Rechnerorientierte Verfahren

Von
Prof. Dr. phil. Bruno Buchberger, Universität Linz
Dipl.-Ing. Bernhard Kutzler, Universität Linz
Prof. Dr. rer. nat. Manfred Feilmeier, Institut für Wirtschafts- und Versicherungsmathematik G.m.b.H., München
Dr. rer. nat. Matthias Kratz, Technische Universität Braunschweig
Prof. Dr. rer. nat. Ulrich Kulisch, Universität Karlsruhe
Priv.-Doz. Dr. rer. nat. Siegfried Rump, Universität Karlsruhe und IBM Böblingen

B. G. Teubner Stuttgart 1986

CIP-Kurztitelaufnahme der Deutschen Bibliothek

Rechnerorientierte Verfahren / von Bruno Buchberger . . . – Stuttgart : Teubner, 1986.
(Mathematische Methoden in der Technik ; Bd. 4)
ISBN 3-519-02617-1
NE: Buchberger, Bruno [Mitverf.]; GT

Printed in Germany
Gesamtherstellung: J. Jllig, Göppingen
Umschlaggestaltung: M. Koch, Reutlingen

Parallele Numerik (M. Feilmeier)

Recherarithmetik und die Behandlung algebraischer Probleme
(U. Kulisch / S.M. Rump)

COMPUTER-ALGEBRA FÜR DEN INGENIEUR

Bruno Buchberger, Bernhard Kutzler
(Institut für Mathematik, Johannes-Kepler-Universität, Linz, Österreich)

1 Was ist Computer-Algebra?

Numerische Mathematik befaßt sich mit algorithmischen (d.h. mit dem Computer realisierbaren) Verfahren für die Behandlung von Problemen, deren Angaben *(Gleitkomma-) Zahlen* sind und deren Lösungen wieder (Gleitkomma-) Zahlen sind.

Computer-Algebra (Symbolic and Algebraic Computation, Formelmanipulation, Buchstaben-Rechnen mit dem Computer) befaßt sich mit algorithmischen Verfahren für die Behandlung von Problemen, deren Angaben *algebraische Objekte* sind und deren Lösungen wieder algebraische Objekte sind.

Die *algebraischen Objekte*, auf denen die Probleme definiert sind und die Lösungsalgorithmen arbeiten, liegen in verschiedenen *algebraischen Bereichen*. Ein algebraischer Bereich ist dabei durch die Menge der Objekte, die zu ihm gehören, (den *"Träger"* des Bereiches) und die *Grundoperationen*, die man auf den Objekten ausführen kann, definiert. Beispiele algebraischer Bereiche sind:

Die *natürlichen, ganzen* und *rationalen Zahlen* ("exakt", d.h. ohne Rundungsfehler dargestellt; mit den bekannten Grundoperationen "Addition", "Subtraktion", "Multiplikation", etc.)

Die *Gauß'schen Zahlen* (d.h. die rationalen Zahlen erweitert um die imaginäre Einheit i; mit entsprechend erweiterten Grundoperationen, z.B. $(1+2i).(\frac{2}{3}-3i) = 20/3 - 5/3\ i$).

Andere *"algebraische" Erweiterungen* der rationalen Zahlen (z.B. der Bereich, der entsteht, wenn man zu den rationalen Zahlen die reellen Zahlen $\sqrt{2}$, $\sqrt{7}$ und noch einige andere Quadratwurzeln oder höhere Wurzeln - als neue Objekte, nicht als rationale Näherungszahlen - hinzufügt und die Grundoperationen entsprechend erweitert, z.B. $\sqrt{2}.\sqrt{3} = \sqrt{6}$.)

Endliche Körper (z.B. der endliche Körper "Z modulo 5", der aus den Zahlen 0,1,2, 3,4 besteht mit den Operationen z.B. $2\oplus4=1$, $3\oplus2=0$ etc., allgemein $x\oplus y :=$ Rest bei der Division von $x+y$ durch 5.)

Der Bereich der *univariaten und multivariaten Polynome* über obigen Zahlbereichen, d.h. der Ausdrücke der Form $3x^3-\frac{1}{2}x+1$ oder $\frac{3}{4}x^3yz^2-\frac{1}{3}xy^2z+13xz$ etc. mit den Operationen $(3x^3-\frac{2}{3}x+1)\oplus(-2x^3+5)=(x^3-\frac{2}{3}x+6)$ etc.

Der Bereich der *rationalen Terme* über obigen Zahlbereichen, d.h. der "gekürzten" Ausdrücke der Form $(5x^2yz-xy+1)/(xz-\frac{2}{3}x)$ mit den entsprechenden Operationen wie z.B. $(x+1)/(x-1) \oplus (x-1)/(x+1) = (2x^2+2)/(x^2-1)$.

Der Bereich der *arithmetischen Terme* über obigen Zahlbereichen, d.h. der Ausdrücke der Form $(x.y.2+3.(0+x.y.z)).(x-z)$. (Beachte: "Polynome" sind spezielle arithmetische Terme, nämlich "vollständig ausmultiplizierte und vereinfachte".)

Der Bereich der *elementaren transzendenten Terme*, das sind Terme, in welchen auch die Symbole exp, log, sin, cos,... für die entsprechenden transzendenten Funktionen vorkommen können.

Gruppen (die z.B. durch eine endliche Zahl von erzeugenden Elementen und "definierenden Relationen" gegeben sind) mit der Gruppenverknüpfung als Grundoperation.

Differentialkörper, das sind Körper, in denen zusätzlich noch eine Operation " ' " ("Ableitung") definiert ist, für welche gilt: $(a+b)'=a'+b'$, $(a.b)'=a.b'+a'.b$. (In Differentialkörpern lassen sich die Probleme, die beim "analytischen", "symbolischen" Integrieren etc. auftreten, algebraisch fassen.)

Restklassenbereiche der obigen Bereiche modulo "Kongruenzrelationen". (Die Bildung von Restklassenbereichen ist eine sehr allgemeine und mächtige Konstruktion, um nützliche Bereiche zu gewinnen. Die meisten der oben angeführten Bereiche sind selbst durch Restklassenbildung aus einfacheren Bereichen entstanden: z.B. der Bereich der Polynome aus dem Bereich der arithmetischen Terme oder z.B. viele Gruppen aus Wortbereichen über endlichen Alphabeten.)

Nicht alle in der Mathematik studierten algebraischen Bereiche können im Computer behandelt werden. Für gewisse algebraische Bereiche kann man nicht einmal eine Darstellung aller Objekte des Bereichs angeben, auch wenn man einen sehr schwachen Begriff der "Darstellung" zuläßt. (Man kann z.B. den Bereich der reellen Zahlen nicht einmal aufzählen, schon gar nicht durch einen Algorithmus. Andere Bereiche sind zwar so einfach, daß man sie algorithmisch aufzählen kann, diese Aufzählung ist aber grundsätzlich nicht eindeutig. Man kann von zwei Darstellungen von Objekten nicht algorithmisch feststellen, ob sie dasselbe Objekt des darzustellenden mathematischen Bereiches darstellen. Ein einfacher Bereich dieser Art ist z.B. der Bereich der Funktionen, die im wesentlichen durch die elementaren transzendenten Terme und die Absolutstriche dargestellt werden können. Seine algorithmische "Unbehandelbarkeit" wurde von /Caviness 1967,1970/ gezeigt.)

Die *erste Klasse von Problemen*, die in der Computer-Algebra vor allen anderen behandelt werden müssen, ist deshalb:

Die Darstellung der Objekte algebraischer Bereiche im Computer, insbesondere die Vereinfachung (*Simplifikation*) von möglichen Darstellungen eines Objektes auf eine Standardform (*kanonische*, eindeutig bestimmte Form).

Die Konstruktion von Algorithmen, mit denen die *Grundoperationen* auf den (Darstellungen der) Objekte im Computer ausgeführt werden können.

Die *Übersetzung* von verschiedenen Darstellungen ein und desselben Bereiches ineinander.

Typische *andere Probleme*, die in der Computer-Algebra behandelt werden, sind z.B.:

Die *Dekomposition* von Objekten in einfachere, bzw. die Frage der *Dekomponierbarkeit*. (Z.B. die *Primzahlzerlegung* von natürlichen Zahlen, die *Faktorisierung* von Polynomen in irreduzible Faktoren, die Darstellung von Funktionen durch *Hintereinanderausführung* einfacherer Funktionen etc.).

Das Auffinden von *gemeinsamen "Teilobjekten"* oder *gemeinsamen "Oberobjekten"* gegebener Objekten. (Z.B. Bestimmung des größten gemeinsamen Teilers

von multivariaten Polynomen; Bestimmung allgemeinster Unifikatoren von Termen, d.h. von Substitutionen, die zwei gegebene Terme identisch machen etc.).

Die (exakte!) Lösung von *Gleichungen und Ungleichungen* in algebraischen Bereichen. (Z.B. lineare und nicht-lineare polynomiale Gleichungen mit rationalen Koeffizienten und beliebig vielen Unbekannten; boole'sche Gleichungen; lineare Gleichungen mit Koeffizienten, die selbst Polynome sind; Gleichungen über Gruppen, Lie-Algebren, etc.).

Die algorithmische Bestimmung von Objekten, die durch *höhere Operationen* in den jeweiligen Bereichen definiert sind. (Z.B. die Bestimmung von elementaren transzendenten Termen, die den *Limes*, die *Ableitung*, das *Integral* der durch einen gegebenen elementaren transzendenten Term dargestellten Funktion darstellen; die Bestimmung von Termen, die die endliche bzw. unendliche *Summe* bzw. das *Produkt* von durch Terme dargestellten Funktionen darstellen).

