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. Illig, Göppingen
Umschlaggestaltung: M. Koch, Reutlingen

Parallele Numerik (M. Feilmeier)

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 - hinzufugt und die Grundoperationen entsprechend erweitert, z.B. $\sqrt{2}.\sqrt{3} = \sqrt{6}$.)

Endliche Korper (z.B. der endliche Körper "Z modulo 5", der aus den Zahlen 0,1,2, 3,4 besteht mit den Operationen z.B. $2 \oplus 4 = 1$, $3 \oplus 2 = 0$ 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 $\frac{2}{3}x^3yz^2 - \frac{1}{2}xy^2z + 13xz$ 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" Ausdrucke 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 konnen.

Gruppen (die z.B. durch eine endliche Zahl von erzeugenden Elementen und "definierenden Relationen" gegeben sind) mit der Gruppenverknupfung 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 ausgefuhrt 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 wiedergegeben, 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.

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

@: 379/945
```

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!;

@:
7886578673647905035523632139321850622951359776871732632947425332443594499634033429203042840119346239041772121389196388302576427902426371050619266249523299311346285727076331723739698894392244562145166424002540329186413122742828485327752422075739324032125740657966866022603190417032406235170085879617892222278962370389737472000000000300000000000000000000000000000000000000000000000000
```

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 *Naherungen 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  (3 Z  + 2 W Z - 8 X Y  + 14 W  Y  - Y  + 18 X  Y)
                                 2    3    2  3       2          2
                (12 W  X Y Z  - W  Z  - 3 X Y  - 29 X - 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 konnen nicht nur unbestimmte, sondern auch manche bestimmte Integrale exakt berechnet werden. Das folgende uneigentliche Integral wurde in etwa 138 Sekunden gefunden (%E steht fur 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 gelost 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):

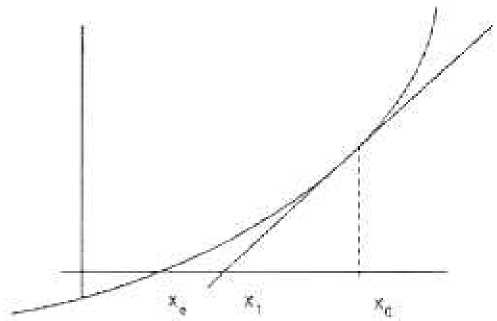
```
(C16) DEPENDS(Y,X)$

(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
                                               2 X  + 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 := x0
while |f(x)| > ε 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 := x0
while ||f(x)|| > ε 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:

$D(f,x)_{ij} = (\partial/\partial x_j) f_i(x)$,
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 Jacobi-matrix.

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 IQUOTE 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*

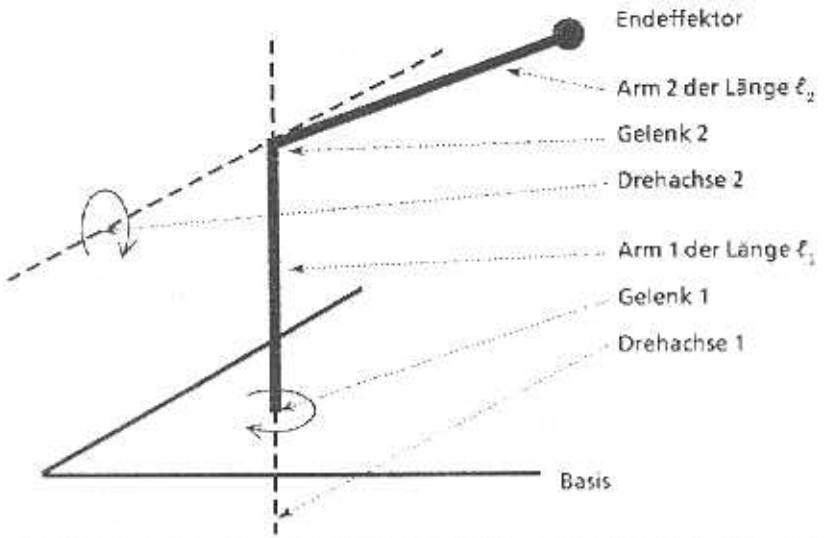
$$_{Basis}T^1 \cdot {}_1T^2 \cdot \ldots \cdot {}_{n-1}T^n \cdot {}_nT^{Endeffektor} = P.$$

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 Großen in den Transformationsmatrizen) führt auf das Problem des Lösens eines Systems von Gleichungen mit trigonometrischen Ausdrücken, welches sich aber nach Substitutionen der Art u = sinδ, v = cosδ leicht als ein System von multivariaten Polynomgleichungen beschreiben laßt, 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, a, 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 θ 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:

$$c1\,c2 - cf\,ct\,cp + sf\,sp = 0, \qquad \ell_2\,c1\,c2 - px = 0,$$
$$s1\,c2 - sf\,ct\,cp - cf\,sp = 0, \qquad \ell_2\,s1\,c2 - py = 0,$$

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

$\ell 2\,s2 + \ell 1 - pz = 0,$
$c1^{**}2 + s1^{**}2 - 1 = 0,$
$c2^{**}2 + s2^{**}2 - 1 = 0,$
$cf^{**}2 + sf^{**}2 - 1 = 0,$
$ct^{**}2 + st^{**}2 - 1 = 0,$
$cp^{**}2 + sp^{**}2 - 1 = 0.$

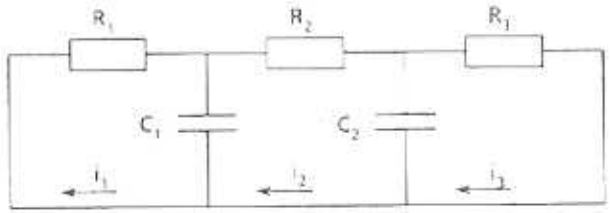
Von den sechs Bestimmungsstücken aus Position und Orientierung können für diesen Roboter (der ja nur zwei Freiheitsgrade besitzt) nur zwei beliebig vorgegeben werden, z.B. px und pz. Außerdem sind noch $\ell 1$ und $\ell 2$ frei wählbar. Wir können also die Polynome als Polynome in den Variablen c1,c2,s1,s2,py,cf,ct,cp,sf,st,sp mit Koeffizienten aus dem rationalen Funktionenkörper $Q(\ell 1,\ell 2,px,pz)$ 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:

$c1^{**}2 + px^{**}2 / (pz^{**}2 - 2\,\ell 1\,pz - \ell 2^{**}2 + \ell 1^{**}2) = 0,$
$c2 + ((pz^{**}2 - 2\,\ell 1\,pz - \ell 2^{**}2 + \ell 1^{**}2) / \ell 2)\ px\,c1 = 0,$
$s1^{**}2 \cdot (pz^{**}2 - 2\,\ell 1\,pz + px^{**}2 - \ell 2^{**}2 + \ell 1^{**}2) /$
$(pz^{**}2 - 2\,\ell 1\,pz - \ell 2^{**}2 + \ell 1^{**}2) = 0,$
$s2 \cdot (pz - \ell 1) / \ell 2 = 0,$
$py + ((pz^{**}2 - 2\,\ell 1\,pz - \ell 2^{**}2 + \ell 1^{**}2) / px)\ c1\,s1 = 0,$
$cf^{**}2 \cdot (pz^{**}2 - 2\,\ell 1\,pz + px^{**}2 - \ell 2^{**}2 + \ell 1^{**}2) /$
$(pz^{**}2 - 2\,\ell 1\,pz - \ell 2^{**}2 + \ell 1^{**}2) = 0,$
$ct = 0,$
$cp + ((pz^{**}3 - 3\,\ell 1\,pz^{**}2 - \ell 2^{**}2\,pz + 3\,\ell 1^{**}2\,pz + \ell 1\,\ell 2^{**}2 - \ell 1^{**}3) /$
$(\ell 2\,pz^{**}2 - 2\,\ell 1\,\ell 2\,pz + \ell 2\,px^{**}2 - \ell 2^{**}3 + \ell 1^{**}2\,\ell 2))\ s1\,cf = 0,$
$sf + ((pz^{**}2 - 2\,\ell 1\,pz - \ell 2^{**}2 + \ell 1^{**}2) /$
$(pz^{**}2 - 2\,\ell 1\,pz + px^{**}2 - \ell 2^{**}2 + \ell 1^{**}2))\ c1\,s1\,cf = 0,$
$st + ((pz^{**}2 - 2\,\ell 1\,pz - \ell 2^{**}2 + \ell 1^{**}2) /$
$(pz^{**}2 - 2\,\ell 1\,pz + px^{**}2 - \ell 2^{**}2 + \ell 1^{**}2))\ s1\,cf = 0,$
$sp + ((pz^{**}4 - 4\,\ell 1\,pz^{**}3 - 2\,\ell 2^{**}2\,pz^{**}2 + 6\,\ell 1^{**}2\,pz^{**}2 + 4\,\ell 1\,\ell 2^{**}2\,pz -$
$4\,\ell 1^{**}3\,pz + \ell 2^{**}4 - 2\,\ell 1^{**}2\,\ell 2^{**}2 + \ell 1^{**}4) / (\ell 2\,px\,pz^{**}2 - 2\,\ell 1\,\ell 2\,px\,pz +$
$\ell 2\,px^{**}3 - \ell 2^{**}3\,px + \ell 1^{**}2\,\ell 2\,px))\ c1\,s1\,cf = 0.$

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,$\ell 1$,$\ell 2$ durch sukzessives Lösen von univariaten polynomialen Gleichungen und Einsetzen der gefundenen Werte in die nachfolgenden Gleichungen. Setzt man nämlich konkrete Werte für px,pz,$\ell 1$,$\ell 2$ in die erste Gleichung ein, so erhält man durch Lösen der ersten Gleichung alle Lösungen für c1. Setzt man px,pz,$\ell 1$,$\ell 2$ 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          ]
(D6)            [    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 ]
```

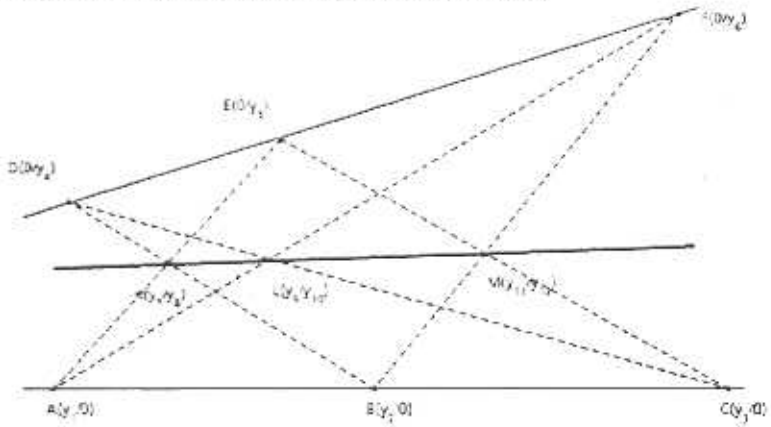
3.5 Beispiel: Automatisches Beweisen geometrischer Satze

Graphische Techniken zur Computerunterstutzung 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 losbar. 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 uber 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:

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

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 Standardgymnasiallehrbuchern 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 erhaltlich.

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

Algorithmen in anderen Systemen. Das System ist eines der wenigen, bei denen man auch die zugrundeliegende mathematischen Ideen studieren kann.

4.6 SCRATCHPAD II

Hardware: IBM Rechner mit VM Betriebssystem.

Bezugsquellen/Verfügbarkeit: Scratchpad wird von der Computer Algebra Group, Knowledge Systems, Computing Technology Department, IBM Thomas J. Watson Research Center, Box 218, Yorktown Heights, New York 10598, USA unter der Leitung von Dr. R.D. Jenks entwickelt. Das System ist derzeit noch nicht allgemein verfügbar.

Literatur: Eine Einführung in die Fähigkeiten des Systems gibt /Jenks 1984/. Eine kurze Beschreibung gibt /Sutor 1985/. Neue Informationen über das System sowie Berichte über Anwendungen findet man im "Scratchpad II Newsletter", der von IBM herausgegeben wird.

