 0$ ein „Graben" mit Breite w_S gezeichnet wird.

Informal können wir den Optimalitätsbeweis anhand der Graphik also wie folgt führen:

1. Das verschachtelte System der Kreisscheiben und Gräben ist nicht überlappend, repräsentiert also eine zulässige Lösung für (LP).

2. Die blauen Linien repräsentieren offensichtlich einen aufspannenden Baum und liefern somit zusammen mit den wie oben definierten Werten für die Variablen y_v $(v \in V)$ eine zulässige Lösung von (DLP).

3. Das System der Kreisscheiben und Gräben hat keine schwarzen Lücken zwischen zwei durch eine Baumkante verbundenen Punkten.

Die obigen Überlegungen führen zu einem interessanten, wenn auch praktisch unbrauchbaren Verfahren zur Bestimmung eines minimalen aufspannenden Baumes in der Euklidischen Ebene (oder auch im dreidimensionalen Euklidischen Raum) auf einem hypothetischen Analogcomputer. Dieses Verfahren möchten wir schließlich noch grob skizzieren: Um jeden Punkt werden gleichmäßig Kreisscheiben „aufgeblasen", die wir der Einfachheit halber auch als (degenerierte) Gräben bezeichnen wollen. Sobald sich zwei Gräben berühren, entsteht eine Baumkante und ein neuer größerer Graben. Dies wird fortgesetzt, bis $n-1$ Baumkanten entstanden sind. Das Titelbild kann auch als Ergebnis eines solchen Prozesses interpretiert werden.

Die Idee zu dem Titelbild und die oben beschriebenen Konstruktionen entstammen einem in Vorbereitung befindlichen Aufsatz der Autoren mit dem Arbeitstitel „New primal and dual matching heuristics". In diesem Aufsatz werden die hier intuitiv verwendeten Begriffe wie z. B. „Graben" formal definiert. Die berechneten Lösungen von (LP) und (DLP) dienen hier zur effizienten Bestimmung von Euklidischen Matchings mit garantierter Qualität der Lösung.

Aufgelesen:

1 Kommunikation

Ein ... Videotelefon bietet die kürzlich gegründete Münchener GmbH des US-Unternehmens PictureTel Corporation nun auch in Deutschland an. Gemeinsam mit Kooperationspartner Siemens, der ISDN-fähige Telefonanlagen verkauft, die mit dem amerikanischen Videosystem kompatibel sind, überzeugte PictureTel die Deutsche Bundespost, einen neuen Standard zu akzeptieren. Das im Gegensatz zum normalen -analogen- Telefonnetz digital übermittelnde ISDN arbeitet mit einer Übertragungsrate von mindestens 128 Kilobit pro Sekunde, die die Post zunächst nur in zwei gleich große Kanäle spalten lassen wollte. Der neue Standard reserviert für die Sprache nur 16 Kilobit pro Sekunde, während die restlichen 112 Kilobit pro Sekunde für das Bild zur Verfügung stehen. Diese Aufteilung ist in den USA und Japan schon längst akzeptiert. Doch die geschickte Aufteilung der Datenströme reicht noch nicht aus, um bewegte Bilder in hoher Qualität zu übertragen. Dazu benötigt das System einen leistungsfähigen Computer, dessen 6 Prozessoren etwa 40 Millionen Rechenschritte pro Sekunde bewältigen. Außerdem wird der unbewegte Hintergrund nur einmal übertragen und beim Empfänger gespeichert.

Wirtschaftswoche 26 (21. Juni 1991)

A bid will be made to the UK government this autum to upgrade JANET, the academic computer network, from its present rate of 2 megabits s^{-1} to 100 megabits s^{-1} (Super-JANET).

