

## Computeralgebrasysteme: Stand der Technik und Perspektiven

Prof. Dr. Benno Fuchssteiner, Universität-GH, Paderborn

### 1 Basisthese

Gemessen an dem was ein durchschnittlicher Anwender in der täglichen Praxis an Mathematik einsetzt, ist CA (Computeralgebra) sehr **effizient**: Wir können leicht praktische Fragen widerspiegelnde 250-seitige Formeln analysieren und umformen, wir können sie automatisch generieren, und brauchen sie in ihrer abscheulichen Erhabenheit nicht einmal zu sehen, um sie auszuwerten: mathematische Objekte werden von einem Interface zum anderen weitergereicht. Diese Arbeitsweise wird den zukünftigen Umgang mit Mathematik bestimmen.

Gemessen an dem was möglich ist, und gemessen an dem was mathematisch anspruchsvolle naturwissenschaftlich-technische Fragen bei realitätsnaher (meist nichtlinearer) Beschreibung eigentlich erfordern, sind Computeralgebrasysteme (CAS) allerdings **ineffizient**, ja sie spiegeln gewissermaßen die Steinzeit mathematischer Arbeitsumgebungen des Computerzeitalters wieder. Die Ausdrucksfähigkeit gegenwärtiger Systeme für subtile mathematisch-technische Sachverhalte ist unterentwickelt.

Diese widersprüchliche These wollen wir erläutern.

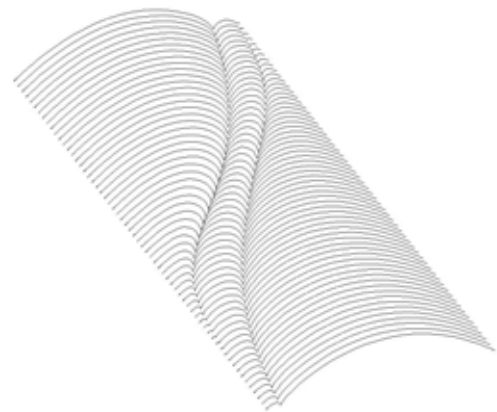
### 2 Computeralgebrasysteme heute

#### 2.1 Was können CAS heute?

Als Beispiel betrachten wir die geplottete Lösung der folgenden nichtlinearen partiellen Differentialgleichung (der Harry Dym Gleichung)

$$u_t = u^3 u_{xxx}$$

Mit numerischen Verfahren kann diese Lösung durch Vorgabe des Anfangswertes nicht gefunden werden, da Störungen des Anfangswertes sich mindestens exponentiell fortpflanzen. Es bleibt also nur die Lösung durch explizite Verfahren übrig.



Dies führt dazu, daß man eine mehr als 250-seitige Formel [2] verarbeiten und numerisch auswerten muß: Ein Kinderspiel für jedes vernünftige Computeralgebrasystem. Computeralgebra trägt bei solchen Beispielen zu einer erheblichen Verbreiterung unserer mathematischen Verfahren zur Lösung technisch-wissenschaftlicher Probleme bei.

Die Stärke von Computeralgebra wird sehr schön anhand eines historischen Beispiels belegt: Charles Eugène Delaunay<sup>1</sup> arbeitete mehr als zwanzig Jahre an den Formeln seiner zweibändigen unvollendeten Theorie der

<sup>1</sup>Charles Eugène Delaunay, geb. am 9.4.1814 in Lussigny, ertrunken am 5.8.1872 bei Cherbourg, Lehrer an der Sorbonne und an der École Polytechnique, Bureau des Longitudes (ab 1866), Leiter der Pariser Sternwarte (ab 1870), 1846-1867 Arbeit an *Théorie de la Lune*, publiziert 1860-67 (unvollendet). Siehe auch: R. Pavelle, M. Rothstein und J. Fitch: Computer algebra, Scientific American, 245, 102-113, December 1981.

Mondbewegung. Die Überprüfung der Formeln alleine nahm die letzten 10 Jahre im Leben dieses kreativen Geistes in Anspruch. 1970 wurden die Ergebnisse – wohlgerne die Formeln, nicht das Rechnen mit Zahlen – erneut überprüft: mit einem Computeralgebrasystem innerhalb von 20 Stunden. Heute würde diese Prüfung, bei der auf Seite 234 ein Fehler gefunden wurde, weniger als zwei Stunden dauern. Was hätte Delauny mit den geschenkten 10 Jahren nicht alles anfangen können?

Die Ausdrucksfähigkeit für mathematische Sachverhalte, die Computeralgebrasysteme offerieren, ist groß: In der Hochsprache eines geeigneten Computeralgebrasystems schreibt man einen neuen Differenzierer, der die Differentiationsfunktion des Systems nicht nutzt, und welcher zum Beispiel die durch

$$\int_1^{y(x)+x} \sin(\ln(x\xi))d\xi + y(x)^2 = 0.1$$

definierte Funktion mühelos in Millisekunden differenziert, auf etwa einer DIN A4 Seite. Man schreibt diesen Differenzierer schneller als man einem Studenten, der die Differentialrechnung schon beherrscht, beibringt, welche neue Regeln (parameterabhängige Integrale, implizite Funktionen) er anwenden muß, um diesen Ausdruck abzuleiten.

Computeralgebra, sofern sie die Definition eigener Datentypen erlaubt, ist ein hervorragendes Werkzeug nicht nur zur Strukturierung mathematischer Sachverhalte sondern auch zur effizienten Verwaltung nichtmathematischer Daten aus Wissenschaft und Technik.

### 2.1.1 Expertenwissen von CAS

Computeralgebrasysteme offerieren in ihren Funktionsbibliotheken einen reichhaltigen Vorrat an mathematischem Expertenwissen. Zur Illustration greifen wir hier einige der Beispiele aus Westers Testsuite [10] heraus. In diesem Test werden 131 Beispiele zur Messung

der Funktionalität von Computeralgebrasystemen vorgelegt. Die angegebenen Beispiele wurden mit MuPAD 1.3 [8], [3], gelöst, die Aufstellung wurde von Paul Zimmermann [11], [12] angegeben. Die Beispiele geben einen guten Einblick in die formalen Fähigkeiten moderner Computeralgebrasysteme. Dabei bezeichne `>>` die Eingabe an das System und `output>` die Antwort des Systems.