Charakterisierung: Scratchpad II zeichnet sich durch eine besondere Philosophie für die vom Benutzer verwendeten "Datentypen" aus, die es ermöglichen, Programme für sehr allgemeine Bereiche zu schreiben, um sie dann für viele konkrete Bereiche zu verwenden. (Z.B. kann ein Algorithmus zur Berechnung des größten gemeinsamen Teilers, der für den Bereich "Euklidischer Ring" geschrieben wurde, sowohl auf ganze Zahlen als auch auf Polynome mit rationalen Koeffizienten angewandt werden.) Scratchpad II ist ein universelles, interaktiv arbeitendes System.

4.7 SMP

Hardware: DEC VAX 11 Serie mit VMS

Bezugsquellen/Verfügbarkeit: SMP wurde im wesentlichen von S. Wolfram entwickelt und wird jetzt von der Inference Corporation, Suite 501, 5300 W. Century Blvd., Los Angeles, California 90045, USA, vertrieben.

Literatur: Eine Einführung in die Fähigkeiten des Systems gibt der von Inference Corp. erhältliche "SMP Primer".

Charakterisierung: SMP ist ein universelles, interaktiv arbeitendes System.

5 Computer-Algebra-Algorithmen

5.1 Vorbemerkung: Computer-Algebra versus reine Mathematik

"Reine" Mathematiker sind manchmal der Meinung, daß die Lösung von mathematischen, insbesondere algebraischen Problemen mit dem Computer darin besteht, "unintelligente" Einzelschritte sehr oft und sehr schnell hintereinander auszuführen, wogegen der Mathematiker bei der "händischen" Lösung von mathematischen Problemen bei jedem Schritt seine mathematischen Einsichten einsetzt (und damit oft durch wenige, langsame, aber "intelligente" Schritte im gesamten oft schneller zum Ziel kommt). Das heißt also: es wird manchmal gleichgesetzt

> Computer-Mathematik = Anwenden *unintelligenter Verfahren* (z.B. "Probieren aller endlich vielen Möglichkeiten") auf *unintelligenter*, aber schneller *Hardware* (dem Computer) (und beobachten, was für Resultate entstehen; "Experimentalmathematik").
>
> Reine Mathematik = Anwenden *intelligenter Verfahren* auf *intelligenter*, aber langsamer Hardware (dem Mathematiker) (und a priori beweisen, welche Eigenschaften die Resultate haben müssen; "Beweisende Mathematik").

Dies ist ein grobes Mißverständnis. Genau das Gegenteil ist der Fall.

> Um mit *unintelligenter* Hardware mathematische Probleme lösen zu können, muß man mit *intelligenteren* Verfahren arbeiten (PRINZIP *der Erhaltung der Summe von Verfahrensintelligenz und Hardwareintelligenz*).

Woher kommen intelligentere Verfahren? Intelligentere Verfahren inkorporieren mehr mathematisches Wissen, mehr mathematische Einsicht. Allgemein bewiesene mathematische Aussagen bilden die Grundlage für *effektive* (d.h. auf völlig unintelligenter Hardware überhaupt ausführbare) und *effiziente* (d.h. mit wenig Rechenzeit und Speicher ausführbare) Verfahren (Algorithmen). Es gilt das PRINZIP:

Mehr mathematisches Wissen ⇒ Effizientere Algorithmen.

Um zu effizienten (algebraischen) Algorithmen zu gelangen, muß man also noch tiefer in der Mathematik einsteigen, als wenn man nur an "prinzipiellen", im allgemeinen nicht algorithmisch ausführbaren Lösungen mathematischer Probleme interessiert ist, d.h. man muß in dem betrachteten Problembereich versuchen,

neues algorithmisch brauchbares Wissen zu beweisen.

Die ganze algorithmische Kraft steckt dabei in den Beweisen. Die Kraft der Beweise beruht auf dem PRINZIP:

> Ein einmal *(in endlich vielen Schritten) geführter Beweis* für einen nicht-trivialen mathematischen Satz führt bei den *unendlich vielen Angaben* für das entsprechende Problem zur *schnelleren* algorithmischen Bestimmung der Lösung.

(Die Beschleunigung, die durch einen mathematischen Satz für die algorithmische Lösbarkeit eines mathematischen Problems erzielt werden kann, könnte geradezu als ein Maß für die Brauchbarkeit, die "Interessantheit" des betreffenden Satzes betrachtet werden.) Demgegenüber ist es oft beim *"händischen" Bearbeiten von Problemen* nicht notwendig, sehr tiefe allgemeine mathematische Gesetzmäßigkeiten anzuwenden. Beim händischen Bearbeiten kann man oft sowieso nicht sehr große Beispiele betrachten und bei den einfachen Angaben sieht man oft Vereinfachungen, die allgemein gar nicht gelten müssen. Oder man kommt tatsächlich mit dem Durchprobieren aller Möglichkeiten aus. (Oft betrachtet die reine Mathematik ein Problem als "gelöst", wenn man gezeigt hat, daß es "nur" endlich viele Möglichkeiten gibt.)

5.2 Ein einfaches Beispiel für obige Prinzipien:

Wir betrachten das

Problem:

Gegeben: Zwei natürliche Zahlen x,y.
Gesucht: Der größte gemeinsame Teiler z von x und y. ▮

Die Definition des Begriffs "größter gemeinsamer Teiler" legt folgenden, *einfachen* ("unintelligenten") *Algorithmus* zur Bestimmung von z nahe:

```
for u := 1 to Minimum von x und y do
  if (u teilt x) und (u teilt y)
  then z := u ▮
```

Zusätzliches mathematisches Wissen (W):

ggT(x,0) = x
ggT(x,y) = ggT(y,Rest(x,y)) (falls y≠0)

(Hier steht "ggT(x,y)" für "größter gemeinsamer Teiler von x und y" und "Rest(x,y)" für "Rest bei der (ganzzahligen) Division von x durch y".) ▮

Beweis dieses Wissens:
Man zeigt in ein paar Zeilen (endlich vielen Zeilen!), daß (im Fall y≠0):
für alle z: z teilt x und y genau dann wenn z teilt y und Rest(x,y)

Das zusätzliche Wissen (W) über den Begriff des größten gemeinsamen Teilers legt folgenden *besseren* ("intelligenteren") *Algorithmus* nahe (Algorithmus von Euklid, ca. 300 v. Chr.):

```
while y≠0 do
  (x,y) := (y,Rest(x,y))
z := x ▮
```

Während das erste einfache Verfahren, sehr grob betrachtet, maximal $c_1.x.L(y)^2$ viele Schritte braucht, braucht das zweite Verfahren maximal $c_2.L(x).L(y)$ viele Schritte. (Hier sei ein "Schritt" eine Operation auf einer Ziffer; L(x) ist die "Länge" der Zahl x, d.h. die Anzahl der Ziffern von x; c_1 und c_2 sind Konstante, die durch die verwendete Maschine und die Implementierung der Algorithmen bestimmt sind; wir haben hier vorausgesetzt, daß L(x) kleiner oder gleich L(y) ist).

Die Abschätzungen über die Rechenzeiten der beiden Algorithmen zeigen, daß das zweite Verfahren "grundsätzlich", d.h. unabhängig von den Konstanten c_1 und c_2, d.h. unabhängig von den verwendeten Maschinen, besser ist. Das heißt genauer: Wie klein auch c_1 ist und wie groß auch c_2, d.h. wie gut auch die Maschine ist, auf der der erste Algorithmus implementiert wird und wie schlecht auch die Maschine ist, auf der der zweite Algorithmus implementiert wird, so gibt es immer eine Länge, ab welcher der zweite Algorithmus weniger Rechenzeit braucht als der erste. Man sagt kurz auch: Die *(Komplexitäts-) Ordnung* des ersten Algorithmus ist größer als die Ordnung des zweiten Algorithmus oder

$$O(x.L(y)^2) > O(L(x).L(y)).$$

Für das handische Rechnen bei kleinen Eingaben ist das starke mathematische Wissen (W) ein "overkill". Wenn man z.B. den größten gemeinsamen Teiler von 28 und 36 rechnet, "sieht man gleich", daß beide Zahlen 2 als Teiler enthalten, sogar zweimal und daß die verbleibenden Teiler 7 und 9 teilerfremd sind. "Also" ggT(28,36)=4.

Die kurze Betrachtung über die Komplexitätsordnung von Algorithmen zeigt noch ein weiteres: Je größer die *Fortschritte in der Hardwaretechnologie* werden, (d.h. z.B.

je schneller die verfügbaren Maschinen werden), desto wichtiger werden Fortschritte in der Verbesserung, d.h. *Beschleunigung von Algorithmen* auf der Basis von mehr mathematischem Wissen. Denn:

> Ein schlechter Algorithmus (mit hoher Komplexitätsordnung) erlaubt bei der Erhöhung der Rechenleistung der verfügbaren Maschinen nur eine geringe Ausdehnung des Eingabebereiches des Algorithmus.
>
> Ein guter Algorithmus erlaubt auf besseren Maschinen hingegen die Ausdehnung des Eingabebereiches um ein Beträchtliches.

(Man überlege z.B.: Ein Algorithmus mit Rechenzeit 2^n, der auf einer schlechten Maschine Eingaben bis zur "Größe" $n = 20$ zuläßt, läßt auf einer 64 mal so schnellen Maschine Eingaben bis zur Größe 26 zu, also nur 1.3 mal so große Eingaben. Ein Algorithmus mit Rechenzeit n^2, der auf derselben schlechten Maschine Eingaben bis zur Größe 1024 zuläßt, läßt auf der 64 mal so schnellen Maschine immerhin Eingaben bis zur Größe 8192 zu, also 8 mal so große Eingaben.)

Zusammenfassend also:

Die Bedeutung der Entwicklung neuen mathematischen, algorithmisch brauchbaren Wissens für die Computer-Mathematik, insbesondere Computer-Algebra, wird in der Zukunft nicht abnehmen, sondern immer mehr steigen. Echte Fortschritte in der Lösung der Probleme der Computer-Algebra werden nur durch Kombination bester mathematischer Techniken mit den besten Errungenschaften der Softwaretechnologie erzielt werden können.

In diesem Abschnitt werden wir anhand einiger ausgewählter Beispiele von Algorithmen der Computer-Algebra die obigen Prinzipien demonstrieren und vor allem die jeweiligen zusätzlichen mathematischen Erkenntnisse skizzieren, auf denen der algorithmische Fortschritt beruht.

Wir müssen hier jedoch bemerken, daß es im begrenzten Umfang dieses Abschnittes unmöglich ist, einen Eindruck von der Reichhaltigkeit der mathematisch/algorithmischen Ideen zu vermitteln, die hinter den heutigen Computer-Algebra-Systemen stehen. Immerhin sollte die getroffene Auswahl jedoch unter anderem auch die Einsicht liefern, daß für die Lösung schwieriger Probleme in der Computer-Algebra die Lösung einer Reihe von Unterproblemen und Unterunterproblemen notwendig ist, sodaß heute ein hierarchisch gegliedertes Netz von Algorithmen vorliegt, wo algorithmische Verbesserungen an einer Stelle zahlreiche Konsequenzen für die Lösung hierarchisch übergeordneter Probleme haben. Wohl eines der besten Beispiele dafür ist das Problem der "Quantoren-Elimination" (siehe später den Collins-Algorithmus). Dieses Problem hat durch die grundlegenden Arbeiten von G. Collins seit Anfang der sechziger Jahre die Forschung für fast alle derzeitigen Grundprobleme und Unterprobleme der Computer-Algebra (Arithmetik in verschiedenen algebraischen Bereichen, exaktes Lösen von polynomialen Gleichungen, Faktorisieren von Polynomen etc.) stimuliert bzw. initiiert.