Physics World, June 1991

2 Forschungsförderung

Zwischen 1984 und 1989 wurden durchschnittlich nur rund 60 000 Mark pro Jahr an Lizenzgebühren für Innovationen eingenommen, deren Entwicklung mit europäischem Forschungsgeld finanziert worden war. Dagegen steht die beachtliche Summe von rund 2 Milliarden Mark, die Brüssel Jahr für Jahr an Industrie und Forschungsinstitute verteilt. Diese Bilanz hat jetzt der EG-Rechnungshof aufgestellt. Aus den Unterlagen der EG-Dienststelle Förderung der Nutzung der Ergebnisse aus der gemeinschaftlichen Forschung und Entwicklung entnahm der Rechnungshof zudem, daß sich seit 1968 nur 50 Innovationen, die mit EG-Geldern entwickelt wurden, kommerzialisieren ließen. Dabei entstanden ganze 280 neue Arbeitsplätze.

Wirtschaftswoche 26 (21. Juni 1991)

Die Computer Zeitung: Eines der vom BMFT geförderten Großforschungsprojekte war Suprenum. Der Erfolg dieses Projektes ist weiterhin sehr umstritten. War Suprenum Ihrer Meinung nach ein Erfolg?
Riesenhuber: Als wissenschaftliches Forschungsvorhaben war Suprenum erfolgreich. Es ist gelungen, den Nachweis der Realisierbarkeit eines neuartigen Hochleistungs-Rechnersystems zu erbringen. Insbesondere im Bereich der Betriebs- und Anwendungssoftware hat Suprenum Wege aufgezeigt, die sich als richtungsweisend erwiesen haben.

Die Computer Zeitung Nr. 8 (27. März 1991)
Interview mit dem Bundesforschungsminister

Mit deutschem Super–Rechner gewaltig verkalkuliert: Ein Flop deutscher Technologie: Den Höchstleistungscomputer Suprenum mit der geplanten gigantischen Kapazität von bis zu fünf Milliarden Rechenoperationen je Sekunde wird es nicht geben. 162 Millionen Mark Steuergelder scheinen vertan, 40 Millionen Privat-Investitionen. Die vier bisher installierten Anlagen haben alle Aussicht, als Museumsstücke ins Kabinett für fehlgeschlagene staatliche Entwicklungsförderung zu wandern.

Frankfurter Rundschau 12.7.1990

3 Forschung + Hochschule

Cray-Award für das HLRZ: Anläßlich der internationalen Konferenz über Supercomputing in New York sind zwei Wissenschaftlergruppen aus dem Höchstleistungsrechenzentrum (HLRZ) und dem Zentralinstitut für Angewandte Mathematik des Forschungszentrums Jülich mit dem "Gigaflop Performance Award" der Firma Cray Research ausgezeichnet worden. Eines der ausgezeichneten Projekte befaßt sich mit der Quantenhadrodynamik (QHD). Joachim Frank vom Institut für Kernphysik der Universität Mainz und Siegfried Knecht (ZAM) erreichten durch Parallelisierung auf der Cray-Y-MP 8/832 eine

Rechenleistung von 2,13 GFLOPS. ... Dr. Edwin Laermann (HLRZ) sowie Siegfried Knecht und Wolfgang Nagel (ZAM) wurden für die Behandlung eines Problems zur numerischen Simulation der Quantenchromodynamik (QCD) ausgezeichnet.

Die Computer Zeitung Nr. 8 (27. März 1991)

Anfang Mai wurde zwischen dem Land Sachsen und der Siemens Nixdorf Informationssysteme AG (SNI), ein Vertrag über einen Vektorrechner in den neuen Bundesländern geschlossen. Mit dem Supercomputer VP200-EX erhält die TU Dresden einen leistungsstarken Rechner für technisch-wissenschaftliche Anwendungen in der ehemaligen DDR. Er bietet Wissenschaftlern und Studenten eine Spitzenleistung von 857 MFLOPS.