*Problem 9:*

Man vereinfache

$$\sqrt{14 + 3\sqrt{3 + 2\sqrt{5 - 12\sqrt{3 - 2\sqrt{2}}}}}$$

```
>> radsimp(sqrt(14 + 3*sqrt(3 +
2*sqrt(5 - 12*sqrt(3 - 2*sqrt(2))))));
```

output>

$$\frac{1}{2} + 3$$

*Problem 34:*

Man vereinfache

$$(2\sqrt{r} + 1)^{\frac{1}{2\sqrt{r}+1}} (4r + 4\sqrt{r} + 1)^{\sqrt{r}\frac{1}{2\sqrt{r}+1}} - 2\sqrt{r}.$$

```
>> e:=
(4*r+4*sqrt(r)+1)^(sqrt(r)/(2*sqrt(r)+1))
*(2*sqrt(r)+1)^(1/(2*sqrt(r)+1))-2*sqrt(r);
```

```
>> simplify(e);
```

output>

$$1$$

*Problem 45:*

Man berechne den Wert von  $\tan^{-1}(\tan 4)$ , nämlich  $4 - \pi$ .

```
>> atan(tan(4));
```

output>

$$-\pi + 4$$

*Problem 58:*

Man löse die Gleichung  $\sqrt{\log x} = \log \sqrt{x}$ .

```
>> solve(sqrt(ln(x))=ln(sqrt(x)),x);
```

output>

{1, exp(4)}

Da die Antwort ohne Annahmen über  $x$  gegeben wurde, sind 1 und  $e^4$  die einzigen Lösungen in der komplexen Ebene.

*Problem 68:*

Man gebe  $\sum_{k=1}^{\infty} (1/k^2 + 1/k^3)$  in geschlossener Form an.

```
>> sum(1/k^2+1/k^3,k=1..infinity);
```

```
output>
```

```
          2
         PI
zeta(3) + ----
```

*Problem 78:*

Man berechne die Ableitung der stückweise definierten Funktion  $|x|$ .

```
>> a:=proc(x)
begin
  if x < 0 then -x else x end_if
end_proc;
```

```
>> D(a);
```

```
output>
```

```
proc(x)
begin
  if x < 0 then
    -1
  else
    1
  end_if
end_proc
```

*Problem 88:*

Man berechne das Integral  $\int_0^{\infty} t^{a-1}/(1+t)dt$  for  $0 \leq \text{Re}(a) \leq 1$ .

```
>> assume(Re(a)>0): assume(Re(a)<1):
```

```
>> int(t^(a-1)/(1+t),t=0..infinity);
```

```
output>
```

```
gamma(a) gamma(- a + 1)
```

*Problem 102:*

Man berechne die Laplacetransformierte von  $\cos((\omega - 1)t)$  bezüglich  $t$ .

```
>> laplace(cos((w-1)*t),t,s);
```

```
output>
```

```
          s
-----
      2      2
s  + (w - 1)
```

*Problem 103:*

Man löse die Gleichung

$$y'' + yy'^3 = 0 \text{ for } y(x).$$

```
>> solve(ode(
    diff(y(x),x,x)+y(x)*diff(y(x),x)^3=0,y(x)
));
```

output>

```

                                     3
{C12,RootOf(6 x + 6 C11 + 6 y C10 - y , y)}
```

Das Ergebnis besagt, daß die Lösung entweder  $y(x) = C12$ , oder die algebraische Funktion ist, die sich durch  $y(x)^3 = 6C10y(x) + 6x + 6C11$  ergibt, dabei sind  $C10$ ,  $C11$  and  $C12$  beliebige Konstante.

## 2.2 Was können CAS'e nicht?

Gemessen an dem was mathematisch anspruchsvolle naturwissenschaftlich-technische Fragen erfordern, sind Computeralgebrasysteme allerdings *ineffizient*. Die Ausdrucksfähigkeit gegenwärtiger Systeme für wirklich subtile mathematisch-technische Sachverhalte ist noch unterentwickelt.

Dazu ein Beispiel:

Will man, ähnlich wie oben, relevante Lösungen der Kawamoto-Gleichung

$$u_t = 10u^4 u_{xx} u_{xxx} + 5u^4 u_x u_{xxxx} + u^5 u_{xxxxx}$$

finden, so muß man zuerst eine in einer Zeile darstellbare Identität eines abgeleiteten zweizeiligen Ausdrucks überprüfen, dessen Berechnung aus rein algebraischer Routinearbeit besteht. Die Expansion dieses Ausdrucks führt dann auf eine Formel, welche sich als Textdokument auf etwa 50 Gigabyte beläuft, denn der in der Expansion entstehende *Intermediate Data Swell* führt dazu, daß die zu überprüfende Identität ein Integro-Differentialoperator von ungefähr vier Milliarden Termen wird. Der Ausdruck dieses Operators, etwa um ihn einem Diplomanden zur Handberechnung anzuempfehlen, ergibt einen Stoß DIN A 4-Blätter von

etwa 18 km Höhe, oder einen Blätterberg von 1000 Kubikmetern: Eine schiere Unmöglichkeit für heutige Computeralgebrasysteme.

Noch dramatischer wird es, wenn man ernsthaft Mathematik, aufbauend auf einer rigorosen axiomatischen Begründung des gesamten Fundaments dieser Wissenschaft, betreiben will. Dann würde man etwa den formal axiomatischen Aufbau, der in [1] dargelegt ist, wählen und damit beginnen die natürlichen Zahlen aus Mengen in der üblichen Ordinalzahldarstellung zu konstruieren. Expandiert man dann einen Ausdruck, der die Zahl 1 in einfacher Weise aus der Null konstruiert, und ersetzt alle bis dahin gewählten Abkürzungen, so beginnt das Ergebnis mit