Wir zeigen den Gedanken des hierarchisch gegliederten Netzes von Algorithmen durch Auswahl einiger markanter Punkte steigender Problemallgemeinheit in diesem Netz:

- der Karatsuba-Ofman-Algorithmus zur Multiplikation von Zahlen,
- der Berlekamp-Hensel-Algorithmus zum Faktorisieren von Polynomen,
- die Methode der Gröbner-Basen für Probleme mit multivariaten Polynomen,
- der Collin'sche Algorithmus zur zylindrisch-algebraischen Dekomposition.

Ein anderes derartiges, hierarchisch sehr globales Problem ist das symbolische Integrieren und Summieren bzw. die symbolische Lösung von Differentialgleichungen, auf das wir hier nicht einmal skizzenhaft eingehen können.

5.3 Der Algorithmus von Karatsuba-Ofman zur Multiplikation

Wir betrachten zunächst ein sehr einfaches, grundlegendes Problem, nämlich die Multiplikation beliebig langer ganzer Zahlen in der üblichen Darstellung als Ziffernfolge z.B. zur Basis 10 oder zu irgend einer anderen Basis. In Computer-Algebra-Systemen nimmt man meist als Basis eine Zahl, die in etwa in einem Computerwort Platz hat, z.B. 2^{31} o.ä., weil man dann Darstellungen der Zahlen mit relativ wenig Ziffern bekommt und trotzdem die elementaren Operationen auf Ziffern noch in einer Zeiteinheit möglich sind. Man wird an diesem Problem sehen, daß man selbst bei einfachen und "im Prinzip" längst behandelten Problemen durch Einbringen von zusätzlichem mathematischem Wissen noch entscheidende algorithmische Verbesserungen erzielen kann.

Problem:

Gegeben: x, y zwei Ziffernfolgen,
Gesucht: z eine Ziffernfolge,
sodaß die durch z dargestellte Zahl =
das Produkt der durch x und y dargestellten Zahlen. #

Einfacher Algorithmus:
Die bekannte Methode, wo "jede Ziffer von x mit jeder Ziffer von y multipliziert wird mit geeigneten Übertragen und Verschiebungen". Die Anzahl der Schritte dieser Methode ist $O(L(x).L(y))$ (also $O(L(x)^2)$, wenn $L(x)=L(y)$).

Analyse des Multiplikationsbegriffs:
Wir schreiben x' für die durch die Ziffernfolge x dargestellte Zahl. Der Einfachheit halber nehmen wir als Basis der Darstellung 10 und betrachten zwei gleich lange Ziffernfolgen x und y, die beide gerade Länge n haben mögen. Die Ziffernfolgen lassen sich dann in zwei gleich lange Ziffernfolgen zerschlagen:

```
[     x     ]  =  [  a  |  b  ]
[     y     ]  =  [  c  |  d  ]
   Länge n          Länge n/2  Länge n/2
```

Es gilt:
$x'.y' = (a'.10^{n/2}+b').(c'.10^{n/2}+d') =$
$= (a'.c').10^{n}+(a'.d'+b'.c').10^{n/2}+b'.d'$ (*).

Die Multiplikation von x' und y' ist damit also reduziert auf die 4 Multiplikationen a'.b', a'.d', b'.c' und b'.d', sämtlich von halber Länge. Durch Fortführung dieser Reduktion würde man schließlich zu lauter Multiplikationen von einzelnen Ziffern kommen. Allerdings zeigt eine leichte Komplexitätsanalyse, daß damit keine Beschleunigung erreicht wird, man kommt im wesentlichen genau zur üblichen Methode, d.h. die Komplexitätsordnung bleibt $O(L(x)^2)$.

Zusätzliches mathematisches Wissen:
Es gilt:
$(a'.d'+b'.c') = (a'+b').(c'+d') - a'.c' - b'.d'$ (**).

Dies zeigt, daß man in (*) die *zwei* Multiplikationen a'.d' und b'.c' durch die *eine* Multiplikation $(a'+b').(c'+d')$ der Zahlen $(a'+b')$ und $(c'+d')$ ersetzen kann, weil die in (**) benötigten Produkte a'.c' und b'.d' in (*) sowieso schon einmal zu berechnen sind. Die Zahlen $(a'+b')$ und $(c'+d')$ haben aber (im wesentlichen) die Länge n/2. Es gilt also:

> *Die Multiplikation x'.y' läßt sich zurückführen auf nur 3 Multiplikationen von halber Länge!*

Diese Beobachtung stammt von /Karatsuba,Ofman 1962/. Rekursive Fortführung dieser Reduktion liefert einen entscheidend besseren Algorithmus, für den man

durch leichte Analyse feststellt, daß seine Komplexitätsordung nur mehr $O(L(x)^{\log 3})$ ist. Dieser Algorithmus wird in der Tat in Computer-Algebra-Systemen für die Multiplikation langer Zahlen (die mehr als nur einige Computerworte einnehmen) angewandt.

Besserer Algorithmus (Karatsuba-Ofman 1962):
$z = x \otimes y$:
Bestimme a,b,c,d, sodaß

```
[   x   ] = [ a | b ],   [   y   ] = [ c | d ]
```

$u := a \otimes c$
$v := b \otimes d$
$w := (a \oplus b) \otimes (c \oplus d) \ominus u \ominus v$

```
z := [   u   |       ] ⊖ [       | w |       ] ⊕ [       |   | v ]
Länge:  n       n             n      n/2  n/2        n      n/2  n/2
```

(hier stehen $\oplus$, $\ominus$, $\otimes$ für die Operationen auf den Ziffernfolgen, sodaß z.B. $(x \oplus y)' = x'+y'$, etc.) ∎

Bemerkung: Der Grundgedanke des Karatsuba-Ofman-Algorithmus läßt sich auch für die Multiplikation von Polynomen anwenden.

5.4 Der Berlekamp-Hensel-Algorithmus zur Faktorisierung von Polynomen

Problem:

Gegeben: f — ein Polynom in einer Variablen mit ganzzahligen Koeffizienten

Gesucht: $p_1,\ldots,p_r$ — ganzzahlige Polynome, sodaß $p_1,\ldots,p_r$ irreduzibel und $f = \Pi_{i=1,\ldots,r}\, p_i$ ("irreduzibel" heißt hier: nicht weiter zerlegbar in Faktoren mit ganzzahligen Koeffizienten) ∎

So einfach dieses Problem zu formulieren ist, so schwierig ist es, einen effizienten Algorithmus zur Lösung dieses Problems zu finden. Das Problem hat eine lange Geschichte mit drei wesentlichen Etappen:

> Kronecker's Methode (1882; in der Tat war Kroneckers Arbeit eine Wiederentdeckung einer bereits früher entwickelten Methode),

die Methode von Berlekamp und Zassenhaus (1967 bzw. 1969) und zahlreiche davon ausgehende Arbeiten,

die Methode von A. Lenstra, H. Lenstra, L. Lovász (1982 ff.) und davon ausgehende Arbeiten.

Die zweite und dritte dieser Etappen ist dadurch gekennzeichnet, daß wesentlich neues mathematisches Wissen entwickelt wurde, das eine drastische Effizienzverbesserung bei der Lösung des Problems zuläßt, während die erste Etappe durch ein sehr einfaches mathematisches Wissen charakterisiert ist, welches den Übergang von einem reinen Existenzbeweis (für die irreduziblen Faktoren) zu einem Algorithmus zur Bestimmung der irreduziblen Faktoren erlaubt. Das Faktorisierungsproblem ist also ein schönes Beispiel für das Prinzip "Mehr mathematisches Wissen ⇒ Besserer Algorithmus". Wir besprechen hier beispielhaft nur die erste Etappe und einen Teil der zweiten. (Die dritte Etappe hat (noch) nicht zu praktischen Rechenzeitverbesserungen geführt, weil der organisatorische Overhead der Methode bei "kleinen" Polynomen die prinzipielle Effizienzverbesserung erschlägt.) Für eine ausführlichere Übersicht über Faktorisierungsmethoden siehe /Kaltofen 1982/.

Einfaches Wissen:

(Homomorphie)

Seien f,g,h ganzzahlige Polynome, sodaß f = g.h, und sei a eine ganze Zahl. Dann gilt auch f(a) = g(a).h(a).

(Interpolationspolynom)

Seien $x_0,...,x_n,y_0,...,y_n$ reelle (z.B. ganze) Zahlen. Dann kann man ein Polynom f n-ten Grades konstruieren, sodaß $f(x_i)=y_i$ (für $i=0,...,n$) (das "Interpolationspolynom" zu den Punkten $(x_0,y_0),...,(x_n,y_n)$). Dieses Polynom ist eindeutig bestimmt. ‡

Aus diesen zwei Tatsachen ergibt sich unmittelbar folgender Algorithmus:

Einfacher Algorithmus (Kronecker 1882):

(Initialisiere)

Wähle beliebige, verschiedene ganze Zahlen $a_0,...,a_n$, wo n die größte ganze Zahl kleiner oder gleich ½(Grad von f) ist. Faktorisiere jede der ganzen Zahlen $f(a_0),...,f(a_n)$ vollständig in Primfaktoren.

(Interpoliere)

Für jede Kombination $r_0,...,r_n$,

wo r_0 Primfaktor von $f(a_0)$, ..., r_n Primfaktor von $f(a_n)$ ist,

bestimme das Interpolationspolynom g zu den Punkten $(a_0,r_0),...,(a_n,r_n)$.

Wenn g ein Teiler von f ist:

Wende den Algorithmus rekursiv auf g und f/g an. Dann ist $\{p_1,...,p_t\}$ die Vereinigung der Mengen der irreduziblen Faktoren von g und f/g (und der Algorithmus kann verlassen werden).

(Irreduzibler Fall)

Wenn für keine Kombination ein Faktor g von f erzeugt wurde:

Antwort "f ist irreduzibel". ‡

Die Betrachtung "aller Kombinationen" im Schritt (Interpoliere) ist ein "exponentieller" Vorgang. Das Verfahren ist daher nicht sehr effizient.

Zusätzliches mathematisches Wissen:

(Hensel-Lemma)

Siehe für dieses Lemma über univariate Polynome die Lehrbücher über Algebra. Unter Verwendung (einer algorithmischen Version) dieses Lemmas hat /Zassenhaus 1969/ das obige Faktorisierungsproblem zurückgeführt auf das Faktorisierungsproblem von Polynomen mit Koeffizienten in den endlichen Körpern Z_p. (Diese Reduktion des Problems kann in besonders ungünstigen Fällen allerdings immer noch zu "exponentiellen" Rechenzeiten führen.)

(Lemma über quadratfreie Faktoren)

Ein Polynom a ist genau dann quadratfrei, wenn a und die Ableitung a' relativ prim sind.

Mit diesem einfachen Lemma kann das Faktorisierungsproblem "einfach" (nämlich nur durch die Bestimmung von größten gemeinsamen Teilern) auf die Faktorisierung von quadratfreien Polynomen zurückgeführt werden. (Quadratfreie Polynome sind Polynome, in welchen jeder Faktor nur einmal vorkommt.)

Problem der Faktorisierung quadratfreier Polynome über Z_p:

Gegeben: f..... ein quadratfreies Polynom mit Koeffizienten in Z_p

(Für eine Primzahl p ist Z_p der algebraische Bereich, der aus den Objekten 0,...,(p-1) besteht, mit den Operationen ⊕, ⊖, ⊗, ⊘ (Division) "modulo p", die wie folgt definiert sind:

$x \oplus y$:= Rest bei der Division von x + y durch p,

$x \ominus y$:= Rest bei der Division von x - y durch p,

$x \otimes y$:= Rest bei der Division von x . y durch p,

$x \oslash y$:= das Element $z \in Z_p$, sodaß $y \otimes z = x$; dieses z kann mit dem Euklid'schen Algorithmus bestimmt werden.)