Die Lösung von *Funktionalgleichungen* über gewissen Bereichen. (Z.B. das Auffinden von Termen, die die Lösungsfunktionen einer Differentialgleichung darstellen).

Die *Analyse* der *Struktur algebraischer* Bereiche. (Z.B. Auffinden des Verbands aller Normalteiler einer gegebenen Gruppe; Auffinden der Primärzerlegung eines Polynomideals, das entspricht der Zerlegung beliebiger höherdimensionaler algebraischer Mannigfaltigkeiten in einfachste geometrische Grundgebilde).

2 Was bringt Computer-Algebra für den Ingenieur?

Die Verfügbarkeit von Algorithmen zur Lösung algebraischer Probleme in benutzerfreundlichen Computer-Algebra-Softwaresystemen ist ein Ergebnis intensivster mathematischer und softwaretechnologischer Forschung in den letzten 25 Jahren. Dies eröffnet für den Ingenieur einen *grundsätzlich neuen und weiten Bereich*, in welchem seine Tätigkeit durch den Computer unterstützt und damit produktiver, genauer und schneller gemacht werden kann. Durch das Zusammenspiel von

Software-Systemen zum numerischen Rechnen (Methodenbanken, Algorithmenbibliotheken),
Software-Systemen der Computer-Algebra,
Software-Systemen zum Graphischen Rechnen (Computergraphik- und CAD-Syteme),
Software-Systemen zur Teilautomatisierung des Beweisens,
Software-Systeme zur Unterstützung des Software-Entwurfs-Prozesses und
Software-Systeme zum Ziehen von Schlüssen aus großen technischen Datenbanken (Expertensysteme)

entstehen in unserer Zeit mächtige Werkzeuge, die den *Arbeitsplatz des Ingenieurs der Zukunft* drastisch verändern werden, nämlich hin zu einer Befreiung des Ingenieurs von immer mehr der Routinearbeit, sodaß er sich immer mehr auf den kreativen Teil seiner Arbeit konzentrieren kann.

Computer-Algebra, algorithmische Geometrie, automatisches Beweisen und automatisches Programmieren verwachsen immer mehr zu einem Gebiet, nämlich "Symbolic Computation" (siehe die gleichnamige Zeitschrift, insbesondere das Editorial im Heft 1/1 dieser Zeitschrift). Gemeinsam mit "Numerical Computation" wird "Symbolic Computation" zum Gesamtgebiet des "Scientific Computation", das eine neue Dimension des Computer-unterstützten Problemlösens im technischen Bereich ermöglichen wird.

Für die praktischen Bedürfnisse des Ingenieurs ist es zunächst ausreichend, wenn er über die Existenz des neuen, mächtigen Werkzeuges, das die Computer-Algebra-Software-Systeme darstellen, informiert ist, und weiß, für welche Probleme diese Systeme fertige Lösungen anbieten und wie man mit ihnen umgeht. Diesem Zweck sind die nächsten beiden Abschnitte über *"Die Problemlösepotenz von Computer-Algebra-Systemen: Einige Beispiele"* und *"Übersicht über wichtige Computer-Algebra-Systeme"* gewidmet. Nach einigen praktischen Experimenten mit solchen Systemen entsteht aber bei den meisten Benutzern auch der Wunsch besser zu verstehen, "was hinter diesen Systemen steht". Damit werden auch geschicktere Anwendungen möglich und die Systeme können für den eigenen Gebrauch erweitert oder verbessert werden. Deshalb ist in diesem Kapitel noch ein längerer Abschnitt über *"Computer-Algebra-Algorithmen"* eingeschlossen. Den Abschluß des Kapitels bildet eine Übersicht über die Computer-Algebra Literatur.

3 Die Problemlösepotenz von Computer-Algebra-Systemen: Einige Beispiele

Im folgenden werden einige mit bestehenden (und im Abschnitt *Computer-Algebra-Systeme* genauer beschriebenen) Computer-Algebra-Systemen durchgeführten Berechnungen wiedergeben, die einen ersten Eindruck von den Fähigkeiten dieser Systeme geben sollen.

Fast alle Computer-Algebra-Systeme arbeiten *interaktiv*, d.h. in Form eines Dialoges mit dem Benutzer. Dabei wiederholt sich nach dem Start des Systems bis zum Ende der "Sitzung" immer wieder der Zyklus

Eingabe - Auswertung - Ausgabe.

Auf jede Eingabe des Benutzers erfolgt eine Auswertung dieser Eingabe (d.h. die Berechnung der gesuchten Lösung) und nachfolgend die Ausgabe des Ergebnisses. Dann wird wieder eine Eingabe erwartet usf.

Im Gegensatz dazu müssen bei Systemen, die im *Batch-Betrieb* arbeiten, alle Berechnungswünsche *vor* dem Start des Systems in eine Datei geschrieben werden. Beim Aufruf des Systems muß dieser Datenbestand als Eingabedatei mitgegeben werden, die Ergebnisse aller Berechnungen werden auf einer Ausgabedatei gesammelt, in die man *nach* Fertigstellung *aller* Berechnungen einsehen kann.

Der Vorteil von interaktiv arbeitenden Systemen gegenüber im Batch-Betrieb arbeitenden Systemen liegt vor allem darin, daß man bei ersteren unmittelbar auf das Ergebnis einer Berechnung reagieren kann, um die nachfolgenden Anweisungen (Berechnungswünsche) an das System dementsprechend zu wählen. Interaktive Systeme genügen also viel eher den Anforderungen für die Computerunterstützung des *unmittelbaren* (kreativen) mathematischen Problemlösens.

Wir geben zunächst eine größere Anzahl von typischen Grundrechenoperationen, die mit Computer-Algebra-Systemen ausgeführt werden können. Dann präsentieren wir vier durch bestimmte Ingenieursanwendungen motivierte Beispiele (Erzeugung von Unterprogrammen für das Newton-Verfahren; Roboterkinematik; Verhalten elektrischer Schaltungen; geometrisches Beweisen). Eine große Fülle weiterer Anwendungsbeispiele von Computer-Algebra-Systemen aus dem Ingenieurbereich finden

sich in den Beispielsammlungen /Rand 1984/, /Pavelle 1985/ und ab 1985 regelmäßig in der Sektion "Application Letters" des Journal of Symbolic Computation.

3.1 Beispiele zu typischen symbolischen Grundoperationen

Die meisten in diesem Abschnitt angegebenen Beispiele benötigen nur wenige Sekunden Rechenzeit. Für die aufwendigeren Beispiele geben wir die Rechenzeiten explizit an.

Der folgende Dialog wurde mit dem auf Personal Computern verfügbaren System muMATH-83 geführt.

```
? 4/5 - (1/2+6/7)*(2/3)^3;

@: 376/945
```

In diesem System steht vor jeder Eingabe ein Fragezeichen und vor jeder Ausgabe ein "Klammeraffe" ("@"). Jede Eingabe muß mit einem Terminationszeichen (";" oder "$") abgeschlossen werden. "$" bewirkt eine Unterdrückung der Ausgabe des Ergebnisses. Computer-Algebra-Systeme rechnen grundsätzlich mit rationaler Arithmetik (d.h. mit Brüchen statt mit Gleitkommazahlen) und dadurch exakt. Man kann sich aber auf Wunsch die Ergebnisse dezimal mit wählbarer Genauigkeit (im Beispiel auf 40 Stellen nach dem Komma) ausgeben lassen. Die Rechnung erfolgt weiterhin mit rationalen Zahlen. (Zunächst weisen wir jedoch noch die letzte Ausgabe der Variablen T zu.)

```
? T: @ $  POINT: 40 $  T;

@: 0.3978835978835978835978835978835978835978
```

Man kann mit beliebig langen ganzen Zahlen rechnen. Im folgenden wird 200! berechnet (n! = 1.2.3.....(n-1).n):

```
? 200!;

@:
78865786736479050355236321393218506229513597768717326329474253324435944996340334
29203042840119846239041772121389196388302576427902426371050619266249528299311134
62857270763317237396988943922445621451664240825403329186413122742829485327752424
20757390324032125740557956866022603190417032406235170085879617892222278962370389
73747200000000000000000000000000000000000000000000000000
```

Eine weitere Fähigkeit solcher Systeme ist die, mit "Buchstaben" zu rechnen.

```
? EXPAND((2 X - 3 Y^2)^3);
```

```
@: 54 X Y^4 - 36 X^2 Y^2 + 8 X^3 - 27 Y^6;
```

(Es genügt, statt des Zeichens "*" für die Multiplikation ein Leerzeichen zu schreiben ebenso, wie sich das auch für das händische Rechnen eingebürgert hat. EXPAND ist eine eingebaute Funktion, die den Argumentterm vollständig ausmultipliziert.)