$$\tau_{x35} \left( \neg (\exists x33) \left( \neg \neg \vee \neg \vee \neg \in x33 \ x35 \neg \right. \right. \\ \left. \left. (\exists x34) \left( \neg \vee \neg \in x34 \ \tau_{x32} \left( \neg (\exists x16) \left( \neg \neg \vee \right. \right. \right. \right. \right. \\ \neg \vee \neg \in x16 \ x32 \neg \vee \neg \in x16 \ \tau_{x19} \left( \neg (\exists x17) \left( \right. \right. \right. \\ \neg \neg \vee \neg \vee \neg \in x17 \ x19 \neg (\exists x18) \left( \neg \vee \neg \in \right. \right. \\ \left. \left. x18 \ x17 \in x18 \ \tau_{x3} \left( \neg \vee \neg \in \tau_{x1} \left( \neg (\exists x2) \left( \right. \right. \right. \right. \right. \\ \neg \neg \in x2 \ x1 \right) \right) \right) \right) \right) \ x3 \neg \neg (\exists x4) \left( \neg \vee \neg \in x4 \right. \\ \left. x3 \in \tau_{x15} \left( \neg (\exists x13) \left( \neg \neg \vee \neg \vee \neg \in x13 \ x15 \right. \right. \right. \\ \left. \left. (\exists x13) \left( \neg \vee \neg \in x14 \ \tau_{x12} \left( \neg (\exists x9) \left( \neg \neg \vee \right. \right. \right. \right. \right. \\ \neg \vee \neg \in x9 \ x12 \vee \neg (\exists x10) \left( \neg \neg \vee \neg \vee \neg \in \right. \right. \\ \left. \left. x10 \ x9 \in x10 \ x4 \neg \vee \neg \in x10 \ x4 \in x10 \ x9 \right) \right) \neg \right. \\ \left. \text{und erstreckt sich über weitere 17 DIN A4 Seiten.} \right.$$

Bei beiden Beispielen hindert nicht nur die Datenflut eine adequate Behandlung dieser Probleme durch CAS'e sondern es wirkt sich - neben anderen Nachteilen - auch die Langsamkeit der Systeme störend aus.

Computeralgebra ist deshalb noch weit davon entfernt wirkliche Mathematik zu machen, geschweige denn das wahre *Wunder der Beherrschung der Mathematik durch den Menschen* überflüssig zu machen, dieses ist ein ganz anderes Phänomen als das Hantieren mit großen algebraischen Daten:

Daß man nämlich in einem eigentlich durch

sterile Strenge beschriebenem Bereich Intuition entwickeln kann, so daß formale Monster in kurzen Begriffen zusammenzufassen sind, daß man in diesem Bereich kreative neue Ideen haben und ausdrücken kann, daß man diese dann als vereinfachtes Abbild von Aspekten der Wirklichkeit sinnvoll einsetzen kann, und daß man solche formal-sterilen Schlußweisen, die nun wirklich einer Maschine mehr Spaß machen sollten als dem Menschen, dem menschlichen Geist gemäß aufbereiten kann. Dieses Wunder der Mathematik, dessen Mathematiker sich zu selten bewußt sind, und das vielleicht bei Ingenieuren und Physikern manchmal besser aufgehoben sein mag, kann niemals durch Maschinen vollzogen werden. Hier liegen die Grenzen, die auch Computeralgebra nicht überwinden wird, Grenzen, die Computeralgebra aber weit, sehr weit verschieben kann. Daß man aber dem menschlichen Geist, sofern wir in diesem Bereich zu immer neueren und formal komplizierteren Sachverhalten aufbrechen, den Gefallen tun sollte, das was maschinell möglich ist, von seinen Schultern zu nehmen, steht außer Frage. Es steht auch außer Frage, daß man durch den Einsatz solcher Hilfsmittel völlig neue Dimensionen für Anwendungen in Technik und Naturwissenschaften eröffnen kann. Dafür braucht man leistungsfähige, bequeme und nutzerfreundliche Computeralgebrasysteme, oder mehr noch: neue komfortable, integrierte und multimediale Arbeitsumgebungen, für die heutige Computeralgebrasysteme nur eine Vorstufe sind.

### 3 Stand der Technik

#### 3.1 Basisfunktionalität

Hohe Anforderungen bei der Nutzung von Computeralgebrasystemen erfordern eine gute Kenntnis der Interna (die von den Herstellern und Entwicklern der Systeme häufig nicht offengelegt werden).

Um zu zeigen weshalb Computer überhaupt symbolische Mathematik, wie Differentiation, Integration, Grenzwertberechnung und ähnliches betreiben können, führen wir kurz in die Baumstruktur mathematischer Ausdrücke ein, und diskutieren Basisfunktionalitäten wie Evaluation, Vereinfachung, Parameterübergabe und Scoping. Eine Problematik der Basisfunktionalität liegt in einer nahezu willkürlichen Funktionsweise, die beim Entwurf gewählt wird, und welche grundlegende Routinen eines Computeralgebrasystems beeinflusst, und die verschiedenen Systeme nahezu unvergleichbar macht.

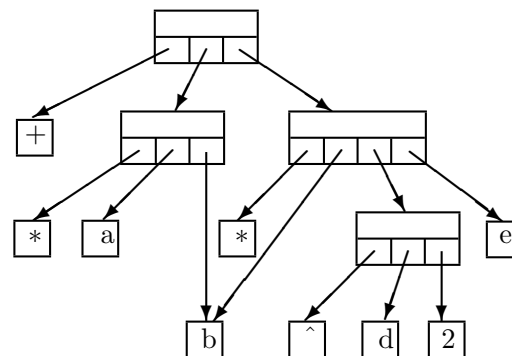
Ein Computeralgebrasystem ist eine Kombination von Basisfunktionen mit Werkzeugen der effizienten Datenmanipulation und mit elementaren logischen Konstrukten. Damit ergibt sich, geschickt implementiert und mit Hilfsroutinen versehen, eine außerordentlich effiziente High-Level-Sprache, die vermutlich viel mathematiknäher ist, als unsere übliche mathematische Schreibweise.