Gesucht: $p_1,...,p_t$ ... irreduzible Polynome mit Koeffizienten in Z_p

sodaß $f = \Pi_{i=1...t}\, p_i$. ‡

Zusätzliches mathematisches Wissen (Berlekamp 1969):

(Faktorentrennung)

Seien $p_1,....p_r$ die irreduziblen Faktoren eines Polynoms f über Z_p und sei v ein Polynom mit folgender Eigenschaft (ein "Trennpolynom"):

Grad von v < Grad von f und
f teilt v^p - v (FT1)

Dann gilt:

Jeder irreduzible Faktor p_j von f teilt eines der Polynome
v-0, v-1, ..., v-(p-1). (FT2)

Außerdem:

Es gibt $s_1,....s_t \in Z_p$, sodaß
für alle i = 1,...,t: p_i teilt (v-s_i) und
für alle i,j = 1,...,t mit $s_i \neq s_j$:
p_i teilt den größten gemeinsamen Teiler von f und v-s_i.
p_j teilt den größten gemeinsamen Teiler von f und v-s_i nicht.
(v-s_i "trennt" also p_i und p_j!)

(Vektorraum der Trennpolynome)

Die Trennpolynome für f bilden einen Vektorraum. #

Der Beweis dieser Hilfssätze ist nicht schwer (unter Verwendung einiger grundliegender Rechengesetze zum Rechnen modulo p). Mit Hilfe des Wissens (Vektorraum der Trennpolynome) und einigen leichten Beweisschritten kann man zeigen, daß man eine Basis $v^{(1)},...,v^{(r)}$ für den Vektorraum aller (Koeffizientenvektoren der) Trennpolynome durch den Gauß'schen Algorithmus als Basislösungsvektoren aus folgendem homogenen linearen Gleichungssystem erhält:

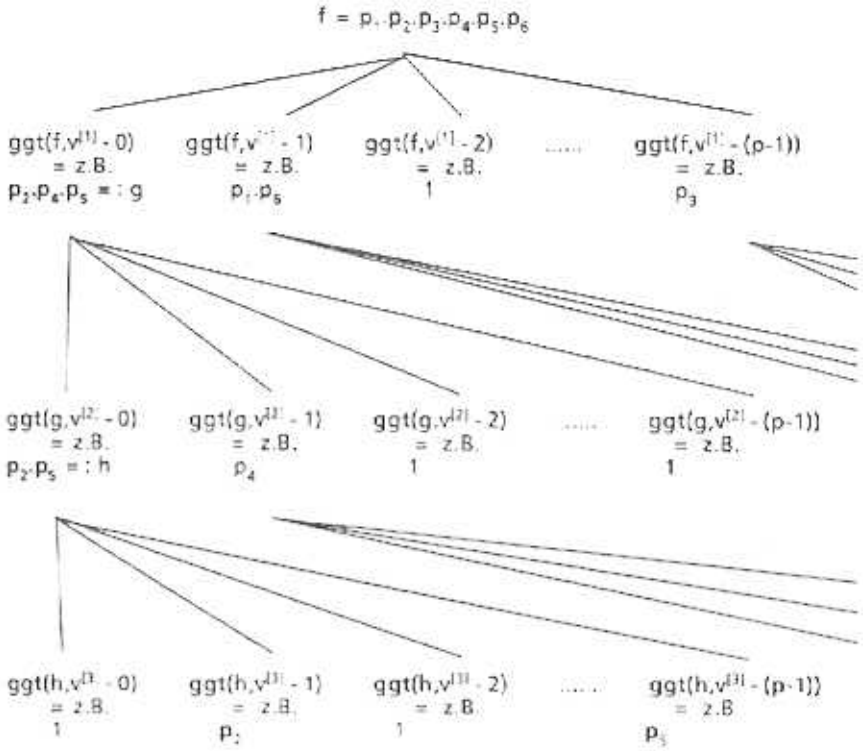
$v.(Q - I) = 0$,

wobei I die Einheitsmatrix ist und Q eine Matrix ist, deren Elemente $Q_{k,\ell}$ wie folgt gebildet werden:

$Q_{k,\ell}$:= der Koeffizient, der bei der Division von $x^{p(k-1)}$ durch f bei der Potenz $x^{(\ell-1)}$ steht ($1 \le k,\ell \le n$, n: = Grad von f).

(In der Tat kann man auch zeigen, daß die Anzahl r der linear unabhängigen Basisvektoren gleich der Anzahl r der Faktoren von f ist.)

Die Kenntnis der durch die Koeffizientenvektoren $v^{(1)},...,v^{(r)}$ bestimmten Trennpolynome (deren Bestimmung durch den Gauß'schen Algorithmus in polynomialer, nämlich kubischer Zeit möglich ist) ermöglicht nun im wesentlichen unter Anwendung des Wissens (FT1) und (FT2) die Bestimmung der Primfaktoren durch folgenden "Faktorentrennvorgang", den wir zunächst anschaulich erklären ("ggt" steht für "großten gemeinsamen Teiler"):



D.h. die einzelnen Primfaktoren von f werden nach und nach durch Bildung des grossten gemeinsamen Teilers mit den Trennpolynomen $v^{(1)},...,v^{(r)}$ wie in einer Kaskade mit "Trennfiltern" getrennt. Die genauen Überlegungen, warum diese Kaskade, im wesentlichen auf der Grundlage des Wissens (FT2) und (FT3), die Trennung tatsächlich immer leistet, können wir hier aus Platzgründen nicht geben. Der geubte Leser ist vielleicht in der Lage, die Detailüberlegungen in die obige Skizze einzufüllen. Zusammenfassend ergibt sich folgender Algorithmus, dessen Komplexität im wesentlichen von der Ordnung $O(pn^3)$ ist, wobei n der Grad des Eingabepolynoms f ist.

Algorithmus (Berlekamp 1969)
zur Faktorisierung von quadratfreien Polynomen über Z_p:

(Bestimmung von Trennpolynomen)

Bestimme wie oben angegeben die Matrix Q
und dann mit dem Gauß'schen Algorithmus eine Basis für den Vektorraum aller Lösungen der homogenen linearen Gleichung v(Q - I) = 0. Sei r die Anzahl der Vektoren in der Basis und seien $v^{(1)},...,v^{(r)}$ die Basisvektoren selbst.

(Trennen der Primfaktoren)

F := {f} (Menge der bereits bekannten Faktoren)

Falls r=1 (f ist irreduzibel):

$p_1 := f$.

Falls r>1 (f zerfällt in Primfaktoren):

Für j := 2,...,r:

Für alle h∈F:

Für s := 0,...,p-1 :

g := ggt(h,$v^{[j]}$-s),

Füge g zu F hinzu.

Streiche aus F alle Polynome, die Vielfaches von einem anderen Polynom in F sind. Falls F jetzt genau r Faktoren enthält, gehe aus der Schleife.

Als $p_1,...,p_r$ kann man nun die Polynome in F nehmen. †

Beispiel (aus Knuth 1969, S 424):

Zu Faktorisieren sei das Polynom $f = x^8 + x^6 + 10x^4 + 10x^3 + 8x^2 + 2x + 8$ über Z_{13}. Wir bestimmen zuerst die Matrix Q. Die erste Reihe von Q ist (1,0,...,0), weil x^0 dividiert durch f wieder x^0 ist. Die nächste Reihe ist (2,1,7,11,10,12,5,11), weil x^{13} dividiert durch f gleich $(11x^7 + 5x^6 + 12x^5 + 10x^4 + 11x^3 + 7x^2 + x + 2)$ ist (alle Rechnungen sind über Z_{13} durchzuführen!). Genau so bestimmt man die weiteren Zeilen von Q durch Division von x^{26}, x^{39} etc. durch f. (In der Tat kann man das Ergebnis für x^{k+1} durch sehr einfache Rechnung aus dem Ergebnis für x^k erhalten!) Insgesamt erhält man so

$$Q = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 2 & 1 & 7 & 11 & 10 & 12 & 5 & 11 \\ 3 & 6 & 4 & 3 & 9 & 4 & 7 & 2 \\ 4 & 3 & 6 & 5 & 1 & 8 & 2 & 3 \\ 2 & 11 & 8 & 8 & 3 & 1 & 3 & 11 \\ 6 & 11 & 8 & 6 & 2 & 7 & 10 & 9 \\ 5 & 11 & 7 & 10 & 0 & 11 & 7 & 12 \\ 3 & 3 & 12 & 5 & 0 & 11 & 9 & 12 \end{pmatrix}$$

Durch Anwenden des Gauß'schen Algorithmus erhält man z.B. die folgenden drei linear unabhängigen Lösungsvektoren für das Gleichungssystem v.(Q - I)=0:

$v^{[1]} = (1\,0\,0\,0\,0\,0\,0\,0)$,

$v^{[2]} = (0\,5\,5\,0\,9\,5\,1\,0)$,

$v^{[3]} = (0\,9\,11\,9\,10\,12\,0\,1)$.

Diese Vektoren spannen den gesamten Lösungsraum auf. Es gibt genau so viele irreduzible Faktoren von f als linear unabhängige Lösungsvektoren im Lösungsraum von v.(Q - I)=0, also 3. Man berechnet nun den ggT(f,$v^{[2]}$ - s) für 0≤s<13, wo $v^{[2]}$

$= x^6 + 5x^5 + 9x^4 + 5x^3 + 5x$. ($v^{[1]}$ braucht man nicht zu betrachten!) Das ergibt

$f_1 := x^5 + 5x^4 + 9x^3 + 5x - 5$ für s = 0 und

$f_2 := x^3 + 8x^2 + 4x + 12$ für s = 2.

Alle anderen ggT sind 1. Damit hat man zwei Faktoren und muß noch ggT(f_1,$v^{[3]}$ -s) betrachten für 0≤s<13, wo $v^{[3]} = x^7 - 12x^5 + 10x^4 + 9x^3 + 11x^2 + 9x$. Das ergibt

$f_3 := x^4 + 2x^3 + 3x^2 + 4x + 6$ für s = 6 und

$f_4 := x + 3$ für s = 8.

Alle anderen ggT sind wieder 1. f_1 kann man jetzt streichen. Die vollständige Faktorisierung von f ist also

$f = (x^4 + 2x^3 + 3x^2 + 4x + 6)(x^3 + 8x^2 + 4x + 12)(x + 3)$.

5.5 Die Methode der Gröbner-Basen für Probleme mit Systemen multivariater Polynome

Allgemeine Strategie:

Gegeben sei eine (endliche) Menge F von multivariaten Polynomen mit Koeffizienten aus einem Körper (z.B. Q). In der algebraischen Geometrie (die u.a. die mathematischen Grundlagen für CAD, Computer-Geometrie, geometrische Probleme der Roboterprogrammierung etc. liefert) studiert man die Lösung verschiedener grundlegender Probleme für solche Polynommengen F: z.B. die Bestimmung aller Lösungen des durch F gegebenen Systems algebraischer Gleichungen (die Lösungen mögen dabei in gewissen Erweiterungskörpern liegen); die Zerlegung der Lösungsmannigfaltigkeit des Gleichungssystems in "irreduzible" Teilmannigfaltigkeiten (Problem der "Primärdekomposition", das als eine Verallgemeinerung des Faktorisierungsproblems univariater Polynome betrachtet werden kann; geometrisch entspricht das der Zerlegung geometrischer Gebilde in einfachste Teilgebilde); Aussagen über die Dimension der Lösungsmannigfaltigkeit; Bestimmung einer endlichen Basis für den "Modul" aller Polynomlösungen von linearen Gleichungen, die als Koeffizienten die Polynome aus F haben; vollkommene Beherrschung der Arithmetik in den algebraischen Bereichen, die aus dem Grundkörper durch Hinzunahme der Lösungen von Gleichungssystemen F entsteht; usw.

Mit der Methode der Gröbner-Basen werden diese Probleme in folgenden zwei Schritten gelöst:

1. Transformation der gegebenen Polynommenge F in eine "Standardform" G (eine *"Gröbner-Basis"*), ohne die wesentlichen algebraischen Eigenschaften (z.B. die Lösungsmannigfaltigkeit) von F zu verändern.

2. Lösung des entsprechenden Problems für G (wobei die Lösung der angeführten Probleme für Gröbner-Basen meist "leicht" ist im Vergleich zur Lösung für beliebige F).