Beim Buchstabenrechnen stellt sich häufig das Problem des *Simplifizierens*, d.h. des Vereinfachens von Termen. Ziel ist es dabei, zu einem gegebenen Term einen möglichst "einfachen" aber äquivalenten Term zu finden. Etwas Ähnliches wurde bereits im ersten Abschnitt erwähnt als das Problem des kanonischen Simplifizierens, d.h. der Berechnung einer eindeutigen Darstellung für ein Objekt eines algebraischen Bereiches. Jedoch muß eine solche kanonische Normalform, falls sie existiert, nicht unbedingt "einfach" im Sinn einer durch eine konkrete Anwendung vorgegebenen Kostenfunktion sein. (Eine sinnvolle Kostenfunktion könnte etwa die Länge eines Termes sein). Z.B. wäre für Polynome die sogenannte *distributive Darstellung* (d.h. die "voll ausmultiplizierte" Form) eine kanonische Normalform, aber $(x-1)^5$ würde im allgemeinen als "einfacher" als $x^5 - x^4 + x^3 - x^2 + x - 1$ betrachtet werden. Computer-Algebra-Systeme haben im allgemeinen eine Reihe von *Simplifikatoren* eingebaut, die für viele Klassen von Termen für die Praxis ausreichend "einfachste" Formen berechnen. Auf Wunsch können aber auch Teilschritte von Simplifikatoren (Anwendungen von Rechengesetzen) vom Benutzer einzeln aufgerufen werden.

```
? TRGSIMP( SIN(X) COS(X) ( TAN(X) + COT(X) );

@: 1

? TRGSIMP(1 - SIN(X)/(1 + COT(X)^2);

@: 1 - SIN(X)^3
```

Eine weitere Klasse von Aufgaben sind das symbolische Differenzieren und Integrieren sowie das symbolische Lösen von (Gleichungen und) Differentialgleichungen. Löst man Probleme dieser Art, d.h. Probleme, bei denen eine Funktion gesucht ist, mit rein numerischen Methoden, so erhält man als Resultat eine Folge von Zahlen, die mehr oder weniger gute *Näherungen der Funktionswerte* der tatsächlichen Lösungsfunktion an diskreten Stellen sind. Um dann Aussagen über die eigentliche Lösung machen zu können, müssen diese Zahlen "interpretiert" werden, wobei jedoch im allgemeinen sehr wesentliche Informationen über mathematische Eigenschaften der Lösungsfunktion aus den Zahlen nicht mehr wiedergewonnen werden können. Es ist oft wünschenswert, die Lösung *analytisch*, d.h. in Form eines die Lösungsfunktion beschreibenden Termes zu kennen. ("Der Sinn des Rechnens

sind nicht die Zahlen, sondern Einsicht!") Dieses Ziel kann nun in vielen Fällen durch Computer-Algebra-Systeme erreicht werden.

Die folgenden Beispiele aus /Pavelle,Wang 1985/ und /Fateman 1981/ wurden mit MACSYMA, einem der derzeit größten und leistungsfähigsten Systeme, auf einer Symbolics 3600 LISP-Maschine bzw. einer DEC VAX 11/780 gerechnet. In MACSYMA werden alle Eingaben und Ausgaben durchnumeriert, damit sie an beliebiger Stelle referenziert werden können. Alle Eingaben fangen mit dem Buchstaben "C", alle Ausgaben mit "D" an. Außerdem erfolgt die Ausgabe in einem sogenannten "Prettyprint"-Format, das der menschlichen (zweidimensionalen) Schreibweise mathematischer Formeln entspricht.

Zunächst das Lösen einer kubischen Gleichung mit Parametern:

```
(C1) SOLVE(X^3+B*X^2+A^2*X^2-9*A*X^2+A^2*B*X-2*A*B*X-9*A^3*X
       +14*A^2*X-2*A^3*B+14*A^4=0,X);

                                  2
(D1)   [X=7 A - B, X=-A , X=2 A]
```

Das folgende Polynom in vier Variablen wird in etwa 112 Sekunden faktorisiert (siehe Abschnitt *Computer-Algebra-Algorithmen* zum Problem des Faktorisierens):

```
(C2) FACTOR(-36*W^2*X^7*Y^4*Z^8+3*W^2*X^6*Y^3*Z^8-24*W^3*X^7*Y^4*Z^6+
  2*W^3*X^6*Y^3*Z^6+96*W^2*X^8*Y^6*Z^5-168*W^4*X^7*Y^6*Z^5+
  12*W^2*X^7*Y^6*Z^5-216*W^2*X^10*Y^5*Z^5-8*W^2*X^7*Y^5*Z^5+9*X^7*Y^5*Z^5+
  14*W^4*X^6*Y^5*Z^5-W^2*X^6*Y^5*Z^5+18*W^2*X^9*Y^4*Z^5+87*X^7*Y^3*Z^5-
  3*W^2*X^6*Y^3*Z^5+6*W*X^7*Y^5*Z^3+58*W*X^7*Y^3*Z^3-2*W^3*X^6*Y^3*Z^3-
  24*X^8*Y^7*Z^2+42*W^2*X^7*Y^7*Z^2-3*X^7*Y^7*Z^2+54*X^10*Y^6*Z^2-
  232*X^8*Y^5*Z^2+414*W^2*X^7*Y^5*Z^2-29*X^7*Y^5*Z^2-14*W^4*X^6*Y^5*Z^2+
  W^2*X^6*Y^5*Z^2+522*X^10*Y^4*Z^2-18*W^2*X^9*Y^4*Z^2);

       6 3 2   3              2       2 2   2       3
(D2)  -X Y Z (3Z +2WZ-8XY +14W Y -Y +18X Y)
                                2    3   2 3      2          2
                        (12W XYZ -W Z -3XY -29X+W )
```

Das symbolische Differenzieren und Integrieren gehört zu den Standardanforderungen an universelle Computer-Algebra-Systeme. Das symbolische Differenzieren bereitet keine algorithmischen Schwierigkeiten. Hier kommt es fast ausschließlich auf den Einsatz guter Simplifikatoren zur Vereinfachung der entstehenden Resultate an. Das symbolische Integrieren wurde in befriedigender Form jedoch erst durch die Entdeckung von sehr viel neuem mathematischen (algorithmisch brauchbaren) Wissen ermöglicht (Algorithmus von /Risch 1970/).

```
(C3) DIFF(X^X^X,X);
```

```
                 X
               X   X                            X - 1
(D3)           X   (X  LOG(X) (LOG(X) + 1) + X       )
```

(C4) INTEGRATE((LOG(X) - 1)/(LOG(X)^2-X^2);

```
              LOG( LOG(X) + X )    LOG( LOG(X) - X )
(D4)          -----------------  -  -----------------
                      2                     2
```

(C5) INTEGRATE(1/(X^3+2),X);

```
                                                  1/3
                                              2 X - 2
                                        ATAN(-----------)
                  2   1/3     2/3                1/3                  1/3
             LOG(X  - 2   X + 2    )            2    √3      LOG(X + 2    )
(D5)       - ------------------------- + ------------------- + ---------------
                         2/3                    2/3                   2/3
                      6 2                    2     √3              3 2
```

Es können nicht nur unbestimmte, sondern auch manche bestimmte Integrale exakt berechnet werden. Das folgende uneigentliche Integral wurde in etwa 138 Sekunden gefunden (%E steht für die Euler'sche Zahl e, %PI für π, %GAMMA für die Euler-Mascheroni Konstante 0.577215664....; INF bezeichnet die obere Grenze ∞):

(C6) INTEGRATE(X^2*%E^(-U*X^2)*LOG(X),X,0,INF);FACTOR;

```
          SQRT(%PI) (LOG(U) + 2 LOG(2) + %GAMMA - 2)
(D6)    - -------------------------------------------
                             3/2
                          8 U
```

Im folgenden demonstrieren wir die Anwendung eines Paketes zum Lösen von Systemen von gekoppelten Differentialgleichungen (Anfangswertproblemen) unter Verwendung der Laplace-Transformation. (Dabei werden mit ATVALUE die Anfangsbedingungen festgelegt.):

(C7) EQ1: 3*'DIFF(F(X),X,2) - 2*'DIFF(G(X),X) = SIN(X);

```
(D7)          3 F(X)      - 2 G(X)    = SIN(X)
                    X X           X
```

(C8) EQ2: A*'DIFF(G(X),X,2) + 'DIFF(F(X),X) = A*COS(X)

```
(D8)          A G(X)      + F(X)    = A COS(X)
                    X X         X
```

(C9) ATVALUE(G(X), X=0, 1) $

(C10) ATVALUE('DIFF(F(X),X), X=0, 0) $

(C11) ATVALUE('DIFF(G(X),X), X=0, 1) $

(C12) EQ1: LAPLACE(EQ1,X,S);

```
                 2                                                              1
(D12)       3 (S  LAPLACE(F(X),X,S) - F(0) S) - 2 (S LAPLACE(G(X),X,S) - 1 ) = ------
                                                                                2
                                                                               S  + 1
```

(C13) EQ2: LAPLACE(EQ2,X,S);

```
                 2                                                           A S
(D13)       A (S  LAPLACE(G(X),X,S) - S - 1) + S LAPLACE(F(X),X,S) - F(0) = ------
                                                                             2
                                                                            S  + 1
```

Diese nun entstandenen linearen Gleichungen für LAPLACE(F(X),X,S) und LAPLACE(G(X),X,S) können mit der Funktion LINSOLVE gelöst werden:

(C14) LINSOLVE([EQ1,EQ2],['LAPLACE(F(X),X,S),'LAPLACE(G(X),X,S)]);

```
                                         4                               2
                               3 F(0) A S  + (F(0) (3 A + 2) + 2 A) S  + 3 A S + 2 A + 2 F(0)
(D14)       LAPLACE(F(X),X,S) = ---------------------------------------------------------------
                                                     5              3
                                                3 A S  + (3 A + 2) S  + 2 S

                                          4         3              2
                                     3 A S  + 3 A S  + (6 A + 2) S  + 3 A S + 1
            LAPLACE(G(X),X,S) = ------------------------------------------------
                                               5              3
                                          3 A S  + (3 A + 2) S  + 2 S
```

Man beachte dabei, daß für F(0) kein Wert spezifiziert wurde, F(0) also in die Lösung als Parameter eingeht. Schließlich werden durch die inverse Laplace Transformation ILT die Lösungen für f und g berechnet.