#### 3.1.1 Baumstruktur

Betrachten wir einmal einen einfachen mathematischen Ausdruck :

$$a * b + b * d ^ 2 * e.$$

Dieser besteht aus Größen, die mit Operationen verknüpft werden. Die geeignete Struktur so etwas darzustellen ist ein Ausdrucksbaum:



Vereinfacht gesprochen besteht die Funktion eines Computeralgebrasystems nun darin, sol-

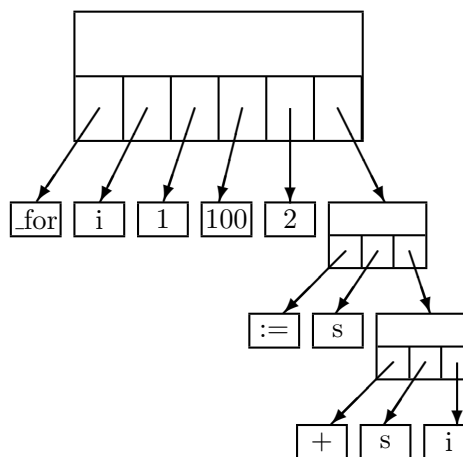
che Bäume nach gewissen Regeln im Interpretermodus abzuarbeiten. Entscheidend für die Effizienz ist deshalb wie gut die Verwaltung solcher Bäume implementiert ist, und wie dateneffizient solche Bäume bei großer Anzahl von Blättern gehalten werden. Ein Problem, welches dabei auftritt sieht man schon am obigen Beispiel: das der eindeutigen Datenrepräsentation (unique data representation). Denn bei der Realisierung des Systems muß man sich dafür entscheiden ob daß Datum  $b$ , welches zweimal vorkommt, nur einmal physikalisch realisiert wird (und beim zweiten Vorkommen durch einem Zeiger ersetzt wird), oder ob es zweimal physikalisch abgelegt wird. Die erste Methode setzt einen aufwendigen Verwaltungsapparat voraus, geht aber sparsamer mit Speicherplatz um, ist manchmal auch zeiteffizienter. Bei einem einzelnen  $b$  scheint diese Frage keine Rolle zu spielen, aber  $b$  selbst könnte auch der Repräsentant eines Datums von vielen Megabyte sein, was dann sehr wohl ins Gewicht fallen würde. Man kann auch einen Mittelweg, vielleicht basierend auf heuristischen Prinzipien, einschlagen. Alle diese Methoden haben Vor- und Nachteile, was dazu führt, daß die Methode, die bei großen Daten geeignet ist, bei kleinen Daten sich als schwerfällig erweist. Allgemeine Aussagen über die Performance von Computeralgebrasystemen sind deshalb immer von geringer Aussagekraft.

Etwas anderes wird durch die gedankliche Reduzierung eines Computeralgebrasystem auf ein Baummanipulationsprogramm klar, nämlich, das das Programm eigentlich gar nichts mit Mathematik zu tun hat, und daß an Stelle mathematischer Daten auch andere technisch-wissenschaftliche Objekte treten können. Natürlich sollte man, wenn solches geplant ist, die Möglichkeit zur Definition beliebiger Datentypen haben: das System sollte ein objektorientiertes System sein<sup>2</sup>

<sup>2</sup>Gegenwärtig gibt es weltweit zwei objektorientierte

Die Implementation einer hochentwickelten Programmiersprache setzt auch nichts anderes voraus als die konsequente Verwirklichung des eben erwähnten Gesichtspunktes. Denn Programme und logische Konstrukte sind auch nichts anderes als Bäume, deren Transformationen gewissen Regeln unterliegen, hier das Beispiel einer for-Schleife:

```
for i from 3 to 100 step 2 do s:=s+i
end_for
```



Manche Systeme, wie zum Beispiel MuPAD, erlauben dem Nutzer den Zugriff auch auf die Elemente der Datenbäume, die durch Programme und logische Konstrukte gegeben sind. Das erlaubt mächtige aber wegen möglicher Seiteneffekte nicht unproblematische Werkzeuge der Programmanipulation.

### 3.1.2 Evaluation und Vereinfachung

Das Problem der Vereinfachung läßt sich kurz abhandeln: es geht dabei darum die Bäume, die mathematische Ausdrücke darstellen, auf möglichst einfache Form zu bringen. Das ist allerdings leichter gesagt als getan, da es keine allgemein akzeptierte Übereinkunft darüber gibt, was ein einfacher ma-

Systeme: Axiom und MuPAD.

thematischer Ausdruck ist. Hier herrscht beim Entwurf von Computeralgebrasystemen deshalb notgedrungen die reine Willkür. Wichtig bei der Implementierung eines Vereinfachers (Simplifiers) ist allerdings, daß er immer da wo es möglich ist, eine Normalform herstellt damit man mathematische Ausdrücke vergleichen kann. Auch dies ist wieder nicht so einfach, den nicht für jede Klasse von Ausdrücken existiert ein Normalformenalgorithmus. Während man bei frühen Systemen dazu tendierte möglichst viel Funktionalität in den Vereinfacher zu stecken, ist man heute dabei vorsichtiger. Man vermeidet damit Performanceverluste, und überträgt dem Nutzer eine größere Verantwortung für die Umformung seiner Objekte.

Der Evaluierer, welcher die entscheidende Rolle im Arbeitszyklus *Parsen - Evaluieren - Ausgeben* spielt, ist das Herz eines Computeralgebrasystems. Er regelt wie die Bäume des Systems transformiert werden. Ihm obliegen die Aufgaben

- Substitution von Bezeichnern
- Ausführung von Systemfunktionen
- Ausführung von Systemoperatoren
- Vereinfachung von Ausdrücken
- Auswertung aller Datenstrukturen
- Ausführung von Anweisungen
- Ausführung benutzerdefinierter Prozeduren
- Behandlung von Domains und Überladungen

Die letztgenannte Aufgabe wird natürlich nur bei objektorientierten Systemen wahrgenommen. Daneben ist noch die Typüberprüfung, die *Unique Data Representation*, die Debugger-Steuerung (MuPAD) und eine Reihe anderer wichtiger Aufgaben zu nennen, die wir hier der Kürze halber weggelassen haben. Die Qualität des Entwurfs des Evaluierers ist für die Performance des Systems entscheidend. Hier steckt der Teil des know-hows eines Systems, der nicht in der Literatur zu finden ist. Die Komplexität des Evaluierers sieht man schon am einfachen Beispiel der *Unique Data*

*Representation*. Führt man diese konsequent aus, muß jedes neu erzeugte Datum mit allen vorhandenen Daten verglichen werden. Wenn das nicht schnell und zuverlässig geschieht, dann geht das System in die Knie und erstickt in seinem eigenen Datenmüll. Allgemeine Grundsätze des Entwurfs dieses für die Funktionalität von Systemen entscheidenden Bausteins gibt es nicht. Kaum ein System erlaubt da einen Einblick (Ausnahme Reduce). Gerade deshalb erscheint es dem Autor als Frage der wissenschaftlichen Integrität, daß weitverbreitete Systeme dem ernsthaft interessierten Anwender einen Einblick in den Source Code des Systems erlauben.