Die Methode der Gröbner-Basen wurde in /Buchberger 1965,1970/ eingeführt und seither in einer Reihe von Arbeiten weiter ausgebaut. Für eine Zusammenfassung mit ausführlichen Literaturhinweisen siehe /Buchberger 1985/.

Definitionen:

(Im folgenden verwenden wir folgende Variablen:

f,g,h ... Polynome mit n Variablen und Koeffizienten aus einem Grundkörper K,

F,G ...(endliche) Mengen von Polynomen,

a,b,c ... Elemente aus dem Grundkörper,

s,t,u,v ... Potenzprodukte (das sind spezielle Polynome der Gestalt $x_1^{i_1}...x_n^{i_n}$,

z.B. $x_1^3x_2x_3^5x_4^2$))

(Reduktion):

$g \rightarrow_F h$ (lies: g *reduziert* modulo F auf h) : $\Leftrightarrow$

es existiert ein $f \in F$ und ein Potenzprodukt t, das in g vorkommt, sodaß

t ein Vielfaches u.s des größten Potenzproduktes s von f ist

und $h = g - (a/b).u.f$, wobei

a der Koeffizient von t in g ist und

b der führende Koeffizient von f ist. ∎

(Die Begriffe "größtes Potenzprodukt" und "führender Koeffizient" beziehen sich dabei auf eine fixe totale Ordnung der Potenzprodukte, die in gewissen Grenzen willkürlich gewählt werden kann. Z.B. ist - für drei Variable x,y,z - eine mögliche Anordnung: $1,x,y,z,x^2,xy,xz,y^2,yz,z^2,x^3,...$. D.h. es wird hier zuerst nach dem Grad und innerhalb eines Grades lexikographisch geordnet.)

Beispiel: $F := \{2x^2y - x + 1, xy^2 + 3y\}$, $g := x^3y^2 - 5x^2y^2 + 3x$:

$g \rightarrow_F h_1 := x^3y^2 - 5/2\ xy + 5/2\ y + 3x$ (mit $f := 2x^2y - x - 1$ und $t := x^2y^2$).

Aber auch $g \rightarrow_F h_2 := x^3y^2 + 15xy + 3x$ (mit $f := xy^2 + 3y$ und $t := x^2y^2$).

(Zum Vergleich: Ein Schritt in der Division eines univariaten Polynoms g durch ein Polynom f ist ein Spezialfall des obigen sehr allgemeinen Reduktionsschrittes. Während jedoch fortgesetzte Divisionsschritte bei der bekannten univariaten Division zu einem eindeutigen "Rest" führen, kommt man durch fortgesetzte Reduktion ausgehend von einem multivariaten f im allgemeinen zu vielen verschiedenen "Resten", d.h. Polynomen, die modulo F nicht mehr weiter reduziert werden können.

Sicher ist lediglich, daß jede fortgesetzte Reduktion nach endlich vielen Schritten abbricht, wie man unschwer beweisen kann.)

(Gröbner-Basen)

Eine (endliche) Polynommenge F ist eine *Gröbner-Basis* :$\Leftrightarrow$

für alle Polynome g:

bei fortgesetzter Reduktion von g modulo F entsteht immer derselbe Rest.

Beispiel: Das F im vorherigen Beispiel ist keine Gröbner-Basis, denn z.B. $g := x^2y^2$ reduziert auf die zwei verschiedenen Reste $h_1 := \frac{1}{2}xy - \frac{1}{2}y$ und $h_2 := -3xy$. (Überlege anhand der Definition (Reduktion), daß weder h_1 noch h_2 weiter reduzierbar sind modulo F.)

Beispiel eines Problems, das man mit Gröbner-Basen lösen kann: Systeme von algebraischen Gleichungen.

Gegeben: F ... eine endliche Menge von Polynomen in n Variablen über einem Körper K (z.B. Q).

Gesucht: 1. Entscheide, ob F (in einem geeigneten Erweiterungskörper E des Grundkorpers, z.B. in C) losbar ist.

2. Falls F losbar ist, entscheide, ob F endlich viele oder unendlich viele Lösungen hat.

3. Falls F endlich viele Lösungen hat, Bestimmung aller Lösungen von F.

(Eine Lösung von F ist dabei ein n-Tupel $(a_1,...,a_n)$ von Elementen aus E, sodaß für alle $f \in F$: $f(a_1,...,a_n) = 0$.)

Mathematisches Wissen:

Für Gröbner-Basen gilt:

(GB: Losbarkeit)

Das durch G bestimmte Gleichungssystem ist losbar (im algebraischen Abschluß des Grundkorpers) $\Leftrightarrow$ In G kommt kein konstantes Polynom vor.

(GB: Endliche Lösungsmenge)

Das durch G bestimmte Gleichungssystem hat nur endliche viele Lösungen (im algebraischen Abschluß des Grundkorpers) $\Leftrightarrow$ Für alle $i = 1,...,n$ existiert ein e, sodaß x_i^e kommt als größtes Potenzprodukt in einem der $g \in G$ vor.

(GB: Elimination)

(Falls die Potenzprodukte lexikographisch angeordnet werden ohne Berücksichtigung des Grades, wobei x_1 vor x_2, x_2 vor x_3, ..., x_{n-1} vor x_n kommen möge und falls G nur endlich viele Lösungen besitzt:)

G enthält einige Polynome $g_1^{(1)}, g_2^{(1)}, ..., g_{k1}^{(1)}$, welche nur von x_1 abhängen, darunter ein Polynom niedrigsten Grades, von dem alle anderen Vielfache sind,

G enthält einige Polynome $g_1^{(2)}, g_2^{(2)}, \ldots, g_{k_2}^{(2)}$, welche nur von x_1, x_2 abhängen,

...

G enthält einige Polynome $g_1^{(n)}, g_2^{(n)}, \ldots, g_{k_n}^{(n)}$, welche von $x_1, x_2, \ldots, x_n$ abhängen.

(Diese Aussagen könnte man, unter Verwendung von mehr algebraischer Terminologie, viel genauer machen. Die Beobachtung (GB: Elimination) wurde das erstemal in /Trinks 1978/ gemacht. Eine Alternative dazu, die für beliebige Anordnungen der Potenzprodukte gilt, findet sich in /Buchberger 1970/.)

Aus diesem Wissen ergibt sich folgender "Algorithmus" zur Lösung des obigen Problems:

Algorithmus:

(Gröbner-Basis herstellen)

Transformiere die gegebene Menge F in eine zugehörige Gröbner-Basis G.

(Endlichkeit der Lösungsmenge testen)

Falls für ein $i = 1, \ldots, n$ kein Potenzprodukt der Gestalt x_i^{α} unter den höchsten Potenzprodukten der $g \in G$ vorkommt:

Antwort "F hat unendlich viele Lösungen".

(Endlich viele Lösungen bestimmen)

Wähle das Polynom g niedrigsten Grades in G, welches nur von x_1 abhängt und bestimme alle Nullstellen des Polynoms.

Für alle Nullstellen a_1 von g:

Setze a_1 in alle übrigen Polynome von G ein. Man erhält ein Gleichungssystem in (n-1) Variablen, das man rekursiv nach derselben Methode lösen kann. Jede Lösung $(a_2, \ldots, a_n)$ dieses Gleichungssystems kann man mit a_1 zusammensetzen zu einer Lösung $(a_1, \ldots, a_n)$ von G (und damit F). #

Der letzte Schritt (Endlich viele Lösungen bestimmen) läßt sehr viele Varianten und Verfeinerungen zu, die für die Effizienz und praktische Durchführung wichtig sind, auf die wir hier aber unmöglich eingehen können. Wir haben "Algorithmus" mit Anführungszeichen geschrieben, weil wir vom ersten Schritt (Gröbner-Basis herstellen) ja noch nicht gezeigt haben, ob er tatsächlich algorithmisch durchführbar ist. In der Tat gibt es einen einfachen, inkonstruktiven Existenzbeweis für Gröbner-Basen, der in seiner Idee für den analogen Fall der Standard-Basen bei Potenzreihen bereits (/Hironaka 1964/) vor Einführung der Gröbner-Basen bekannt war. Dieser Beweis geht im wesentlichen so: Man wählt als Gröbner-Basis G zu F

$G := \{g \in Ideal(F) \mid g$ kann nicht durch ein anderes Polynom $f \in Ideal(F)$ reduziert werden$\}$

und zeigt alle gewünschten Eigenschaften. Hier ist Ideal(F) das durch "F erzeugte Polynomideal" (zu diesem Begriff siehe die Algebra-Lehrbücher). Dieser Beweis gibt aber keine Möglichkeit zur algorithmischen Bestimmung von Gröbner-Basen, denn man müßte "alle" (unendlich vielen) $g \in Ideal(F)$ durchsuchen und für jedes solche g für die unendlich vielen f untersuchen, ob g durch f reduziert werden kann oder nicht. Erst das folgende zusätzliche mathematische Wissen erlaubt die algorithmische Konstruktion von Gröbner-Basen:

Mathematisches Wissen (Buchberger 1965):

(Endliche Charakterisierung von Gröbner-Basen)

Sei "Rest(f,F) ein Algorithmus, der durch Iterieren des Reduktionsschrittes ein beliebiges Polynom f modulo einem beliebigen F zu einem Rest (nicht mehr weiter reduzierbaren Polynom) reduziert. Dann gilt:

G ist eine Gröbner-Basis $\Leftrightarrow$

Für alle $g_1, g_2 \in G$: $Rest(S\text{-}Polynom(g_1, g_2), G) = 0$.

(Hier ist das "S-Polynom von g_1 und g_2" wie folgt definiert:

$S\text{-}Polynom(g_1, g_2) = a_2 \cdot (kgV(s_1, s_2)/s_1) \cdot g_1 - a_1 \cdot (kgV(s_1, s_2)/s_2)) \cdot g_2$,

wobei a_i der führende Koeffizient von g_i

und s_i das höchste Potenzprodukt von g_i ist $(i = 1, 2)$;

kgV steht für "kleinstes gemeinsames Vielfaches".) #

Der Beweis dieses Satzes ist bedeutend schwieriger als der inkonstruktive Existenzbeweis und ist im wesentlichen kombinatorisch, siehe /Buchberger 1965, 1970/. Man beachte, daß dieses Wissen nunmehr eine algorithmische Charakterisierung der Gröbner-Basen erlaubt, denn es ist nur mehr der Test von *endlich vielen* Paaren (g_1, g_2) von Polynomen notwendig und jeder einzelne Test besteht einfach in der Anwendung von zwei Algorithmen, nämlich "S-Polynom" und "Rest". Das obige Wissen gestattet aber nicht nur einen algorithmischen Test, ob eine gegebene Polynommenge G eine Gröbner-Basis ist, sondern auch die Konstruktion von Gröbner-Basen G zu beliebig vorgegebenen Polynommengen F. Genauer kann man damit folgendes Problem algorithmisch lösen:

Problem.

Gegeben: F ... eine endliche Menge multivariater Polynome.

Gesucht: G ... eine endliche Menge multivariater Polynome,

sodaß F und G dasselbe Ideal erzeugen (und damit insbesondere dieselbe Lösungsmenge haben) und G eine Gröbner-Basis ist. #

Algorithmus (Buchberger 1965):

$G := F$

$B := \{\{f_1,f_2\} \mid f_1,f_2 \in G,\ \neg(f_1 = f_2)\}$
<u>while</u> $B \neq \emptyset$ <u>do</u>
  $\{f_1,f_2\} :=$ ein Paar aus B
  $B := B - \{\{f_1,f_2\}\}$
  $h :=$ Rest(S-Polynom(f_1,f_2),G)
  <u>if</u> $h \neq 0$ <u>then</u>
    $B := B \cup \{\{g,h\} \mid g \in G\}$
    $G := G \cup \{h\}$!

Der Block nach der Abfrage "h≠0?" sorgt dafür, daß, wenn ein S-Polynom nicht auf 0 reduziert, der entsprechende Rest zur Basis hinzugenommen wird, um damit die Reduktion auf 0 zu erzwingen. Es ist ein nicht-triviales Problem zu beweisen, daß diese "Vervollständigung" der Basis nur endlich oft geschehen kann, der Algorithmus also nach endlich vielen Schritten abbricht. (Eine kombinatorisch befriedigende Lösung dieses Problems ist mit dem Lemma von /Dickson 1913/ möglich.) Auch dieser Algorithmus läßt viele Varianten und Verfeinerungen zu.