(C15) MAP(LAMBDA([A],ILT(A,S,X)),"%);

```
(D15) IS A POSITIVE, NEGATIVE, OR ZERO?
      POSITIVE;
                         5/2      √6 X
                     27 A    SIN(---------)
                                  3 √A           2        √6 X
                     ---------------------- - 3 A  COS(---------)
                          √6 (3 A - 2)                    3 √A        3 A SIN(X)
         [F(X) = ------------------------------------------------ - ------------ + A + F(0)),
                                      3 A                              3 A - 2

                        3/2      √6 X            2         √6 X
                     9 A    SIN(---------)   27 A   COS(---------)
                                 3 √A                     3 √A
                     -------------------- + ----------------------
                             √6                    6 A - 4              (3 A + 1) COS(X)     1
         G(X) = ---------------------------------------------------- - ------------------ + ---]
                                      3 A                                   3 A - 2           2
```

Zum Abschluß noch eine gewöhnliche Differentialgleichung, die mit der eingebauten Funktion ODE (für "ordinary differential equation") gelöst wird. (Mit DEPENDS erklärt man Y als von X abhängig. %K1 und %K2 bezeichnen zwei bei der Lösung entstehende Integrationskonstante):

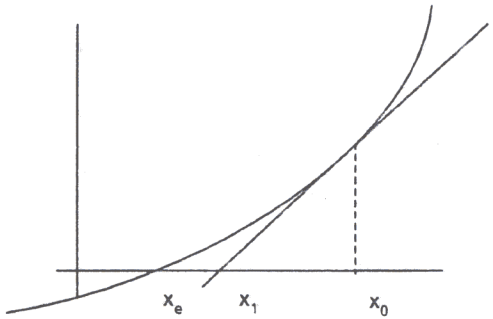
```
(C16) DEPENDS(Y,X)$

(C17) (1+X^2)*DIFF(Y,X,2)-2*Y=0;

                    2
(D17)          (X  + 1) Y     - 2 Y = 0
                         X X

(C18) ODE(D17,Y,X);

                                    2       ATAN(X)        X                2
(D18)          Y = %K2 (X  + 1) (------------ + ----------) + %K1 (X  + 1)
                                       2          2
                                              2X  + 2
```

3.2 Beispiel: Automatische Generierung eines Unterprogramms für das Newton-Verfahren

Im folgenden Abschnitt soll anhand des Problems der Bestimmung einer Nullstelle einer (reellen) Funktion das Zusammenspiel von Computer-Algebra und Numerik demonstriert werden. Das Ziel sei die Entwicklung eines FORTRAN-Programmes für das Newton-Verfahren. Dabei tritt als Unterproblem die Bestimmung von Ableitungen gegebener Funktionen auf. Mit vielen Computer-Algebra-Systemen kann man nicht nur die gesuchten Ableitungen bestimmen, sondern sie sogar als FORTRAN-Code ausgeben. Im folgenden ist nur die Lösung dieses Unterproblems (die /Lichtenberger 1981/ entnommen wurde) ausgeführt.

Das aus der numerischen Mathematik bekannte (eindimensionale) Newton-Verfahren gestattet es, zu einer gegebenen (Gleitkomma-) Zahl x_0 (die als "Näherungslösung" dient), einer Genauigkeitsschranke ε und einer reellen Funktion f (die "gewisse Eigenschaften" erfüllen muß) eine (Gleitkomma-) Zahl x' zu berechnen, sodaß $|f(x')|<\varepsilon$, d.h. x' ist im Rahmen der gewünschten Genauigkeit eine Nullstelle von f.

Der Grundgedanke des Newton-Verfahrens ist der folgende:



wenn f "gewisse Eigenschaften" erfüllt, dann ist der Schnittpunkt x_1 der Tangente an f in x_0 mit der x-Achse näher bei der exakten Nullstelle x_e als x_0.

Dies führt zum folgenden Algorithmus :

Ein-dimensionales Newton-Verfahren:

$x := x_0$
while $|f(x)|>\varepsilon$ **do**
    $x := x - f(x)/f'(x)$
$x' := x$

Dieses Verfahren läßt sich auf n Gleichungen in n Variablen verallgemeinern:

n-dimensionales Newton-Verfahren:

$x := x_0$
while $\|f(x)\|>\varepsilon$ **do**
    h := so, daß $D(f,x).h = f(x)$
    $x := x - h$
$x' := x$

Hier bezeichnet f eine n-dimensionale, n-wertige Funktion (d.h. f: $\mathbb{R}^n \to \mathbb{R}^n$), x_0, x, h sind Spaltenvektoren und D(f,x) ist die sogenannte "Jacobi-Matrix" oder "Funktional-Matrix" für f an der Stelle x. Sie ist wie folgt definiert:

$D(f,x)_{i,j} = (\partial/\partial x_j) f_i(x)$,
wobei $(\partial/\partial x_j) f_i$ die Ableitung der Funktion f_i nach der j-ten Variable
und f_i die i-te Komponentenfunktion von f bezeichnet
(d.h. $f(x) = (f_1(x),\ldots,f_n(x))^T \in \mathbb{R}^n$).

Als $\|f(x)\|$ könnte z.B. $\Sigma_{i=1,\ldots,n}|f_i(x)|$ genommen werden.

Im verallgemeinerten Newton-Verfahren ist also bei jedem Schleifendurchlauf ein lineares Gleichungssystem zu lösen. Dies bietet keine Schwierigkeiten, da numer-

ische Algorithmen (z.B. die GAUSS-Elimination) in den meisten Programmbibliotheken gefunden werden können. Es müssen jedoch zunächst die Koeffizienten des Gleichungssystems berechnet werden, d.h. es müssen die Funktion f und alle n^2 partiellen Ableitungen von f an der Stelle x ausgewertet werden.

Zur Auswertung von f(x) wird man ein FORTRAN-Unterprogramm schreiben. Zur Auswertung der partiellen Ableitungen kann man verschiedene Wege gehen:

- Man bildet die n^2 partiellen Ableitungen händisch und codiert diese Formeln in ein FORTRAN-Unterprogramm. Dies ist bei komplizierteren Funktionen recht aufwendig und muß, wenn sich an f etwas ändert, wiederholt werden.

- Man bildet die Ableitungen durch numerisches Differenzieren (und nimmt an dieser Stelle Ungenauigkeiten in Kauf).

- Man differenziert die gegebene Funktion unter Zuhilfenahme eines Computer-Algebra-Systems symbolisch und generiert dann automatisch mit dem Computer-Algebra-System ein FORTRAN-Unterprogramm zur Auswertung der Jacobimatrix.

Wie bereits weiter oben angekündigt, wird im folgenden eine Lösung für die dritte Alternative vorgestellt. Das folgende Programm wurde in REDUCE geschrieben und generiert zu einer gegebenen Funktion f ein FORTRAN-Unterprogramm, das zu einem gegebenen Vektor x sowohl den Wert der Funktion f(x) als auch den Wert der Jacobi-Matrix D(f,x) berechnet. In diesem Beispiel sollen Nullstellen der Funktion

$$f(x_1,x_2,x_3) = ((1-x_3).\sin(1+x_1)+e^{x_2} - e\ ,\ 2x_1^2.x_2 - x_2.x_3 - 1\ ,\ e^{x_1x_2}+x_1.x_2 - 2)^T$$

gesucht werden:

```
(RESTORE (QUOTE REDUCE))
(BEGIN)
ARRAY FUN(10),VAR(10),BOUT(10),AOUT(10,10);
MAXDIM := 10$
NDIM := 3$
VAR(1) := X1$
VAR(2) := X2$
VAR(3) := X3$
FUN(1) := (1-X3)*SIN(1 + X1) + E**X2-E$
FUN(2) := 2*X1**2*X2-X2*X3-1$
FUN(3) := E**(X1*X2) + X1*X2-2$
ON FORT;
OFF ECHO;
OUT DEMF2;
WRITE"    SUBROUTINE GENKOE(XIN,NDIM,BOUT,AOUT)";
WRITE"    DIMENSION XIN(NDIM),BOUT(NDIM),AOUT(NDIM,NDIM)";
WRITE"    E = 2.71828";
```



```
WRITE"    X1 = XIN(1)";
WRITE"    X2 = XIN(2)";
WRITE"    X3 = XIN(3)";
FOR I := 1 : NDIM DO
  WRITE BOUT(I) := FUN(I);
FOR I := 1 : NDIM DO
  FOR J := 1 : NDIM DO
    WRITE AOUT(I,J) := DF(FUN(I),VAR(J));
WRITE"    RETURN";
WRITE"    END";
END;
```

Sollen Nullstellen einer anderen Funktion berechnet werden, so sind nur die fettgedruckten Teile des Programms zu ändern. Dieses REDUCE Programm erstellt folgendes FORTRAN-Unterprogramm mit dem Namen GENKOE:

```
SUBROUTINE GENKOE(XIN,NDIM,BOUT,AOUT)
DIMENSION XIN(NDIM),BOUT(NDIM),AOUT(NDIM,NDIM)
E = 2.71828
X1 = XIN(1)
X2 = XIN(2)
X3 = XIN(3)
BOUT(1.) = -E + E**X2-SIN(X1 + 1.)*X3 + SIN(X1 + 1.)
BOUT(2.) = -X3*X2 + 2.*X2*X1**2-1.
BOUT(3.) = (E**X1 + E**X2*X2*X1-2.*E**X2)/E**X2
AOUT(1.,1.) = COS(X1 + 1.)*(-X3 + 1.)
AOUT(1.,2.) = E**X2
AOUT(1.,3.) = -SIN(X1 + 1.)
AOUT(2.,1.) = 4.*X2*X1
AOUT(2.,2.) = -X3 + 2.X1**2
AOUT(2.,3.) = -X2
AOUT(3.,1.) = (E**X1 + E**X2*X2)/E**X2
AOUT(3.,2.) = (-E**X1 + E**X2*X1)/E**X2
AOUT(3.,3.) = 0
RETURN
END
```

Dieses Unterprogramm kann dann an geeigneter Stelle des Hauptprogrammes für das Newton-Verfahren (das hier nicht wiedergegeben wird) aufgerufen werden. Insgesamt reduziert sich also (wenn man die Ableitungen exakt rechnen möchte)

die Aufgabe des Bestimmens der n^2 partiellen Ableitungen sowie des Codierens der Funktion f und der partiellen Ableitungen als FORTRAN-Unterprogramm auf

die Aufgabe des Codierens der Funktion f in das obige REDUCE-Programm.