Die Computer Zeitung Nr. 13 (5. Juni 1991)

Transputer-Power für zwei Universitäten: Das nach Angaben größte freikonfigurierbare Parallelrechner−System Europas wurde in der Gesamthochschule Paderborn mit einem Festkolloquium eingeweiht. Das hier gegründete Paderborner Center für Paralleles Computing, kurz PC^2, ist ein offenes paralleles Rechenzentrum, das interdisziplinär sowohl von den technisch-wissenschaftlichen Bereichen der Hochschule als auch von externen Anwendern genutzt werden kann. ... Der zweite Hochleistungs-Parallelrechner mit 256 Transputern wurde im Rechenzentrum der RWTH Aachen in einem Festakt seiner Bestimmung übergeben. Die Deutsche Forschungsgemeinschaft DFG errichtet hier einen bundesweiten Schwerpunkt für Strömungssimulation mit Hochleistungsrechnern, der auch vom Land Nordrhein-Westfalen gefördert wird.

Die Computer Zeitung Nr. 13 (5. Juni 1991)

4 Software

The Norton Antivirus Survey revealed that computer virus infections increased from 3,000 in 1988 to 30,000 in 1990, which could push costs worldwide to £5 billion per year.

Informatika12, Vol1.1 (Spring 91)

Software companies in West Germany tend to be smaller and less active internationally than their European neighbours, which leads to doubts being cast upon their preparations for 1992.

Informatika12, Vol1.1 (Spring 91)

5 Produkte + Firmen

Der von den Beteiligten allzuoft wegdiskutierte Riß zwischen Microsoft und IBM scheint tiefer zu sein denn je. Oder warum sonst interessiert sich IBM für eine Lizenz des Apple-Betriebssystems und Apple seinerseits für die der RS/6000-Risk-Chip-Technologie von IBM? Künftig bekommt Intimfeind Apple von IBM sogar Festplattenlaufwerke.

Die Computerzeitung Nr. 14 (19. Juni 1991)

Frage: Wäre etwa eine industrielle Troika aus Apple, Sony und Disney denkbar?
Spindler: Nicht schlecht. In diesem Verbund könnte jeder seinen Teil dazu beitragen, eine aufregende und spannende neue Industrie zu schaffen. Aber solche Visionen unterstützen natürlich das etwas esoterische Image, das die Branche von uns hat, nach dem Motto: Die Kerle bei Apple haben nicht mehr alle. Trotzdem, ich wage zu prophezeihen, daß Kooperationen dieser Art kommen werden.

Wirtschaftswoche 24 (7. Juni 1991)
Inteview mit Michael H. Spindler
President und Chief Operating Officer
von Apple Computer

Love at First Byte: Giving up their blood rivalry, Apple and IBM join forces to develop bold new hybrids in personal computers... The two companies decided last week to put away their boxing gloves ... and share technology in a potentially powerful partnership that could reshape the computer industry.

Time International (July 15, 1991)

... the New Mac Portables: The original portable came with 1MB of static RAM standard on the motherboard. Because 1 MB was basically useless, Apple has now increased the standard configurations to 2 or 4 megabytes, but Apple has also changed the type of RAM. Instead of static RAM the new portable uses "low power", or pseudo-static, RAM that's cheaper than static RAMs but steals a tiny amount of power from the batteries.The new portable, which replaces the original model, still weighs 15 pounds, and the only thing about it that's smaller is the price: Apple has cut it by more than $ 1,000.

Mac User (June 1991)

Der angeschlagene amerikanische Computerproduzent Wang Laboratories Inc, Lowell (Massachusetts), hat mit dem Branchenführer IBM International Business Machines Corporation, Armonk (New York), ein Marketingabkommen geschlossen, demzufolge Wang in Zukunft IBM-Computer vertreiben wird.

Der Schritt könnte nach Ansicht von Fachleuten darauf hinauslaufen, daß Wang die eigene Herstellung von Computern weitgehend aufgibt und sich auf die Herstellung von Software sowie auf den IBM-Vertrieb spezialisiert.

FAZ (21. Juni 1991)

Der Goliath der DV-Branche, IBM, senkte Ende April die Preise für einige seiner Modelle der PS-Reihe um bis zu 34 Prozent. Jetzt, nur sechs Wochen später, kündigt das Unternehmen bereits eine erneute Preissenkung von diesmal bis zu 24 Prozent an. Der Gigant hat offensichtlich Schwierigkeiten, seine PS-Reihe zu vermarkten. Allein im ersten Quartal 1991 ging der Umsatz in diesem Segment um 15 Prozent zurück. Der David, der IBM das Fürchten lehrt, ist Apple.