### 3.1.3 Scoping

Die für den Anfänger fehlerträchtigste Komponente, welche häufig den Anlaß für weitgehendes Unverständnis liefert, ist die des Scoping. Scoping regelt die Art und Weise wie und in welchem Kontext Prozeduren auf ihre Argumente zugreifen. Mögliche Seiteneffekte führen dazu, daß man die Effekte des Scoping erst nach langem Gebrauch eines Systems übersieht. Man unterscheidet:

*Statisches Scoping*: hier greifen Prozeduren  $h(\text{argumente})$ , die in Prozeduren  $f(\text{argumente})$  aufgerufen werden, mit ihren globalen Variablen auf den externen Kontext zu.

*Dynamische Scoping*: bei diesem greift dasselbe  $h(\text{argumente})$  auf den von  $f(\text{argumente})$  geschaffenen Kontext zu.

So einfach das klingt, so führt es doch zu zahllosen Verwirrungen, weil die damit festgelegte Funktionsweise vielfach der Intuition widerspricht, welche sich im Laufe der Zeit des Umgangs mit Mathematik ausgebildet hat.<sup>3</sup>

<sup>3</sup>MuPAD ist eines der wenigen Systeme, welche das dynamische Scoping verwenden, die Meinungen ob dies ein Vor- oder Nachteil ist sind geteilt. Der Autor dieser Zeilen zieht das dynamische Scoping vor, ist sich aber

Beim Aufruf der Funktionsvariablen selbst spielt es eine wichtige Rolle ob diese entweder einen *call bei value*, einen *call by name* oder einen *call by evaluated name* ausführen. Beim *call by value*, etwa in MuPAD, wird das in der Prozedur vorkommende Argument immer mit seinem momentanen Wert genommen, während beim *call by evaluated name* das Argument mit dem Wert genommen wird, den es am Anfang des Aufrufs der Prozedur hat. Maple arbeitet mit *call by evaluated name*. Beides hat wieder Vor- und Nachteile: die zweite Methode ist einfacher, hat weniger Seiteneffekte, aber deshalb auch weniger Gestaltungsmöglichkeiten. Außerdem kann man den *call by evaluated name* ohne die Implementation eines Stacks realisieren, was verführerisch und trügerisch ist. Verführerisch deshalb, weil es zu großen Geschwindigkeitsvorteilen führt, trügerisch deshalb, weil dies zu Ärger mit lokalen Prozeduren führt und die Verwirklichung eines später auflösbaren holds sehr schwierig macht.<sup>4</sup>

Beim *call by name* ist es ähnlich wie beim *call by evaluated name*, nur wird beim Aufruf  $f(x)$ , ein in der Prozedurdefinition vorkommender Parameter  $a$ , der auf  $x$  verweist, durch den Bezeichner von  $x$  ersetzt, und nicht durch den Wert von  $x$ .

Da diese entscheidenden Basisfunktionalitäten in verschiedenen Systemen verschieden realisiert sind, ist eine eins-zu-eins Übertragung von Routinen eines Systems auf ein anderes nicht möglich: Wir werden also noch lange Zeit mit verschiedenen Computeralgebrasystemen zu leben haben, was aber wegen der dadurch gegebenen Konkurrenz wohl als Vorteil anzusehen ist.

---

fast sicher, daß sich wegen der Unkompliziertheit auf die Dauer das statische Scoping durchsetzen wird. Deshalb werden wir in den nächsten Jahren sicher erleben, daß MuPAD eine nutzergesteuerte Auswahl zwischen beiden Methoden erlaubt.

<sup>4</sup>In dieser Tatsache scheint mir der Grund für die günstige Geschwindigkeit von Maple zu liegen, eine Geschwindigkeit, die m. M. nach aber teuer erkauft ist.

## 3.2 Aufbau eines Systems

Um kommende Entwicklungen in ihrer Bedeutung abschätzen zu können, ist es wichtig den grundlegenden Aufbau eines Computeralgebrasystems zu kennen. Ich beziehe mich in der folgenden Beschreibung auf MuPAD, das einzige System, bei dem ich (neben Reduce) den Sourcecode kenne.

MuPAD ist in vier hierarchisch aufeinander aufbauenden Ebenen realisiert.

Die oberste Ebene ist durch die MuPAD *Funktionsbibliothek* gegeben. Darin sind Programme, die in der MuPAD-Hochsprache entweder vom Nutzer selbst geschrieben wurden oder vom System bereits zur Verfügung gestellt werden.

Die Funktionsbibliothek setzt auf dem *MuPAD-Kern* auf. Der MuPAD-Kern enthält die Implementierung der MuPAD-Hochsprache sowie die für ein Computeralgebra-System grundlegenden Mechanismen wie Substitution, Vereinfachung (Bildung einer Normalform) und Evaluierung, weiterhin eine Reihe von Systemfunktionen (Kernfunktionen). Systemfunktionen sind grundlegende Funktionen, die nur auf dieser Ebene zu realisieren sind oder die sehr häufig benutzt werden, und daher aus Effizienzgründen statt in der Library im Kern untergebracht wurden. Solche Funktionen sind erheblich schneller als Library-Funktionen, da diese interpretiert werden, während Systemfunktionen direkt ausführbare Binär-codeobjekte sind, die viele der ausschließlich im Kern zugänglichen Methoden direkt nutzen können.

Unter dem MuPAD-Kern liegt MAMMUT (Memory Allocation Management UniT). MAMMUT erfüllt in erster Linie die Aufgabe einer Speicherverwaltung, weist aber darüber hinausgehend eine große Anzahl vom Kern benötigter Fähigkeiten auf. Hierbei ist besonders die Realisierung der *Unique-Data-Representation* zu nennen. Diese wird

durch die Verwendung von Referenzzählern in optimaler Weise durch MAMMUT unterstützt. Die Verschiebbarkeit der von MAMMUT zur Verfügung gestellten Objekte dient der Vermeidung von Fragmentierung innerhalb des Speichers und macht Annahmen über Objektgrößen (Maple?), die wegen der großen Bandbreite von Aufgaben der Computeralgebra unzulässig sind, unnötig. Auch eine durch disziplinierten Umgang des Kernes mit den Speicherobjekten nur in wenigen Fällen benötigte *garbage collection* wird bereitgestellt. Ein weiterer Vorteil der eigenen Speicherverwaltung ist die bessere Portierbarkeit des Systems.