3.3 Beispiel: Roboterkinematik

Ein Industrieroboter ist ein allgemeiner Manipulator, der aus einer Reihe von starren Armen besteht, die jeweils durch Dreh- oder prismatische Gelenke miteinander verbunden sind. Das eine Ende dieser Kette von Armen ist fest mit einer Basis verbunden. Am anderen (freien) Ende ist ein *Endeffektor* befestigt, der es gestattet, die gewünschten Arbeiten durchzuführen (z.B. eine "Zange" zum Greifen, eine Spritzpistole zum Lackieren oder ein Punktschweißer). Jede Bewegung in den Gelenken (z.B. Rotation um die Drehachse bei einem Drehgelenk) bewirkt eine Bewegung der Arme relativ zueinander und insgesamt eine (absolute) Positionierung und Orientierung des Endeffektors (und damit des Werkzeuges) innerhalb eines (fix mit der Basis des Roboters verbundenen) Basiskoordinatensystems.

In der Kinematik von Robotern unterscheidet man folgende zwei Fragestellungen:

> Das Problem der *direkten Kinematik*, d.h. der Bestimmung von Position und Orientierung des Endeffektors (innerhalb des Basiskoordinatensystems), wenn die relativen Bewegungen aller Gelenke (z.B. die Drehwinkel bei Drehgelenken) bekannt sind.
>
> Das Problem der *inversen Kinematik*, d.h. der Bestimmung der notwendigen relativen Bewegungen aller Gelenke, um eine vorgegebene Positionierung und Orientierung des Endeffektors zu erreichen.

Grundsätzlich geht man zur Lösung dieser kinematischen Probleme so vor: Man "befestigt" zunächst im Gelenk eines jeden Roboterarmes sowie im Endeffektor ein Koordinatensystem. Die relativen Bewegungen zweier benachbarter Koordinatensysteme sowie auch jene zwischen dem Basiskoordinatensystem und dem Koordinatensystem des ersten Gelenkes beschreibt man durch Transformationsmatrizen (${}_{i-1}T^{i}$ für die Transformation von Gelenk i-1 nach Gelenk i) in homogenen Koordinaten. In die Transformationsmatrizen gehen die geometrischen Parameter des Roboters, wie z.B. die Länge der Gelenke und die Lage der Drehachsen zueinander, sowie die Bewegungen der Gelenke, wie z.B. die Drehwinkel, ein. Die relativen Bewegungen zweier nicht benachbarter Arme und damit auch die Bewegung des Endeffektors relativ zum Basiskoordinatensystem lassen sich dann als Produkte dieser Transformationsmatrizen beschreiben. Aus dem Matrixprodukt für die Bewegung des Endeffektors innerhalb des Basiskoordinatensystems und der (vorgegebenen bzw. gesuchten) Matrix P für Position und Orientierung des Endeffektors ergeben sich die sogenannten *kinematischen Gleichungen*

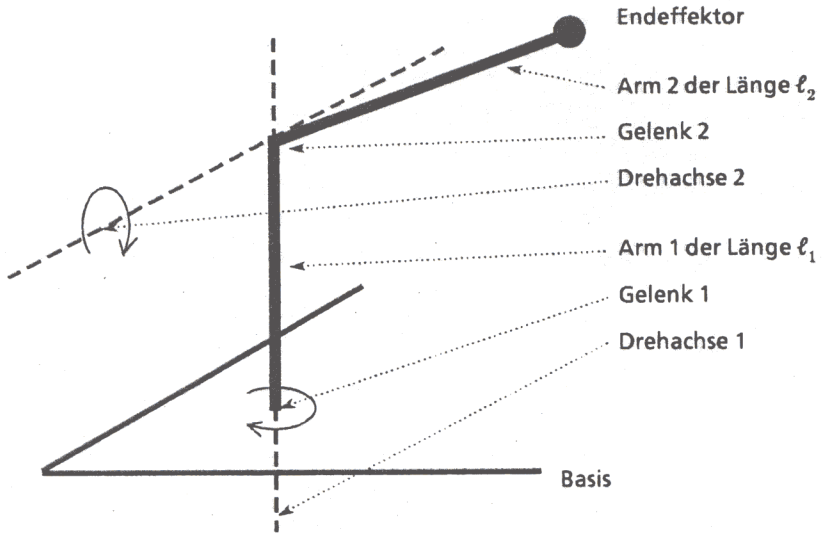
$${}_{\text{Basis}}T^{1} \cdot {}_{1}T^{2} \cdot \ldots \cdot {}_{n-1}T^{n} \cdot {}_{n}T^{\text{Endeffektor}} = P.$$

Das Problem der direkten Kinematik (d.h. unbekannter Matrix P und bekannten Transformationsmatrizen) läßt sich dann einfach durch Ausmultiplizieren der linken Seite dieser Gleichung lösen. Das Problem der inversen Kinematik (d.h. bekannter Matrix P und unbekannten Größen in den Transformationsmatrizen) führt auf das Problem des Lösens eines Systems von Gleichungen mit trigonometrischen Ausdrücken, welches sich aber nach Substitutionen der Art $u = \sin\delta$, $v = \cos\delta$ leicht als ein System von multivariaten Polynomgleichungen beschreiben läßt, für das im Abschnitt *Computer-Algebra-Algorithmen* ein Lösungsverfahren angegeben wird.

Wir wollen uns also zunächst mit der Aufstellung der kinematischen Gleichung beschäftigen, wofür noch eine Beschreibung der *Geometrie des Roboters* benötigt wird. Eine gebräuchliche Methode dafür stammt von /Denavit,Hartenberg 1955/. Diese Methode legt zunächst die genaue Lage der den einzelnen Roboterarmen/gelenken zugeordneten Koordinatensysteme fest und weist dann jedem Gelenk vier Werte δ, α, a, d zu, von denen drei durch die Konstruktion des Roboters bestimmt sind und einer die Bewegung des Gelenkes beschreibt, also je nach Art der Fragestellung ebenfalls bekannt oder unbekannt ist. (Die Bewegung von Drehgelenken wird durch den Drehwinkel δ, die Bewegung von prismatischen Gelenken wird durch die Verschiebung d beschrieben.) Aus dieser Denavit-Hartenberg-Darstellung eines Roboters läßt sich dann (durch Aufstellen und Ausmultiplizieren der Transformationsmatrizen) die linke Seiten der kinematischen Gleichungen gewinnen. Die notwendigen (Symbol-) manipulationen werden von dem folgenden Programm im Computer-Algebra-System SCRATCHPAD-II durchgeführt, das als Eingabe die Denavit-Hartenberg-Darstellung in Form einer Matrix DH erwartet (in der in der i-ten Zeile die Parameter des i-ten Gelenkes stehen) und als Ausgabe die linken Seiten der kinematischen Gleichungen liefert.

```
KINEMATICS(DH) = =
    RES : = 1
    FOR I IN 1..SIZE(DH) REPEAT RES : = RES*TRANS(DH(I-1))
    RETURN RES

TRANS(PL) = =
    DE : = PL(0)
    AL : = PL(1)
    A : = PL(2)
    D : = PL(3)
    [[COS(DE),-SIN(DE),0,0],[SIN(DE),COS(DE),0,0],[0,0,1,D],[0,0,0,1]]*
    [[1,0,0,A],[0,COS(AL),-SIN(AL),0],[0,SIN(AL),COS(AL),0],[0,0,0,1]]
```

(Das Hauptprogramm KINEMATICS führt die Multiplikation der vom Unterprogramm TRANS aus den Denavit-Hartenberg-Parametern erstellten Transformationsmatrizen benachbarter Gelenke durch.)



Der in obiger Skizze dargestellte zweiarmige Roboter werde durch folgende Denavit-Hartenberg Parameter beschrieben:

Gelenk 1:	δ_1	$\alpha_1 = \pi/2$	$a_1 = 0$	$d_1 = \ell_1$
Gelenk 2:	δ_2	$\alpha_2 = 0$	$a_2 = \ell_2$	$d_2 = 0$

(Da beide Gelenke Drehgelenke sind, beschreibt jeweils der Parameter δ die Bewegung und ist daher variabel. α, a und d beschreiben jeweils die konkrete Geometrie des Roboterarmes.)

Beachte: Das Program KINEMATICS multipliziert Matrizen in welchen Terme mit *Variablen* vorkommen!

```
KINEMATICS([[D1,PI()/2,0,L1],[D2,0,L2,0]]

| COS(D1) COS(D2)     - COS(D1) SIN(D2)     SIN(D1)      L2*COS(D1) COS(D2)     |
| COS(D2) SIN(D1)     - SIN(D1) SIN(D2)     - COS(D1)    L2*COS(D2) SIN(D1)     |
|      SIN(D2)              COS(D2)              0        L2*SIN(D2) + L1        |
|         0                    0                 0              1.0              |
```

Möchte man jetzt für konkrete Drehwinkel δ_1 und δ_2 die Position und die Orientierung des Endeffektors berechnen, so muß man nur noch für D1 und D2 diese Werte einsetzen und die einzelnen Terme auswerten. Die Position wird durch den äußerst rechten Spaltenvektor angegeben. Die Orientierung wird durch die linke obere 3x3 Matrix dargestellt, aus der man leicht z.B. die Euler-Winkel ϕ,θ,ψ (d. h. eine Beschreibung der Orientierung durch eine Rotation ϕ um die z-Achse, dann eine Rotation δ um die y-Achse und zuletzt eine Rotation ψ um die neue z-Achse) ausrechnen kann.