Die Computer Zeitung Nr. 13 (5. Juni 1991)

Sun Microsystems konnte zum 29. März 1991 große Zuwachsraten sowohl für das dritte Quartal als auch für die ersten neun Monate des Geschäftsjahres 1991 verbuchen. Die Einnahmen im dritten Quartal beliefen sich auf 848,3 Millionen Dollar. Dies entspricht einer Steigerungsrate von 34 Prozent gegenüber 632,2 Millionen Dollar im gleichen Zeitraum des Vorjahres.

Die Computer Zeitung Nr. 13 (5. Juni 1991)

Apple-PC beim Discounter: Apple Computer Inc hat entschieden seine Personal Computer im Niedrigpreisbereich über Compusa Inc, die größte Discount-Kette für Computer in den USA, zu verkaufen, berichtete das "Wall Street Journal".

Die Computer Zeitung Nr. 13 (5. Juni 1991)

6 Chips und Halbleiter

Fünf der führenden Halbleiterhersteller haben unabhängig voneinander den ersten Prototyp einer neuen Generation von Super-Chips mit einer Speicherkapazität von 64 MB (64M DRAM) entwickelt. Mit der Ankündigung folgten Fujitsu Ltd, Masushita Electric Industrial Co, Mitsubishi Electric Corp und Toshiba Corp dem Vorreiter Hitachi Ltd, der erfolgreiche Entwicklungspläne in dem Sektor bereits im letzten Jahr bekanntgegeben hatte. Mit dem Beginn der Musterproduktion wird nicht vor Ende 1993/Anfang 1994 gerechnet. Die kommerzielle Fertigung kann schätzungsweise ein Jahr später anlaufen.

Die Computer Zeitung Nr. 10 (24. April 1991)

US-Chip-Umsatz auf Rekordhöhe: Die Auslieferungen der US-Halbleiterindustrie stiegen im März 1991 auf den Rekordwert von 1,409 Milliarden Dollar. Gegenüber Februar 1991 mit 1,2 Milliarden Dollar bedeutet dies eine Zunahme um 17,4 Prozent.

Die Computer Zeitung Nr. 10 (24. April 1991)

Aufbauend auf der gewonnenen Technologiebasis wird bei Siemens seit 1988 an der 16-Mbit-Generation gearbeitet, von der 1990 bereits Funktions-Muster an Kunden gegeben werden konnten. Seit April 1990 laufen auch die Vorarbeiten für das 64-Mbit-DRAM, das gemeinsam mit IBM in East Fishkill, USA, entwickelt wird. Dieser Speicher soll Mitte der 90er Jahre marktreif sein.

Die Computer Zeitung Nr. 8 (27. März 1991)

Irgendwann hat es dann in den letzten Jahren den Europäern einmal gedämmert, in welche Abhängigkeit sie hineingeraten sind. Die Antwort Europas war JESSI- und das war, zumindest in seiner noch bestehenden "Konstruktion",- leider die falsche Antwort.

Die Computer Zeitung Nr. 8 (27. März 1991)
Beitrag von Erich J. Lejeune

European companies have only a 10 % share of the world semiconductor market, compared with 52 % for Japanese manufacturers and 35 % for US companies. ... EEC imports of semiconductors are increasing, leading to a trade deficit of ECU 2,5 billion for the sector in 1989. Although the EC is promoting European cooperation programmes such as Jessi and Eureka, it has not proposed any real solution to the problem. ... Japanese companies will account for over 58 % of total turnover for the world's ten leading electronics companies in 1995, up from 37 % in 1989, according to Pierre Audoin, an analyst and Chairman of PAC. Moreover, he expects Japan to account for 47 % of the computer market, and Europe for just 7 %.

Informatika12, Vol1.1 (Spring 91)