Die Unterstützung der *Unique-Data-Representation* besteht, wie schon gesagt, darin, daß Objekte, die logisch mehrfach benötigt werden, physikalisch nur einmal vorhanden sind und die logischen Kopien durch Zeiger auf das entsprechende physikalische Objekt realisiert werden. Die dahinter liegende Funktionalität ist kompliziert, da bei lokalen Änderungen in (algebraischen Ausdrücken entsprechenden) Teilbäumen jeweils entschieden werden muß, welche Teilbäume, Blätter und Äste von dieser Änderung so berührt werden, daß aus der logischen Kopie zumindest teilweise eine physikalische Kopie werden muß. Durch eine recht rigorose Unique-Data-Representation trägt MAMMUT allerdings wesentlich zur Speicherplatzökonomie bei.

Damit MuPAD eine parallele Erweiterung erlaubt, gibt es zwischen MAMMUT und dem Betriebssystem des oder der Rechner die *Virtuelle Maschine*, die bei Shared-Memory-Maschinen bzw. bei sequentiellen Maschinen entfällt, so daß MAMMUT dort direkt auf dem Betriebssystem aufsetzen kann. Bei anderen Architekturen werden weitgehende Anpassungen an diese Plattformen emuliert.

Erfahrungen im Bereich nichtlinearer dynamischer Systeme, wie sie anfangs erläutert wurden, haben gezeigt, daß die Behandlung

strukturell interessanter und anwendungsrelevanter mathematischer Probleme leicht zu einem *intermediate data swell* im zweistelligen Gigabyte-Bereich führen kann. MuPAD wurde deshalb im Hinblick auf die Bearbeitung algebraischer Daten in diesem Größenbereich ausgelegt.

Daneben gibt es in MuPAD, wie in jedem anderen System, einzelne Tools (Graphiktool, Interaktiver Source Level Debugger, Interaktives Hypertext On-line Helpsystem), die unabhängig von den mathematischen Zielsetzungen des Systems, den Entwurf mit einer außerordentlich bedienerfreundlichen Oberfläche abrunden.

## 4 Entwicklungstrends

### 4.1 Objektorientiertheit

Mathematik entwickelt sich seit etwa 2000 Jahren dadurch, daß ständig neue Objekte, geschaffen durch den menschlichen Geist, hinzukommen. Mathematik ist die Wissenschaft in der Objektorientiertheit per se realisiert ist. Ständig werden neue Strukturen alten Namen und Bezeichnungen zugeordnet (Überladen), und abstrakte Strukturen werden auf konkrete Objekte vererbt. Natürlich müssen mathematische Expertensysteme diese Arbeitsweise unterstützen und fördern. Deshalb wird auf Dauer kein Computeralgebrasystem überleben, welches nicht alle Möglichkeiten der Schaffung neuer Datentypen erlaubt, die Überladung von Kernfunktionalität ermöglicht und kategorielle Datenklassen realisieren läßt. Systeme, welche zum Beispiel, da die Funktionalität des  $*$ -Zeichens nun einmal im Kern festgelegt ist, die Multiplikation zweier Matrizen nur auf den kryptischen Befehl `evalm(A &* B)` hin ausführen, werden entweder eine schmerzhaft Anpassung ihres grundlegenden Entwurfs hinnehmen müssen, oder wenig Gegenliebe bei zukünftigen Anwendern finden. Axiom hat das Verdienst

dieses grundlegende Entwurfsproblem das erste Mal in den Blickpunkt der Anwender gerückt zu haben. Erstaunlich ist nur, daß seitdem außer MuPAD kein System entstanden ist, welches Objektorientiertheit im Kern verwirklicht hat. Objektorientiertheit ist in MuPAD in folgender Weise realisiert [5]:

Durch eine Implementation auf Kernebene ist dafür Sorge getragen, daß nur geringe Effizienzverluste in Kauf genommen werden müssen. Jede Kernfunktion ist überladbar, kann also entsprechend den neu geschaffenen Datentypen durch eine Libraryfunktion ersetzt werden, wobei jede Funktion anhand der evaluierten Argumente entscheidet, ob eine Überladung stattfindet. Die Suche nach den zuständigen Operationen findet über Hashtabellen oder Listen, die einen schnellen Zugriff erlauben, statt.

Der Vorteil einer solchen Vorgehensweise ist beachtlich: Eine Entwicklung generischer Algorithmen ist möglich, Domains erlauben eine Zusammenfassung der Operationen auf einem Datentypen, und durch Kategorien konstruiert man Klassen algebraischer Strukturen mit gemeinsamen Eigenschaften. Und alles was hier in Bezug auf mathematische Objekte möglich ist, kann ohne Einschränkung auf andere Wissensgebiete übertragen werden, die grundlegenden Tools dafür sind im System vorhanden, eine Ausdehnung auf andere Strukturen ist nur eine Frage von leicht erstellbaren Libraryprogrammen in einer komfortablen Hochsprache (die schnelles Prototyping erlaubt).

Wir wollen die mit Objekten gegebenen Arbeitsmöglichkeiten an einem kleinen Beispiel erläutern, der Behandlung der  $2 \times 2$ -Matrizen. Wir kreieren zuerst die Domain, die wir mit MQ bezeichnen, durch einen der vom System zur Verfügung gestellten Domainkonstruktoren

```
>> MQ:= SquareMatrix(2, Rational):
```

Neue Matrizen, das sind Domanelemente mit Zeigern auf eine Zusammenfassung der Operationen, werden nun durch Aufruf von MQ kre-

iert, ihre Typabfrage gibt Auskunft über die entsprechende Domain:

```
>> A:= MQ([[1, 2/3], [1/2, 3]]):
>> type(A);
```

```
      SquareMatrix(2, Rational)
```

Die nun auf die Matrizen wirkenden Operationen sind bei Nichtüberladung die üblichen Systemfunktionen, oder bei Überladung zum Beispiel von +, \* or ^ Funktionen, die diese Operationen in der üblichen Weise auf Matrizen ausdehnen. Man erhält dann durch den Aufruf von

```
>> 2*A^2 + 1/A;
```

das Ergebnis

```
      +-                +-
      | 91/24 , 61/12 |
      |                |
      | 61/16 , 457/24 |
      +-                +-

```

wobei das Inverse automatisch mitberechnet wurde. Man vergleiche dies etwa mit Funktionen wie `evalm(A &* B)`, die notwendig sind, um Matrizen in traditionellen ausdrucksorientierten Systemen zu multiplizieren. Der Nutzer braucht natürlich nicht viel von all dem zu verstehen, für ihn sind die neugeschaffenen Objekte genauso zu handhaben wie alle anderen Ausdrücke des Systems. Durch die Möglichkeit generische Algorithmen zu schreiben, wird die Entwicklungszeit mit solchen Systemen bedeutend abgekürzt, und die Strukturierung von Programmen wird sauberer.