Beispiel:
Für das folgende Gleichungssystem $F := \{4x^2+xy^2 - z + \frac{1}{4},\ 2x + y^2z + \frac{1}{2},\ x^2z - \frac{1}{2}x - y^2\}$ ergibt sich durch Anwenden des obigen Algorithmus als zugehörige Gröbner-Basis G (wenn man z vor y vor x anordnet):

$G := \{z^7 - 1/2\,z^6 + 1/16\,z^5 - 13/4\,z^4 + 75/16\,z^3 - 171/8\,z^2 + 133/8\,z - 15/4,$
$y^2 - 19188/497\,z^6 + 318/497\,z^5 - 4197/1988\,z^4 - 251555/1988\,z^3 - 481837/1988\,z^2 + 1407741/1988\,z - 297833/994,$
$x + 4638/497\,z^6 - 75/497\,z^5 + 2111/3976\,z^4 + 61031/1988\,z^3 + 232833/3976\,z^2 - 85042/497\,z + 144407/1988\}$

(Man beachte: alle Schritte in obigem Algorithmus sind über dem ursprünglichen Grundkörper, z.B. Q, möglich. Bis hierher ist also noch keine Körpererweiterung notwendig!)

Ein Gleichungssystem dieser Art kann man durch "sukzessive Elimination" lösen wie im Schritt (Endlich viele Lösungen bestimmen) beschrieben. Hier beachte man aber, daß man nun entweder numerisch rechnen muß (d.h. mit Gleitkommazahlen und damit mit Rundungsfehlern, die sich beim Einsetzen in die nächsten Gleichungen fortpflanzen) oder man muß in entsprechenden Körpererweiterungen algebraisch weiterrechnen. Für letzteres sind die algorithmischen Techniken zwar bekannt, siehe /Loos 1982/ für eine Übersicht und zum Teil neue Algorithmen, im allgemeinen ist das Rechnen in algebraischen Körpererweiterungen jedoch relativ aufwendig. Die Berechnung von Gröbner-Basen selbst ist grundsätzlich ein aufwendiges

Problem, da sich, wie anfangs erwähnt, viele sehr komplexe Probleme, nämlich "inhärent" komplexe Probleme, auf die Berechnung von Gröbner-Basen zurückführen lassen. Man muß bei solchen Problemen mit "exponentiellen" Rechenzeiten rechnen (siehe /Mair,Meyer 1981/). Trotzdem ist noch ein weites Feld für praktische Verbesserungen offen, insbesondere in der Kombination dieser algebraischen Techniken mit numerischen. Ein weiteres Beispiel wurde im Abschnitt *"Problemlösepotenz von Computer-Algebra-Systemen. Einige Beispiele"* gegeben, welches zeigt, daß man Gröbner-Basen auch für Gleichungssysteme mit "Parametern" berechnen kann, was grundsätzlich den Bereich der numerischen Mathematik übersteigt.

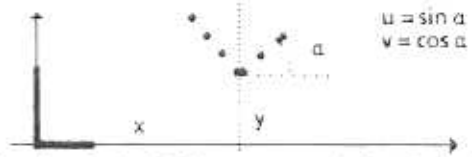
5.6 Der Collins-Algorithmus zur zylindrisch-algebraischen Dekomposition

Motivation:

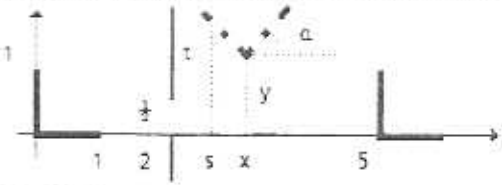
In der Roboter-Programmierung ist eines der (derzeit wichtigsten) Probleme die Bestimmung von kollisionsfreien Wegen von Objekten in der Gegenwart von Hindernissen. Ein sehr einfaches Beispiel einer solchen Aufgabe (in der Ebene) ist:

Das Objekt L soll von der Position A durch die Öffnung O hindurch in die Position B gebracht werden, ohne mit dem Hindernis H zu kollidieren.

Jede Zwischenposition des Objektes kann durch die Translation des Objektemittelpunktes von der Ausgangslage und durch (z.B. Sinus und Cosinus des) Drehwinkels gegenüber seiner Ausgangsorientierung beschrieben werden:



also im wesentlichen durch vier Zahlen x,y,u,v (mit der Zusatzbedingung $u^2+v^2=1$). Betrachten wir z.B. folgende konkrete Angaben für obiges Problem:



Die Eigenschaft "das Objekt kollidiert in der durch x,y,u,v bestimmten Position nicht

mit dem Hindernis" kann durch folgende Formel beschrieben werden:

$$F := \neg(\exists s,t)(B'(s,t,x,y,u,v) \wedge H(s,t))$$

d.h. kein Punkt liegt zugleich auf dem Objekt in der durch x,y,u,v bestimmten Position und dem Hindernis, wobei

$$B'(s,t,x,y,u,v) := (\exists s',t')(B(s',t') \wedge (s,t)=(s',t')\begin{pmatrix} v & u \\ -u & v \end{pmatrix}+(x,y) \wedge u^2+v^2=1)$$

d.h. ein Punkt liegt auf dem Objekt in der durch x,y,u,v bestimmten Position, wenn er aus einem Punkt des Objektes in Ausgangslage durch Anwendung der durch x,y,u,v bestimmten Transformation entsteht, wobei

$$B(s',t') := (0 \le s' \le 1 \wedge t'=0) \vee (s'=0 \wedge 0 \le t' \le 1)$$

und

$$H(s,t) := (s=2 \wedge (t \le 0 \vee t \ge 1))$$

Es kann nun gezeigt werden, siehe /Schwarz,Sharir 1982/, daß das Problem, einen kollisionsfreien Weg zu finden (d.h. (x,y,u,v) durch eine stetige Transformation von der Ausgangsposition (0,0,0,1) in die Endposition (5,0,0,1) überzuführen), reduziert werden kann auf das Problem, eine "zylindrisch-algebraische Dekomposition" des R^m bzw. R^k zu finden. (m ist die Anzahl der in F vorkommenden Variablen, in unserem Beispiel m = 6; k ist die Anzahl der in F vorkommenden freien Variablen, in unserem Beispiel k = 4; nämlich x,y,u,v sind frei.) Die zylindrisch-algebraische Dekomposition muß dabei für die in F vorkommenden Polynome "vorzeicheninvariant" sein. (Um kein Mißverständnis aufkommen zu lassen: Man beachte, daß es hier um eine Dekomposition des R^m bzw. R^k geht und nicht um eine Dekomposition des R^2 oder R^3, in welchem sich die ursprünglichen Objekte befinden!). Dann muß man folgende Schritte ausführen:

1. Bestimme die "Zelle" A der Dekomposition von R^k, in welcher die Ausgangsposition liegt.
2. Bestimme die "Zelle" E, in welcher die Endposition liegt.
3. Entscheide, ob es eine Folge von Zellen $C_1,...,C_k$ gibt, sodaß alle C_i die Formel F "erfüllen" (d.h. nur Positionen enthalten, die nicht kollidieren), (für i = 1,...,k)
   $C_1 = A$, $C_k = E$,
   C_i ist zu C_{i+1} "benachbart" (für i = 1,...,k-1).
4. Wenn es keine solche Folge gibt:
   Antworte "Es existiert kein kollisionsfreier Pfad".
5. Wenn es eine solche Folge gibt:
   Konstruiere einen Pfad durch die Zellen $C_1,...,C_k$.

Man sieht hier, daß die Konstruktion von "zylindrisch-algebraischen Dekompositionen" für höherdimensionale Räume zu einem zentralen Problem wird. In der Tat

lassen sich noch eine ganze Reihe von anderen grundliegenden Problemen der Roboterprogrammierung auf diese Konstruktion zurückführen.

Viel allgemeiner läßt sich diese Konstruktion verwenden, um eine beliebige Formel der "Theorie der reell abgeschlossenen Körper" in eine äquivalente quantorenfreie Formel umzuwandeln und dann für konkrete Werte der freien Variablen zu entscheiden. Die Formeln dieser Theorie sind aber allgemein genug, um eine ungeheure Fülle von interessanten Sätzen über (geometrische) Sachverhalte im R^n auszudrücken. Unter anderem sind Aussagen über die Lösbarkeit in R von beliebigen algebraischen Gleichungs- und Ungleichungssystemen in dieser Theorie möglich. Der Begriff der "zylindrisch-algebraischen Dekomposition" und ein Algorithmus zur Konstruktion solcher Dekompositionen wurde 1973 von G. Collins eingeführt, siehe /Arnon,Collins,McCallum 1984/ für die neueste und ausführlichste Version. Der Collins'sche Algorithmus ist beweisbar besser (d.h. komplexitätsordnungsmäßig besser) als ein älteres Verfahren von /Tarski 1948/. Der Collins'sche Algorithmus löst wohl eine der allgemeinsten Problemstellungen der Computer-Algebra und greift in seinen algorithmischen Details auf faktisch alle anderen algebraischen Algorithmen zurück. Er war und ist somit sicher einer der für die Entwicklung der Computer-Algebra wichtigsten Algorithmen. Seine praktische Relevanz für eine breite Klasse von modernen Ingenieurproblemen wird erst in den allerletzten Jahren erkannt und es ist zu erwarten, daß in nächster Zeit aufbauend auf diesem Verfahren einige Durchbrüche in der Anwendung erzielt werden. Im Augenblick setzt die große Komplexität des Verfahrens der Anwendung noch Grenzen.

Wir geben jetzt eine Formulierung des durch den Collins-Algorithmus gelösten Problems und dann eine sehr grobe Skizze des Algorithmus und des zugrundeliegenden mathematischen Wissens.

Problem:

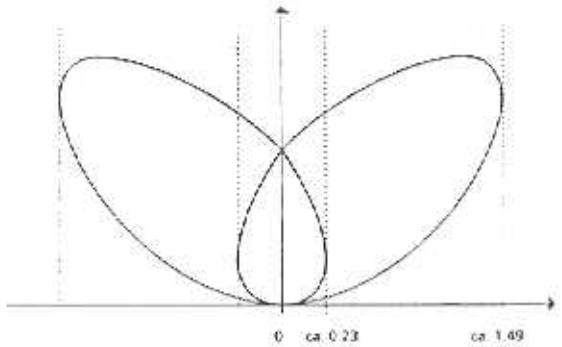
Gegeben: A ... eine Menge von Polynomen (in r Variablen) mit ganzzahligen Koeffizienten.