Weitaus interessanter ist natürlich das Problem der inversen Kinematik, das von zentraler Bedeutung für die Robotersteuerung ist. Man unterscheidet dabei die folgenden Fälle:

IK1) Die Position und die Orientierung des Endeffektors sind bekannt (d.h. liegen als Gleitkommazahlen vor). Gesucht sind Gleitkommazahlen für die die Bewegung beschreibenden Größen (z.B. Drehwinkel).

IK2) Die Position und die Orientierung des Endeffektors sind variabel (d.h. gehen als Parameter in die Gleichung ein). Gesucht sind Lösungen (d.h. Terme) für die die Bewegung beschreibenden Größen, in denen diese Parameter vorkommen.

IK3) Wie IK2), zusätzlich sind aber die Längen der Gelenke variabel.

IK1 führt (nach geeigneter Substitution für die Winkelfunktionen) auf ein multivariates polynomiales Gleichungssystem, dessen Lösungen z.B. mit den im Abschnitt *Computer-Algebra-Algorithmen: Gröbner-Basen* besprochenen Methoden oder mit numerischen Verfahren berechnet werden können. Allerdings muß dieses Problem für jede neue Position und Orientierung neu gelöst werden. Viel interessanter ist eine Lösung von IK2, die für den *konkreten* Roboter eine generelle Lösung des Inverse-Kinematik-Problems bedeutet. Sobald man eine generelle Lösung hat bekommt man die speziellen Lösungen für eine konkrete Position durch einfaches Einsetzen. Eine Lösung von IK3 ist eine generelle Lösung des Inverse-Kinematik-Problems für eine ganze Roboter*klasse*.

Im Folgenden wollen wir für den obigen zweiarmigen Roboter das Problem IK3 lösen. Dies ist ebenfalls mit der Methode der Gröbner-Basen möglich. Zunächst setzen wir die rechte Seite der kinematischen Gleichungen unter Verwendung der Euler-Winkel zur Beschreibung der Orientierung und der Koordinaten px, py, pz für die Position des Endeffektors an. Dies führt auf ein Gleichungssystem, das zunächst noch trigonometrische Funktionen enthält. Durch die Einführung neuer Variablen (z.B. c1 für cos(δ1), s1 für sin(δ1), cf für cos(ϕ), etc..) sowie die Hinzunahme der zwischen diesen Variablen geltenden Beziehungen (c1**2+s1**2 - 1=0, etc.) erhält man das folgende Gleichungssystem:

c1 c2 - cf ct cp + sf sp = 0,	$\ell 2$ c1 c2 - px = 0,
s1 c2 - sf ct cp - cf sp = 0,	$\ell 2$ s1 c2 - py = 0,

wobei u ein Eigenvektor zum Eigenwert λ ist. Dies führt also auf das Eigenwertproblem

$$A . u = \lambda . u,$$

das mit der in MACSYMA zur Verfügung stehenden Funktion EIGENVECTORS gelöst werden kann:

```
(C1) A: MATRIX([-1,1,0],[P,-2*P,P],[0,1,-1]);

                [ -1      1     0  ]
                [                  ]
(D1)            [ P     -2 P    P  ]
                [                  ]
                [ 0       1    -1  ]

(C2) EIGENVECTORS(A);

(D2)        [[[-2 P - 1, -1, 0], [1, 1, 1]], [1, -2 P, 1], [1, 0, -1], [1, 1, 1]]
```

Im ersten Vektor stehen die drei Eigenwerte -2p-1, -1 und 0 gefolgt von einem Vektor, der ihre Vielfachheiten (je 1) angibt. Dann folgen die jeweils dazugehörenden Eigenvektoren. Im folgenden Dialog werden die Eigenwerte im Vektor EW gespeichert, die Eigenvektoren in der Matrix EV. Dann wird der Lösungsvektor I aus EW und EV konstruiert. (K_1,K_2,K_3 bezeichnen Integrationskonstante.)

```
(C3) EW:PART(PART(%,1),1)$
(C3) EV: ADDCOL(TRANSPOSE(PART(%,2)),TRANSPOSE(PART(%,3),TRANSPOSE(PART(%,4)));

                [ 1      1       1  ]
                [                   ]
(D3)            [ -2 P   0       1  ]
                [                   ]
                [ 1      -1      1  ]

(C4) I: 0 $ FOR J: 1 THRU 3 DO I: I + K[J]*%E**(COL(EV,J)*T)*EW(J);
(D4)         DONE
(C5) I;
                [       (-2P-1) T        -T        ]
                [ K %E           + K %E   + K      ]
                [   1                 2         3  ]
                [                                  ]
                [                 (-2P-1) T        ]
(D5)            [   K  - 2 K P %E                  ]
                [    3      1                      ]
                [                                  ]
                [       (-2P-1) T        -T        ]
                [ K %E           - K %E   + K      ]
                [   1                 2         3  ]
```

Zu Demonstrationszwecken führen wir auch die Probe durch, die bei einem Computer-Algebra-System zu exakten Nullen führen muß:


```
(C6) EXPAND(DIFF(I,T) - A.I);

                [ 0 ]
                [   ]
(D6)            [ 0 ]
                [   ]
                [ 0 ]
```

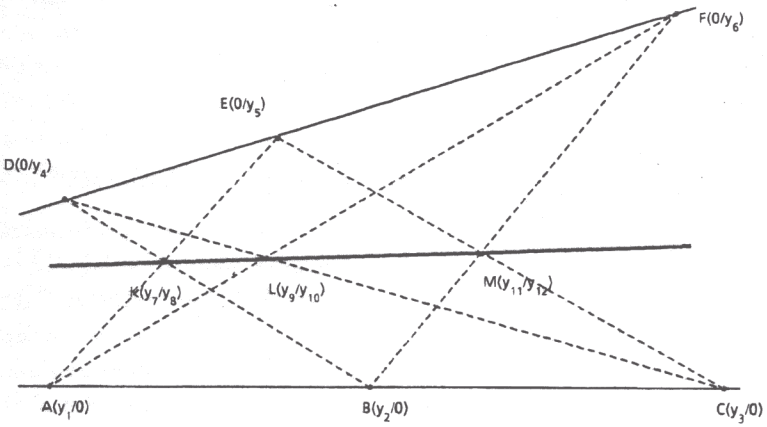
3.5 Beispiel: Automatisches Beweisen geometrischer Sätze

Graphische Techniken zur Computerunterstützung des Problemlösens, wie z.B. Computergraphik und CAD (Computer Aided Design), haben in letzter Zeit die Geometrie wieder in den Mittelpunkt des Interesses gerückt. Und auch in diesem Gebiet sind viele Probleme mit Methoden aus der Computer-Algebra lösbar. Als ein anspruchsvolles Beispiel präsentieren wir eine Computer-Algebra-Methode zum automatischen Beweisen geometrischer Sätze.

Bei diesem Problem geht es darum, die Gültigkeit einer Aussage über einen geometrischen Sachverhalt (automatisch) zu beweisen. Als Beispiel betrachten wir den *Satz von Pappus* aus der *projektiven Geometrie*:

> "Liegen die Ecken eines Sechseckes abwechselnd auf zwei Geraden, dann sind die drei Schnittpunkte der drei Gegenseitenpaare kollinear."

Zunächst wird das geometrische Objekt in ein Koordinatensystem gelegt (das in der projektiven Geometrie nicht rechtwinkelig sein muß), um dann (mit Hilfe der den Eckpunkten zugeordneten Koordinaten) den geometrischen Sachverhalt algebraisch (und zwar durch polynomiale Gleichungen) auszudrücken:



s2+st cp=0,
- c1 s2 - cf ct sp+sf cp=0,
- s1 s2+sf ct sp - cf cp=0,
c2 - st sp=0,
s1 - cf st=0,
- c1 - sf st=0,
ct=0.

ℓ2 s2+ℓ1 - pz=0,
c1**2+s1**2 - 1=0,
c2**2+s2**2 - 1=0,
cf**2+sf**2 - 1=0,
ct**2+st**2 - 1=0,
cp**2+sp**2 - 1=0,

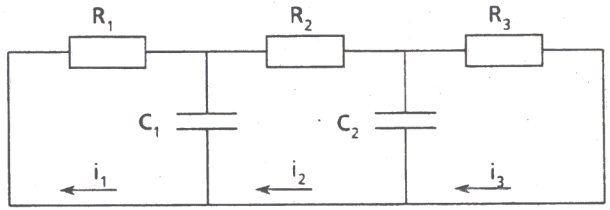
Von den sechs Bestimmungsstücken aus Position und Orientierung können für diesen Roboter (der ja nur zwei Freiheitsgrade besitzt) nur zwei beliebig vorgegeben werden, z.B. px und pz. Außerdem sind noch ℓ1 und ℓ2 frei wählbar. Wir können also die Polynome als Polynome in den Variablen c1,c2,s1,s2,py,cf,ct,cp,sf,st,sp mit Koeffizienten aus dem rationalen Funktionenkörper Q(ℓ1,ℓ2,px,pz) betrachten. Die Gröbner-Basis des obigen Systems über diesem Bereich, (die die gleiche Lösungsmenge wie das ursprüngliche System besitzt), berechnet mit einer Implementierung des Algorithmus von /Buchberger 1970/ durch /Gebauer,Kredel 1982/ im Computer-Algebra-System SAC-2, hat dann die folgende Gestalt:

c1**2+px**2 / (pz**2 - 2 ℓ1 pz - ℓ2**2+ℓ1**2)=0,
c2+((pz**2 - 2 ℓ1 pz - ℓ2**2+ℓ1**2) / ℓ2) px c1=0,
s1**2 - (pz**2 - 2 ℓ1 pz+px**2 - ℓ2**2+ℓ1**2) /
(pz**2 - 2 ℓ1 pz - ℓ2**2+ℓ1**2)=0,
s2 - (pz - ℓ1) / ℓ2=0,
py+((pz**2 - 2 ℓ1 pz - ℓ2**2+ℓ1**2) / px) c1 s1=0,
cf**2 - (pz**2 - 2 ℓ1 pz+px**2 - ℓ2**2+ℓ1**2) /
(pz**2 - 2 ℓ1 pz - ℓ2**2+ℓ1**2)=0,
ct=0,
cp+((pz**3 - 3 ℓ1 pz**2 - ℓ2**2 pz+3 ℓ1**2 pz+ℓ1 ℓ2**2 - ℓ1**3) /
(ℓ2 pz**2 - 2 ℓ1 ℓ2 pz+ℓ2 px**2 - ℓ2**3+ℓ1**2 ℓ2)) s1 cf=0,
sf+((pz**2 - 2 ℓ1 pz - ℓ2**2+ℓ1**2) /
(pz**2 - 2 ℓ1 pz+px**2 - ℓ2**2+ℓ1**2)) c1 s1 cf=0,
st+((pz**2 - 2 ℓ1 pz - ℓ2**2+ℓ1**2) /
(pz**2 - 2 ℓ1 pz+px**2 - ℓ2**2+ℓ1**2)) s1 cf=0,
sp+((pz**4 - 4 ℓ1 pz**3 - 2 ℓ2**2 pz**2+6ℓ1**2 pz**2+4 ℓ1 ℓ2**2 pz -
4 ℓ1**3 pz+ℓ2**4 - 2 ℓ1**2 ℓ2**2+ℓ1**4) / (ℓ2 px pz**2 - 2 ℓ1 ℓ2 px pz+
ℓ2 px**3 - ℓ2**3 px+ℓ1**2 ℓ2 px)) c1 s1 cf=0.

Wir sehen, das dieses System "trianguliert" ist (d.h. die erste Gleichung hängt nur von der einen Variablen c_1 ab, die zweite nur von c_1 und c_2, usw; es werden nie mehrere Variable gleichzeitig von einem Polynom eingeführt), es erlaubt also eine Berechnung aller Lösungen für konkrete Werte px,pz,ℓ1,ℓ2 durch sukzessives Lösen von univariaten polynomialen Gleichungen und Einsetzen der gefundenen Werte in die nachfolgenden Gleichungen. Setzt man nämlich konkrete Werte für px,pz,ℓ1,ℓ2 in die erste Gleichung ein, so erhält man durch Lösen der ersten Gleichung alle Lösungen für c1. Setzt man px,pz,ℓ1,ℓ2 und c1 in die zweite Gleichung ein, so kann man alle Lösungen für c2 berechnen, usw. Es bleibt also das Lösen von Gleichungen in einer Variablen übrig, für das, falls eine "analytische" Lösung (mit "Radikalen") nicht möglich ist, numerische Verfahren oder exakte Verfahren in algebraischen Erweiterungskörpern von Q angewendet werden können. (Im hier behandelten Beispiel ist sogar eine analytische Lösung möglich, weil die Variablen c1,c2,s1,s2,py,cf, ct,cp,sf,st,sp höchstens quadratisch vorkommen.)

3.4 Beispiel: Verhalten einer elektrischen Schaltung

Anhand der Berechnung des Verhaltens des folgenden Schaltkreises demonstrieren wir die Fähigkeiten von Computer-Algebra-Systemen bei der Manipulation von Matrizen (Berechnung der Eigenwerte, Eigenvektoren, etc.). Das Beispiel wurde mit MACSYMA gerechnet und ist /Rand 1984/ entnommen.



Gesucht sind die Lösungen für den Strom i (d.h. i_1,i_2,i_3) für die folgende Wahl der Parameter: $R_1=R_3=C_1=C_2=1$, $R_2=1/p$. Dabei ergibt sich die Differentialgleichung

$$i' = A.i$$

wobei

$$A = \begin{pmatrix} -1 & 1 & 0 \\ p & -2p & p \\ 0 & 1 & -1 \end{pmatrix}, \quad q = \begin{pmatrix} q_1 \\ q_2 \\ q_3 \end{pmatrix}.$$

Die Lösung hat dann die Gestalt

$$i(t) = u \, . \, e^{\lambda t},$$

wobei u ein Eigenvektor zum Eigenwert λ ist. Dies führt also auf das Eigenwertproblem

$$A \cdot u = \lambda \cdot u,$$

das mit der in MACSYMA zur Verfügung stehenden Funktion EIGENVECTORS gelöst werden kann:

```
(C1) A: MATRIX([-1,1,0],[P,-2*P,P],[0,1,-1]);

                [ -1      1     0  ]
                [                  ]
(D1)            [ P     -2 P    P  ]
                [                  ]
                [ 0       1    -1  ]

(C2) EIGENVECTORS(A);

(D2)        [[[-2 P - 1, -1, 0], [1, 1, 1]], [1, -2 P, 1], [1, 0, -1], [1, 1, 1]]
```

Im ersten Vektor stehen die drei Eigenwerte -2p-1, -1 und 0 gefolgt von einem Vektor, der ihre Vielfachheiten (je 1) angibt. Dann folgen die jeweils dazugehörenden Eigenvektoren. Im folgenden Dialog werden die Eigenwerte im Vektor EW gespeichert, die Eigenvektoren in der Matrix EV. Dann wird der Lösungsvektor I aus EW und EV konstruiert. (K_1,K_2,K_3 bezeichnen Integrationskonstante.)

```
(C3) EW:PART(PART(%,1),1)$
(C3) EV: ADDCOL(TRANSPOSE(PART(%,2)),TRANSPOSE(PART(%,3),TRANSPOSE(PART(%,4)));

                [ 1      1       1  ]
                [                   ]
(D3)            [ -2 P   0       1  ]
                [                   ]
                [ 1      -1      1  ]

(C4) I: 0 $ FOR J: 1 THRU 3 DO I: I+K[J]*%E**(COL(EV,J)*T)*EW(J);
(D4)         DONE
(C5) I;
                [          (-2P-1) T          -T          ]
                [ K %E             + K %E   + K           ]
                [  1                 2          3         ]
                [                                         ]
                [                    (-2P-1) T            ]
(D5)            [   K  - 2 K  P %E                        ]
                [    3       1                            ]
                [                                         ]
                [          (-2P-1) T          -T          ]
                [ K %E             - K %E   + K           ]
                [  1                 2          3         ]
```

Zu Demonstrationszwecken führen wir auch die Probe durch, die bei einem Computer-Algebra-System zu exakten Nullen führen muß:


```
(C6) EXPAND(DIFF(I,T) - A.I);

                [ 0 ]
                [   ]
(D6)            [ 0 ]
                [   ]
                [ 0 ]
```

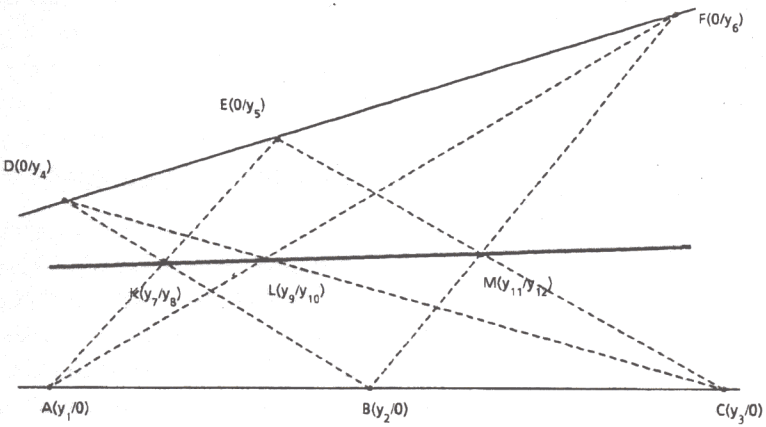
3.5 Beispiel: Automatisches Beweisen geometrischer Sätze

Graphische Techniken zur Computerunterstützung des Problemlösens, wie z.B. Computergraphik und CAD (Computer Aided Design), haben in letzter Zeit die Geometrie wieder in den Mittelpunkt des Interesses gerückt. Und auch in diesem Gebiet sind viele Probleme mit Methoden aus der Computer-Algebra lösbar. Als ein anspruchsvolles Beispiel präsentieren wir eine Computer-Algebra-Methode zum automatischen Beweisen geometrischer Sätze.

Bei diesem Problem geht es darum, die Gültigkeit einer Aussage über einen geometrischen Sachverhalt (automatisch) zu beweisen. Als Beispiel betrachten wir den *Satz von Pappus* aus der *projektiven Geometrie*:

> "Liegen die Ecken eines Sechseckes abwechselnd auf zwei Geraden, dann sind die drei Schnittpunkte der drei Gegenseitenpaare kollinear."