Das Konzept geht natürlich weit über die Behandlung mathematischer Objekte hinaus: Will man zum Beispiel die Ausgabe von Matrizen als  $\text{\TeX}$ -formatierte Zeichenketten haben, so überlädt man:

```
>> MQ::print:= MQ::TeX:
```

um dann zu erhalten

```
>> 2*A^2 + 1/A;
```

```
\begin{array}{cc}
```

```

91/24 & 61/12\\
61/16 & 457/24\\
\end{array}

```

Man kann diese Möglichkeiten nun benutzen um etwa technische Objekte in dem Computeralgebrasystem über graphische und konstruktive Objekte ein- und auszugeben.

## 4.2 Modulkonzept

Computeralgebrasysteme sind bei vielen Aufgaben langsam, ja unerträglich langsam. Dies liegt an der Dualität von Systemfunktionen versus Libraryfunktionen (Bibliotheksfunktionen). Beide Funktionstypen bringen Vorteile und Nachteile. Systemfunktionen liegen in Binärcode vor, Libraryfunktionen werden vom System interpretiert. Systemfunktionen haben den Vorteil großer Geschwindigkeit und den Nachteil einer schwerfälligen Speicherdynamik, außerdem, und das ist viel bedenklicher, stören Sie die Erweiterbarkeit des Systems. Um diese Nachteile auszugleichen, gibt es die Bibliotheksfunktionen, womit man aber den Geschwindigkeitsvorteil verliert. Libraryfunktionen brauchen bei gleicher Funktionalität etwa 10 mal so viel Zeit.

Um die Vorteile beider Welten zu vereinen gibt es in MuPAD [9], [7] die sogenannten Modulfunktionen. Dies sind Binärcodeobjekte die zur Laufzeit geladen werden, die kernunabhängig sind, und eine Ein- und Auslagerung zur Laufzeit erlauben. Ihre Daten werden genau wie die Daten des Systems behandelt, der Nutzer sieht also keinen Unterschied zwischen Modulfunktionen und System- oder Libraryfunktionen. Modulfunktionen sind zudem einfach zu programmieren, portabel und erlauben die kernunabhängige Integration fremder Software in MuPAD. Bei der Programmierung dieser Funktionen, etwa in C++ kann man natürlich auch auf die Ressourcen des Systems zugreifen.

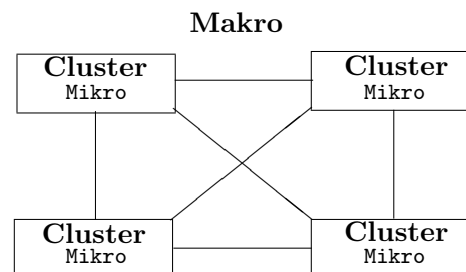
Mit diesem Konzept und den darin enthaltenen Möglichkeiten steht der Erweiterung des

Systems zu einem umfassenden mathematisch-technischen Expertensystem nichts mehr im Wege.

## 4.3 Parallelität

Eine weitere Möglichkeit der Leistungssteigerung von Computeralgebrasystemen bietet sich durch parallele Architekturen an. Man sollte bei rein mathematischen Fragen die dadurch gegebenen Möglichkeiten aber nicht überschätzen. Denn über die Parallelisierung vieler Algorithmen ist noch zu wenig bekannt. Trotzdem lohnt sich ein Einstieg in diese Technologie schon deshalb, weil man durch die Systemsprache mit ihren parallelen Konstrukten eine Möglichkeit zum Rapid Prototyping auf parallelen Architekturen bekommt. MuPAD hat eine parallele Erweiterung [6], von der wir hoffen, sie demnächst als beta-Release zur Nutzung freigeben zu können.

Das Ziel des Entwurfs ist Beschleunigung bei der Verarbeitung großer Datenmengen. Besonderes Augenmerk bei der Implementierung wurde auf die Systemunabhängigkeit gelegt. Grundlegendes Architekturmodell ist ein Cluster von Shared Memory Maschinen, eine Architektur wie sie etwa beim Power-Challenge-Array von SGI realisiert ist:



Der Entwurf der Parallelität unterscheidet zwischen Mikro- und Makroparallelität [8], Begriffe, die der Granularität der Tasks nur unzureichend entsprechen. Die parallelen Strukturen zwischen den Multiprozessoren des Netzwerks bilden die Makroparallelität, und auf den einzelnen Multiprozessoren spielt sich die

Mikroparallelität ab.

Die Realisierung bezüglich der gewählten Architektur geschieht folgendermaßen: Zwischen MUPAD und dem Betriebssystem der Hardwareplattform, gibt es eine weitere Schicht, eine als Virtuelle Maschine bezeichnete Shared-Memory-Maschine. Die Definition der untereinander lose gekoppelten Bereiche, die der Aufteilung in verschiedene Cluster entspricht, ist vom Nutzer zu konfigurieren. Die Parallelisierung im Bereich der Mikroparallelität führt das System bei Verwendung paralleler Konstrukte der Programmiersprache automatisch durch (Problem Heap). Bei Vorliegen einer wirklichen Shared-Memory-Maschine oder einer einzigen sequentiellen Maschine kann die Ebene der Virtuellen Maschine entfallen. Im Fall anderer Architekturen ist die Simulation einer Shared-Memory-Maschine nötig. MuPAD erzeugt unabhängig voneinander ablaufende parallele Prozesse und erlaubt den Zugriff auf gemeinsame Speicherbereiche. Die wesentlichen parallelen Konstrukte sind

- parbegin end\_par
- seqbegin end\_seq
- for from to parallel end\_for
- for in parallel end\_for

Bei Verwendung der Mikroparallelität hat man Unique Data Repräsentation über Speichergrenzen hinweg, was bei der Makroparallelität nicht so ist. Vorteil des Konzepts ist eine freie Skalierbarkeit von Speicherplatz und Rechenzeit und ein potentiell superlinearer Speedup durch Vermeidung von Swappen.