Gesucht: I ... eine endliche Menge von (Beschreibungen für) "Zellen" (zusammenhängende Punktmengen) im R^r,

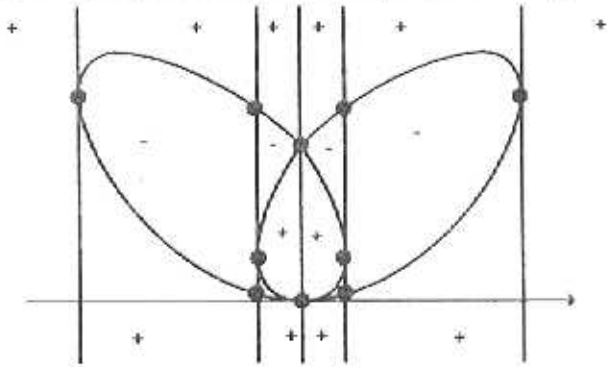
sodaß die Zellen von I zusammen eine 'zylindrisch-algebraische Dekomposition" des R^r ergeben , auf welcher alle Polynome von A "vorzeicheninvariant" sind. ∎

Bevor wir die Definitionen für wenigstens einige der vorkommenden Begriffe angeben, geben wir folgendes

Beispiel (aus /Arnon,Collins,McCallum 1984/):
Sei $A := \{y^4 - 2y^3 + y^2 - 3x^2y + 2x\}$. Das Nullstellengebilde von A hat in etwa folgende Gestalt:



Eine zugehorige "zylindrisch-algebraische Dekomposition" des R^2, auf welcher A "vorzeicheninvariant" ware, besteht aus folgenden Zellen: den dick gezeichneten Punkten, den Kurvenstucken, sowie den zusammenhängenden weissen Flachen zwischen den Punkten und Kurvenstücken in folgender Zeichnung:



Man beachte, daß das obige Problem nicht einfach durch numerisches, approximatives Ausrechnen von Punkten der Kurve an einigen Stellen gelöst werden kann, weil es hier um globale Eigenschaften der Kurve geht, z.B. Anzahl der Zellen, Nachbarschaft von Zellen, etc. Ein noch so leichtes "Verrutschen" von Werten, z.B. Verschieben einer der eingezeichneten vertikalen Linien würde diese Eigenschaften drastisch verändern. Eine Möglichkeit, mit irrationalen Zahlen im Computer zu rechnen, ist die Darstellung von solchen Zahlen (falls sie "algebraisch" sind, d.h. einer Polynomgleichung mit rationalen Koeffizienten genügen) durch Angabe eines Intervalls mit rationalen Endpunkten, in welchem die Zahl liegt, und eines Polynoms, welches

diese Zahl als Wurzel hat (siehe z.B. /Loos 1982/). So kann z.B. $\sqrt{2}$ durch die Intervallendpunkte 1 und 2 und durch das Polynom $x^2 + 0x - 2$ dargestellt werden, d.h. insgesamt durch die 5 rationalen Zahlen (1,2,-2,0,1). Diese Information ist endlich und bestimmt $\sqrt{2}$ eindeutig (d.h. keine andere reelle Zahl liegt im Intervall (1,2) und erfüllt das Polynom $x^2 - 2$). Für das Rechnen mit so dargestellten reellen Zahlen muß man dann Algorithmen angeben, die aus den Darstellungen von zwei reellen Zahlen die Darstellung der Summe dieser beiden Zahlen rechnen etc. (Allerdings kann man hier wesentliche Vereinfachungen treffen, die darauf hinauslaufen, daß man das darstellende Polynom nicht jedesmal neu berechnen muß.) Auch ist ein Problem, wie man nun die einzelnen Zellen (oder wenigstens die interessierenden Eigenschaften der Zellen) einer vorzeicheninvarianten zylindrisch-algebraischen Dekomposition durch ein endliches Stück an Information darstellen kann. Solche Informationen sind z.B.

Die Angabe eines *"Testpunktes"* $a \in \mathbf{R}^r$, der in der Zelle liegt, und dessen Komponenten sämtlich algebraische Zahlen sind. (Für viele Fragen genügt diese Darstellung, z.B. fur die Frage, ob eine Zelle C eine Formel F erfüllt - weil wegen der Vorzeicheninvarianz die Erfullung der Formel ja nicht vom gewählten Punkt innerhalb der Zelle abhängt.)

Die Angabe, durch welche Vereinigungs-, Durchschnitts- und Komplementbildung die Zellen aus Mengen der Art

$\{x \in \mathbf{R}^r \mid A(x) \geq 0\}$, wo A ein Polynom ist,

gebildet werden kann (Darstellung der Zellen als *"semialgebraische"* Mengen).

Die Angabe systematischer Namen (*"Indizes"*) für Zellen und die Angabe, welche der so benannten Zellen in den Zylindern übereinander liegen und welche Zellen mit welchen benachbart sind.

Definitionen:
Ein Polynom (in r Variablen) A heißt auf einer Menge $M \subseteq \mathbf{R}^r$ *vorzeicheninvariant* :$\Leftrightarrow$

für alle $(a_1,\ldots,a_r) \in M$: $A(a_1,\ldots,a_r) > 0$ oder
für alle $(a_1,\ldots,a_r) \in M$: $A(a_1,\ldots,a_r) = 0$ oder
für alle $(a_1,\ldots,a_r) \in M$: $A(a_1,\ldots,a_r) < 0$.

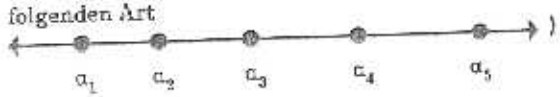
Eine Dekomposition D (Partition) von $\mathbf{R}^r$ ist eine *zylindrische Dekomposition* von $\mathbf{R}^r$:$\Leftrightarrow$ Entweder $r = 1$ und

D besteht aus

$(-\infty,a_1), \{a_1\}, (a_1,a_2), \{a_2\}, (a_2,a_3), \ldots, (a_{k-1},a_k), \{a_k\}, (a_k,\infty)$

für gewisse reelle Zahlen $a_1 < a_2 < \ldots < a_k$

(d.h. D besteht aus offenen Intervallen und einzelnen Punkten in der folgenden Art



oder r>1 und

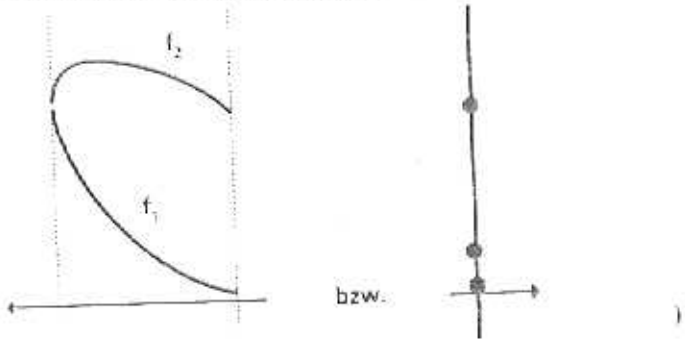
D ist die Vereinigung von "Stacks" S_i, die zu den "Zylindern" Z(D') einer zylindrischen Dekomposition $D'=(D'_1,...,D'_\ell)$ von R^{r-1} gehören $(i=1,...,\ell)$.

$(Z(M):=M\times R$ für $M\subseteq R^{r-1}$. Ein "Stack" S über einer "Zelle" $D'\subseteq R^{r-1}$ ist eine Dekomposition von Z(D') in die folgenden Mengen

$\{(x,y) \mid x\in D', f_k(x)<y\}$,
$\{(x,f_k(x)) \mid x\in D'\}$,
$\{(x,y) \mid x\in D', f_{k-1}(x)<y<f_k(x)\}$,
$\{(x,f_{k-1}(x)) \mid x\in D'\}$,
$\{(x,y) \mid x\in D', f_{k-2}(x)<y<f_{k-1}(x)\}$,
...
$\{(x,y) \mid x\in D', f_1(x)<y<f_2(x)\}$,
$\{(x,f_1(x)) \mid x\in D'\}$,
$\{(x,y) \mid x\in D', y<f_1(x)\}$.

für gewisse reelle, stetige, auf D' definierte Funktionen $f_1<f_2<...<f_k$, die "Sektionen" des Stacks.

Man gehe die Definition in der Zeichnung durch für
D' = das zweite Intervall von links auf der x-Achse und für
D' = der zweite Einzelpunkt. Es ergeben sich anschaulich die Stacks



Sei A eine endliche Menge von Polynomen in r Variablen mit rationalen Koeffizienten und sei PROJ(A) die folgende Menge von Polynomen in (r-1) Variablen:

PROJ(A) := $\bigcup_{F\in A} \bigcup_{G\in RED(F)} (\{ldcf(F)\} \cup PSC(G,G'))$
$\cup$
$\bigcup_{F1,F2\in A} \bigcup_{G1\in RED(F1),G2\in RED(F2)} PSC(G1,G2)$

(wobei

ldcf(F) ... der *höchste Koeffizient* des Polynoms F
(ldcf(F) ist ein Polynom in (r-1) Variablen!)

RED(F) ... die Menge der *Redukta* von F, das ist die Menge der Teilpolynome kleineren Grades in F

PSC(F,G) ... die Menge der *Haupt-Subresultanten-Koeffizienten* von F und G, das sind gewisse Unterdeterminanten der Sylvester-Determinante von F und G
(alle Elemente in PSC(F,G) sind Polynome in (r-1) Variablen!))

Eine *zylindrisch-algebraische Dekomposition* von R^r ist eine zylindrische Dekomposition, bei welcher alle in der Dekomposition vorkommenden Mengen M semialgebraisch sind.

Mathematisches Wissen (Collins 1973ff):
(Intuition: Wie man an obigem Beispiel sieht, sind die "wesentlichen" Stellen, bei denen man die Begrenzungen der zylindrischen Streifen errichten muß, die Stellen, wo das Nullstellengebilde Kreuzungspunkte, Spitzen, isolierte Punkte oder vertikale Tangenten hat. Es geht darum, eine algebraische Charakterisierung dieser wesentlichen Stellen im R^r zu finden und zu beweisen, daß man damit das Problem eine zylindrisch-algebraische Dekomposition in R^r zu finden, zurückführen kann auf dasselbe Problem im R^{r-1}. Ein zentrales Instrument dafür ist der folgende Satz.)

Sei nun R eine Zelle einer zylindrisch (algebraischen) Dekomposition von R^{r-1}, auf welcher alle Polynome von PROJ(A) vorzeicheninvariant sind. Dann sind alle Polynome F von A delineierbar (oder identisch 0) über R und für je zwei Polynome $F,G\in A$ gilt: Eine F-Sektion und eine G-Sektion in Z(R) sind entweder disjunkt oder identisch.

(F heißt *delineierbar* über R, wenn der Teil des Nullstellengebildes von F, der in Z(R) liegt, in $k\geq 0$ disjunkte Sektionen, die "F-Sektionen", zerfällt - und damit auf natürliche Weise einen Stack über R erzeugt.)

Dieser Satz gibt die Grundlage für folgenden Algorithmus zur Lösung des Problems der Konstruktion einer zylindrisch-algebraischen Dekomposition D des R^r, die für eine vorgegebene Menge A von Polynomen mit r Variablen vorzeicheninvariant ist:

Algorithmus (Collins 1973ff):

Falls r = 1:

Löse das Problem durch Isolierung der reellen Nullstellen der irreduziblen Faktoren der Polynome in A.

Falls r > 1:

Lose das Problem (durch rekursive Anwendung des Algorithmus) für die Menge PROJ(A) von Polynomen in (r-1) Variablen. Sei I' die dadurch erhaltene zylindrisch-algebraische Dekomposition von R^{r-1}, auf welcher alle Polynome von PROJ(A) vorzeicheninvariant sind.

Für jede Zelle R von I':

Errichte über R den durch A bestimmten Stack

(unter Verwendung des obigen Satzes, dessen Beweis konstruktiv ist).

Nimm diesen Stack in I auf. ∎

(Die sehr komplizierten Details des Algorithmus, die insbesondere ein subtiles Umgehen mit algebraischen Zahlen erfordern, gehen weit über den Rahmen dieser Übersichtsarbeit.)

Übersicht über die Literatur zur Computer-Algebra

Lehrbücher

Es gibt zwei Lehrbücher, die sich vor allem mit den Polynomalgorithmen beschäftigen:

KNUTH D.E., 1969: The Art of Computer Programming - Volume 2 / Seminumerical Algorithms. Addison-Wesley Publishing Company.

LIPSON J.D., 1981: Elements of Algebra and Algebraic Computing. Addison-Wesley Publishing Company.

Das folgende Buch versucht, einen Überblick über das Gesamtgebiet der Computer-Algebra zu geben (Algorithmen aus allen Teilgebieten der Computer-Algebra, Software-Systeme und Anwendungen) und gibt bis 1982 einen ziemlich vollständigen Hinweis auf die Literatur:

BUCHBERGER B., COLLINS G.E., LOOS R. (Hsg.), 1982: Computer Algebra - Symbolic and Algebraic Computation. Springer Verlag.