Zunächst wird das geometrische Objekt in ein Koordinatensystem gelegt (das in der projektiven Geometrie nicht rechtwinkelig sein muß), um dann (mit Hilfe der den Eckpunkten zugeordneten Koordinaten) den geometrischen Sachverhalt algebraisch (und zwar durch polynomiale Gleichungen) auszudrücken:



Eine algebraische Formulierung des Satzes von Pappus hätte dann die Gestalt:

$$(\forall y_1,\ldots,y_{12})(\,(y_7-y_1)y_5+y_8y_1=0 \wedge (y_7-y_2)y_4+y_8y_2=0 \wedge (y_9-y_1)y_6+y_{10}y_1=0 \wedge$$
$$(y_9-y_3)y_4+y_{10}y_3=0 \wedge (y_{11}-y_2)y_6+y_{12}y_2=0 \wedge (y_{11}-y_3)y_5+y_{12}y_3=0$$
$$\Rightarrow\ (y_9-y_7)(y_{12}-y_8)-(y_{10}-y_8)(y_{11}-y_7)=0\,)$$

Dabei entsprechen die sechs polynomialen Gleichungen vor dem Implikationspfeil den Hypothesen des Satzes, nämlich daß K der Schnittpunkt der Geraden AE und BD sei, etc. Die Gleichung nach dem Implikationspfeil entspricht der Behauptung des Satzes, nämlich daß K,L und M dann kollinear sind.

Formeln dieser Gestalt lassen sich sowohl mit der im nächsten Abschnitt besprochenen Methode der Gröbner-Basen als auch mit der dort ebenfalls besprochenen Methode von Collins zur zylindrisch-algebraischen Dekomposition entscheiden. Mit Gröbner-Basen gelingt die Entscheidung im wesentlichen auf folgende Art: Für die multivariaten Polynome vor dem Implikationspfeil wird die zugehörige Gröbner-Basis berechnet. Dann wird geprüft, ob das Polynom nach dem Implikationspfeil in Bezug auf die Gröbner-Basis auf 0 reduziert. Einzelheiten dazu (insbesondere den Beweis der Korrektheit) findet man in /Kutzler,Stifter 1986/. Im gegenständlichen Beispiel dauert die gesamte Rechenzeit 11 Sekunden auf einer IBM 4341 mit einer Implementierung des Algorithmus in SAC-2. Mit dieser Methode können heute bereits sämtliche in Standardgymnasiallehrbüchern behandelten geometrischen Sätze in wenigen Sekunden bewiesen werden. Darüberhinaus auch eine Reihe (bisher ca. 50) von sehr viel schwierigeren Sätzen aus verschiedenen Geometrien (u.a. der Satz von Pascal), sowie Beispiele aus den Bewerben der Internationalen Mathematikolympiaden mit Rechenzeiten von wenigen Sekunden bis zu einigen Stunden.

4 Übersicht über wichtige Computer-Algebra-Systeme

In diesem Abschnitt wollen wir kurz einen Überblick über wichtige Computer-Algebra-Systeme geben. Für jedes System geben wir die Hardware, auf der das System läuft, die Bezugsquelle(n) sowie Bemerkungen zur Verfügbarkeit und Literaturhinweise für weitere Informationen an. Außerdem geben wir eine Einteilung der

Systeme in wichtige Klassen, nämlich Interaktiv / Batch (siehe auch Abschnitt *Problemlösepotenz von Computer-Algebra-Systemen: Einige Beispiele*), Universell (d.h. mit Grundalgorithmen für Anwendungen in großen Bereichen) / Speziell (d.h. mit Algorithmen für Anwendungen in einem Spezialbereich), Großrechner / Microcomputer an. Eine ausführlichere Übersicht ist /VanHulzen, Calmet 1982/. Außerdem findet man laufend Beschreibungen von Software-Systemen in der Sektion *"Systems Descriptions"* des Journal of Symbolic Computation. Alle Systeme verfügen auch über eine Programmiersprache, mit der sowohl Erweiterungen des Systems als auch eigene Anwendungen realisiert werden können.

4.1 MACSYMA

Hardware: Symbolics 3600 LISP-Maschine; DEC VAX 11.

Bezugsquellen/Verfügbarkeit: Das wesentliche Grundgerüst des Systems wurde unter der Leitung von Prof. J. Moses (MIT) entwickelt. Für beide Maschinentypen ist MACSYMA von der Firma Symbolics GmbH, Frankfurter Str. 63-69, D-6236 Eschborn/Ts., BRD erhältlich. Eine etwas andere, nur für DEC VAX 11 erhältliche Version (DOE-MACSYMA) kann vom National Energy Software Center, Argonne Nat. Laboratory, 9700 South Cass Avenue, Argonne, Illinois 60439, USA bezogen werden.

Literatur: Eine Einführung in die Fähigkeiten des Systems gibt /Pavelle,Wang 1985/. Eine Sammlung zahlreicher Anwendungen dieses Systems auf Probleme aus den verschiedensten Sachgebieten bietet /Pavelle 85/, einer Zusammenfassung der interessantesten Arbeiten der Proceedings der drei bisherigen MACSYMA-Benutzerkonferenzen (1977, 1979 und 1984). Zahlreiche Informationen findet man auch im "MACSYMA Newsletter" der Firma Symbolics sowie im "MACSYMA Applications Newsletter", der monatlich von der Firma Paradigm Associates, Inc., 29 Putnam Avenue, Suite 6, Cambridge, MA 02139, USA herausgegeben wird.

Charakterisierung: MACSYMA ist derzeit eines der größten und leistungsfähigsten Computer-Algebra-Systeme. Es gehört zu den universellen Systemen und arbeitet interaktiv.

4.2 MAPLE

Hardware: DEC VAX bzw. MicroVAX mit VMS oder Berkeley Unix; DEC 20; IBM mit VM/CMS; TOPS-20; sowie auf verschiedenen Mini-Computern mit dem Betriebssystem Unix, wie z.B. MASSCOMP, Cadmus.

Bezugsquellen/Verfügbarkeit: MAPLE wurde unter der Leitung von Prof. K.O. Geddes in der Symbolic Computation Group, Department of Computer Science, University of Waterloo, Waterloo, Ontario, Kanada N2L 3G1, entwickelt.

Literatur: Eine Einführung in die Fähigkeiten des Systems gibt /Char et al. 1986/.

Charakterisierung: Maple gehört zu den wenigen auf Kleinrechnern verfügbaren Systemen. Der Speicherplatzbedarf ist mit wenigen hundert KByte deutlich unter dem der großen Systeme. Ein Hauptziel bei der Entwicklung war die durch die Wahl des Betriebssystems erreichte Portabilität und die Benutzung im Studienbetrieb großer Universitäten.

4.3 muMATH-83

Hardware: IBM PC / MS-DOS (und kompatible); Sony und Hewlett-Packard / MS-DOS (mit 3½ Zoll Disketten); fast alle Microcomputer mit CP/M-80 einschließlich jenen mit 8 Zoll Disketten sowie Apple II und IIe mit SoftCard.

Bezugsquellen/Verfügbarkeit: muMATH-83 wurde von Prof. D. Stoutemyer, Soft Warehouse Inc., P.O.Box 11174, Honolulu, Hawaii 96828, USA, entwickelt.

Literatur: Eine Einführung in das System gibt /Stoutemyer 1985/. Laufende Neuigkeiten bietet der von Soft Warehouse Inc. herausgegebene "muMATH Newsletter".

Charakterisierung: muMATH ist das einzige auf Microcomputern verfügbare Computer-Algebra-System. Es zeichnet sich wegen seiner modularen Bauweise durch wenig Speicherplatzbedarf (mindestens 128 KByte unter MS-DOS, mindestens 56 KByte unter CP/M-80) bei hoher Leistungsfähigkeit aus. (Nur die für die konkrete Anwendung notwendigen Module werden in den Hauptspeicher geladen). muMATH ist ein universelles, interaktiv arbeitendes System.

4.4 REDUCE 3.2

Hardware: IBM Serie 360 und ähnliche; DEC 10 und 20; DEC VAX; SUN; HP9000 Serie 200; HP Integral; Sage; Pinnacle; HLH Orion; Acorn 32016; GEC System 63; Apollo Domain; CRAY-1; Symbolics 3600 LISP-Maschine; Xerox Dolphin und Dandelion; DG Exlipse MV; Honeywell 68/DPS; CDC Cyber Serie; UNIVAC 1100; Burroughs B6000 und B7000; Tektronix Work Station.

Bezugsquellen/Verfügbarkeit: REDUCE wurde von Dr. A.C. Hearn und seiner Gruppe entwickelt und wird jetzt von The Rand Corporation, 1700 Main Street, P.O.Box 2138, Santa Monica, California 90406, USA, vertrieben.

Literatur: Eine Einführung in die Fähigkeiten des Systems gibt /Fitch 1985/. Laufend neue Informationen über das System findet man im "Reduce Newsletter", der von der Rand Corporation herausgegeben wird.

Charakterisierung: REDUCE gehört wegen seiner allgemeinen Verfügbarkeit zu den am weitesten verbreiteten Computer-Algebra-Systemen. REDUCE ist ein universelles, interaktiv arbeitendes System.

4.5 SAC-2

Hardware: prinzipiell alle Rechner, auf denen FORTRAN verfügbar ist.

Bezugsquellen/Verfügbarkeit: SAC-2 wurde von Prof. G.E. Collins, University of Wisconsin-Madison, Computer Science Department, 1210 West Dayton Street, Madison, Wisconsin 53706, USA und Prof. R. Loos, Universität Karlsruhe, Institut für Informatik I, Zirkel 2, D-7500 Karlsruhe, BRD entwickelt. Das System wird nicht kommerziell vertrieben, ist jedoch für wissenschaftliche Zwecke auf Anfrage bei den obigen Adressen erhältlich.

Literatur: Eine kurze Beschreibung des Systems gibt /Collins 1985/.

Charakterisierung: SAC-2 ist ein für das Arbeiten mit Polynomen entwickeltes System. Es sind die meisten der wichtigen Algorithmen für Polynome, einschließlich des Algorithmus von Collins zur zylindrisch-algebraischen Dekomposition implementiert. SAC-2 ist ein spezielles, im Batch-Betrieb arbeitendes System und war eines der ersten Computer-Algebra-Systeme. Seine Algorithmen sind die Grundlage vieler