Ein wirkliches Lehrbuch über Computer-Algebra, das Überblick und Details des gesamten Gebietes bietet, steht jedoch derzeit noch aus. Ein Buch, das einen Eindruck von den mannigfaltigen Anwendungen gibt, ist:

PAVELLE R., 1985: Applications of Computer Algebra. Kluwer Academic Publishers.

Übersichtsartikel

Die folgenden Artikel geben einen Überblick über das gesamte Gebiet der Computer-Algebra:

CAVINESS B.F., 1985: Computer Algebra: Past and Future. Proc. EUROCAL 1985, Linz, Austria, Lecture Notes in Computer Science, vol. 203, S. 1-18, Springer Verlag.

WINKLER F., 1986: Computer Algebra. In R.A. Meyers (ed.): 'The Encyclopedia of Physical Science and Technology'. Academic Press, 1986, erscheint demnächst.

PAVELLE R., ROTHSTEIN M., FITCH J., 1982: Computer Algebra. Spektrum der Wissenschaft, Februar 1982, S. 71-78.

Die folgende Arbeit gibt vor allem einen Überblick über Computer-Algebra-Software-Systeme:

YUN D.Y.Y., STOUTEMYER R.D., 1980: Symbolic Mathematical Computation. In J. Belzer, A.G. Holzman, A. Kent (eds.), Encyclopedia of Computer Science and Technology, vol. 15, S. 235-310. Marcel Dekker.

Originalliteratur

Seit 1985 gibt es eine wissenschaftliche Zeitschrift, die sich ausschließlich mit Symbolic Computation, insbesondere mit Computer-Algebra befaßt:

Journal of Symbolic Computation (B. BUCHBERGER et al., Hsg.), Academic Press.

Den großten Teil des Umfanges des Journal of Symbolic Computation machen Originalarbeiten aus (mathematische Grundlagen neuer und verbesserter Algorithmen zur Computer-Algebra und anderen Bereichen des Symbolic Computation). In diesem Journal erscheinen aber auch laufend - in einer eigenen Sektion - Übersichtsartikel über Teilgebiete der Computer-Algebra und - in einer

anderen Sektion - kurze Berichte über erfolgreiche Anwendungen der Computer-Algebra in allen Ingenieursbereichen.

Bis zum Erscheinen des Journal of Symbolic Computation war die Originalliteratur zur Computer-Algebra verstreut in verschiedenensten Zeitschriften. Insbesondere aber findet man einen Niederschlag fast aller wichtigen Beiträge zur Computer-Algebra seit 1966 in den Konferenzberichten der internationalen Konferenzen, die seit 1966 von der SIGSAM-Gruppe (Special Interest Group in Symbolic and Algebraic Manipulation) von ACM (Association for Computing Machinery) und SAME (Symbolic and Algebraic Manipulation in Europe) veranstaltet wurden. Eine vollständige Liste aller Konferenzen mit den bibliographischen Angaben für die Konferenzberichte findet sich in /Buchberger,Collins,Loos 1982,S.4ff/.

Seit 1982 haben folgende große internationale Computer-Algebra-Konferenzen mit Proceedings stattgefunden:

EUROCAM 82, Marseille, Frankreich, April 1982.
Proceedings (J. Calmet, Hsg.): Lecture Notes in Computer Science, vol. 144, Springer Verlag.

EUROCAL 83: London, England, März 1983.
Proceedings (J.A. van Hulzen, Hsg.): Lecture Notes in Computer Science, vol. 162, Springer Verlag.

RIKEN 84: Wako-shi, Saitama, Japan, Juni 1984.
Proceedings (N. Inada, T. Soma, Hsg.): Series in Computer Science, vol. 2, World Scientific Publ. Comp.

EUROSAM 84: Cambridge, England, Juli 1984.
Proceedings (J. Fitch, Hsg.): Lecture Notes in Computer Science, vol. 174, Springer Verlag.

EUROCAL 85: Linz, Österreich, April 1985.
Proceedings Vol. 1 (Invited Lectures) (B. Buchberger, Hsg.): Lecture Notes in Computer Science, vol. 203, Springer Verlag.
Proceedings Vol. 2 (Research Contributions) (B.F. Caviness, Hsg.): Lecture Notes in Computer Science, vol. 204, Springer Verlag.

SYMSAC 86: Waterloo, Kanada, Juli 1986.

Eine Konferenz zum speziellen Thema der algorithmischen Gruppentheorie fand im August 1982 in Durham, England, statt. Die zugehörigen Proceedings sind

ATKINSON A.,1984: Computational Group Theory. Academic Press.

Zitierte Literatur

ARNON D.S., COLLINS G.E., MCCALLUM S., 1984: Cylindrical Algebraic Decomposition I: The Basic Algorithm. SIAM Journal of Computing, vol. 13, no. 4, S. 865-877.

BUCHBERGER B., 1965: Ein Algorithmus zum Auffinden der Basiselemente des Restklassenringes nach einem nulldimensionalen Polynomideal. Dissertation, Universität Innsbruck und Aequationes Mathematicae, vol. 4, fasc. 3, S. 374-383 (1970).

BUCHBERGER B., 1985: Gröbner Bases: An Algorithmic Method in Polynomial Ideal Theory. In N.K. Bose (Hsg.): 'Multidimensional Systems Theory', S. 184-232, D.Reidel Publishing Company.

CAVINESS B.F., 1967: On Canonical Forms and Simplifikation. Ph.D. Diss., Pittsburgh, Carnegie-Mellon University, 1967 and Journal of the ACM, vol. 17, no. 2, S. 385-396 (1970).

CAVINESS B.F., 1985: Computer Algebra: Past and Future. Proc. EUROCAL 85, Linz, Austria, Lecture Notes in Computer Science, vol. 203, S. 1-18, Springer Verlag.

CHAR B.W., FEE G.J., GEDDES K.O., GONNET G.H., MONAGAN M.B., 1986: A Tutorial Introduction to Maple. Journal of Symbolic Computation, vol. 2, no. 2.

COLLINS G.E., 1985: The SAC-2 Computer Algebra System. Proc. EUROCAL 85, Linz, Austria, Lecture Notes in Computer Science, vol. 204, S. 34-35, Springer Verlag.

DICKSON L.E., 1913: Finiteness of the Odd Perfect and Primitive Abundant Numbers with n Distinct Prime Factors. American Journal of Mathematics, vol. 35, S. 115-138.

PATEMAN R.J., 1981: Symbolic and Algebraic Computer Programming Systems. SIGSAM Bulletin, vol. 15, no. 1, S. 21-32.

GEBAUER R., KREDEL H., 1983: Buchberger Algorithm System. SIGSAM Bulletin, vol. 18, no. 1.

HIRONAKA H., 1964: Resolution of Singularities of an Algebraic Variety over a Field of Characteristic Zero: I,II. Annals of Math., vol. 79, S. 109-327.

FITCH J., 1985: Solving Algebraic Problems with REDUCE. Journal of Symbolic Computation, vol. 1, no. 2, S. 211-227.

JENKS R.D., 1984: A Primer - 11 Keys to New SCRATCHPAD. Proc. EUROSAM 84, Cambridge, England, Lecture Notes in Computer Science, vol. 174, S. 123-147, Springer Verlag.

KALTOFEN E., 1982: Factorization of Polynomials. In B. Buchberger, G.E. Collins, R. Loos (Hsg.): 'Computer Algebra - Symbolic and Algebraic Computation', Springer Verlag, S. 95-113.

KARATSUBA A., OFMAN Y., 1963: Multiplication of Multidigit Numbers on Automata. Soviet Phys. Dokl. 7, S. 595-596.

KNUTH D.E., 1969: The Art of Computer Programming - Volume 2 / Seminumerical Algorithms. Addison-Wesley Publishing Company.

KUTZLER B., STIFTER S., 1986: Automated Geometry Theorem Proving using Buchberger's Algorithm. Proc. SYMSAC'86, Waterloo, Kanada, erscheint demnächst.

LICHTENBERGER F., 1981: REDUCE - Ein Beispiel eines Software-Systems für symbolisches und algebraisches Rechnen. Universität Linz, Institut für Mathematik, CAMP-Publ. 81-11.0.

LOOS R., 1982: Computing in Algebraic Extensions. In B. Buchberger, G.E. Collins, R. Loos (Hsg.): 'Computer Algebra - Symbolic and Algebraic Computation', Springer Verlag, S. 173-188.

MAYR E.W., MEYER A.R., 1981: The Complexity of the Word Problems for Commutative Semigroups and Polynomial Ideals. Report LCS/TM-199, MIT Laboratory of Computer Science.

PAVELLE R., 1985 Applications of Computer Algebra. Kluwer Academic Publishers.

PAVELLE R., WANG P.S., 1985: MACSYMA from F to G. Journal of Symbolic Computation, vol. 1, no. 1, S. 69-100.

RAND R.H., 1984 Computer Algebra in Applied Mathematics: An Introduction to MACSYMA. Research Notes in Mathematics, vol. 94, Pitman Publishing Inc

RISCH R.H., 1970: The Solution of the Problem of Integration in Finite Terms. Bulletin AMS 76, S. 605-608

SCHWARTZ J.T., SHARIR M., 1983: On the 'Piano Movers' Problem - II. General Techniques for Computing Topological Properties of Real Algebraic Manifolds. Advances in Applied Mathematics, vol. 4, S. 298-351

STOUTEMYER D.R., 1985: A Preview of the Next IBM-PC Version of muMATH. Proc. EUROCAL 85, Linz, Austria, Lecture Notes in Computer Science, vol. 203, S. 33-44

SUTOR R.S., 1985 The Scratchpad II Computer Algebra Language and System Proc. EUROCAL 85, Linz, Austria, Lecture Notes in Computer Science, vol. 204, S. 32-33.

TARSKI A., 1948 A Decision Method for Elementary Algebra and Geometry. Univ. of Calif. Press.

TRINKS W., 1978 On B. Buchberger's Method for Solving Systems of Algebraic Equations. J. Number Theory, vol. 10, no. 4, S. 475-488.

VAN HULZEN J.A., CALMET J., 1982: Computer Algebra Systems. In B. Buchberger, G.E. Collins, R. Loos (Hrg.): 'Computer Algebra - Symbolic and Algebraic Computation', Springer Verlag, S. 221-243

ZASSENHAUS H., 1969 On Hensel Factorization. International Journal of Number Theory, vol. 1, S. 291-311

Ausbildung

An vielen Computer Science Departments in den USA und einigen Informatik-Instituten in Deutschland gibt es die Möglichkeit, sich in Computer-Algebra zu spezialisieren. An der Universität Linz wird für den gesamten Bereich des Symbolic Computation ein systematischer Studienschwerpunkt (ca. 30 Einzelkurse) angeboten, der sowohl für Studenten der Informatik als auch für Studenten der Mathematik offen steht. Eine Detailbeschreibung dieses Studienschwerpunktes kann beim ersten Autor dieses Beitrages bezogen werden.

Unser besonderer Dank gilt Prof. D.R. Stoutemyer für das Überlassen einer Kopie von muMATH-83, sowie IBM Wien und IBM Yorktown Heights für die Möglichkeit eines Besuches des Forschungsinstitutes in Yorktown Heights, um dort SCRATCHPAD zu benützen. Die Arbeit an diesem Manuskript erfolgte im Rahmen eines von SIEMENS München geförderten Forschungsprojektes. Das Manuskript wurde mit dem Arbeitsplatzsystem 5815 der Firma SIEMENS erstellt.

ALGORITHMEN ZUR METHODE DER FINITEN ELEMENTE FÜR VEKTORRECHNER

Matthias Kratz
(Rechenzentrum, Technische Universität Braunschweig)

"Now the engineer has to watch both the clock and the cash box. He is not able to try every possible combination of design parameters, but must strike an economic balance between the information he would like and the time and cost of getting it. In practice, the ease with which a final design can be modified will determine how closely the final product approaches the 'best possible' design. The automatic computer, by making calculations cheap and easy, enables the hard-pressed engineer to extend the range and depth of his design studies; not only can he examine a greater number of alternative designs, he can examine each of them much more thoroughly."

S.H. HOLLINGDALE, High Speed Computing (1959)