#### 4.4 Kompilierung: Bytecode

Als einfachste Methode zur Leistungssteigerung von Computeralgebra bietet sich auf den ersten Blick die Schaffung eines Compilers an, der die Libraryprogramme in Maschinencode übersetzt. So einfach das klingt, so beladen

ist es aber mit Tücken. Bisher gibt es einen Compiler nur in Axiom und in MuPAD (im Rahmen des Modulkonzepts als Prototyp). Bei beiden Systemen scheint mir der Durchbruch in dieser Frage noch nicht gelungen. Durch lange Kompilierungszyklen geht der Vorteil des direkten Interpretierens der Programme und die damit gegebene Möglichkeit des Experimentierens verloren. Die Entwicklungszeit wird auch nicht unbedingt kürzer. Die Maschinenabhängigkeit scheint mir noch zu stark zu sein, außerdem ist nicht immer klar, wieviel vom Code kompiliert werden soll, denn nicht immer ist die Neukompilation von effizienten Teilen des Systems von Vorteil.

Als Ausweg für die Zukunft bietet sich hier der Übergang zu einem Bytecode-Interpreter an. Bytecode kann man als eine eigene Maschinensprache verstehen, die auf einer virtuellen Hardware lauffähig ist. Da die Hardware in der Regel nur virtuell ist, muß man sie mit Hilfe von Software auf realer Hardware nachbilden. Man muß die *virtual machine* programmieren, was aber gar keinen so großen Effizienzverlust bedeuten muß. Die Bytecode-Maschinensprache ist in der Regel nicht so kompliziert wie es eine wirkliche Maschinensprache ist, da man sich über Optimierungen, die man möglicherweise irgendwie in die Hardware gießen kann, keine Gedanken zu machen braucht. Man wird auch die virtuelle Maschine so definieren, daß man sie auf jeder Hardware relativ einfach und effizient implementieren kann. Ein Vorteil von Bytecode ist, daß man die virtuelle Maschine auf einer neuen Hardware nur zu implementieren braucht, und danach alle Programme, die im Bytecode existieren, laufen lassen kann. Außerdem ist Bytecode meist kompakter als Maschinen-Code.

Die Untersuchung der Möglichkeiten von Bytecode wird die Entwicklung und Forschung beim Entwurf von Computeralgebrasystemen in den nächsten Jahren dominieren.

## 5 Interfaces

Es darf nicht übersehen werden, daß Mathematik eine Wissenschaft mit einem 2500-jährigen Interface-Problem ist. Die Rolle von Interfaces muß deshalb bei Mathematikssystemen, und insbesondere bei Computeralgebrasystemen, neu definiert werden. Die Frage von Interface und Effizienz eines Systems läßt sich nicht ganz trennen, denn das Interface-Problem wird insofern zum Effizienzproblem, als es darüber entscheidet, was Anwender mit Computeralgebra anfangen können, um ihre heutigen mathematischen Probleme und diejenigen, die mit ihrer Arbeit von morgen entstehen werden, zu lösen. Die Weiterentwicklung der Computeralgebra erfordert eine Übereinkunft der Mathematiker und Anwender über neue syntaktische Regeln der Mathematik sowie über das, was wir als Normalform bezeichnen und das, was ein einfacher, im Gegensatz zu einem komplizierten, Ausdruck ist.

Das *Oxford Dictionary of computing* definiert 'user interfaces' als *the means of communication between a human user and a computer system referring in particular to the use of input/output devices with supporting software*. Als typische Beispiele werden dann die üblichen MMGW's (mouse, menus, graphics und windows) gegeben. Obwohl diese Definition wohl der allgemeinen Ansicht entspricht, scheint sie mir zu eng, insbesondere dann wenn es sich um den Umgang mit Mathematik per Computer handelt. Wegen dieses eingeschränkten Verständnisses spielt die Diskussion von *User Interfaces* in der Computeralgebra kaum eine Rolle, obwohl jeder von uns weiß, daß das Verständnis mathematischer Sachverhalte häufig mehr von Notation und Darstellung als vom Inhalt abhängt, denn guter Notation wird es gelingen, die hinter einem Begriff liegende Intuition adequat auszudrücken. Damit stellt sich das Verständnis von Mathematik weitgehend als Interfaceproblem dar.

Allgemein dient der Computer als Transfermechanismus für den menschlichen Gedanken zum Anwendungsgebiet. Mathematik am Computer weicht von dieser allgemeinen Regel insofern ab, als der ursprüngliche Gedanke nach formaler Modifikation zum Menschen selbst zurückgegeben wird. Es handelt sich also häufig um einen geschlossenen Kreislauf, der deshalb den geeigneten Interface eine besondere Rolle zuweist.

Die Arbeit mit einem universellen Computeralgebrasystem basiert auf einer sich ständig ändernden Wechselwirkung zwischen Mensch und Maschine. Deshalb brauchen wir eine dynamische Definition von User Interface nicht eine statische, eine Definition welche die Beziehung des einzelnen Nutzers zur Mathematik mit einschließt. Die Frage von Nutzerinterfaces im Bereich der Mathematik ist mehr eine zweidimensionale als eine eindimensionale: eine Dimension ist durch die Funktionalität und die Eigenschaften des Systems gegeben, eine zweite durch das individuelle Verhältnis des Nutzers zur Mathematik. Die zweite Dimension fügt natürlich widersprüchliche Aspekte hinzu, denn das was dem einen Nutzer gefällt mag dem anderen Nutzer stark mißfallen. Jeder Entwickler kennt das am Beispiel der Nutzerproteste bei der Vereinfachung von  $\sqrt{a^2}$ . Läßt der Rechner das so stehen, dann protestieren die einen wegen der vermeintlichen Dummheit des Systems, wird dies zu  $a$  vereinfacht, dann hat man die Rigoristen auf dem Hals. Der Ausweg aus dieser Zwickmühle ist eine weitgehende Anpassungs- und Lernfähigkeit des Systems. Anpassungsfähigkeit haben wir heute, wenn auch in unzureichendem Maße, in einigen Bereichen. Über Lernfähigkeit gibt es hingegen nur wenige Gedanken.

Die entscheidenden Punkte nach denen die Nutzer die Qualität eines Systems beurteilen (oder beurteilen sollten) scheinen mir:

1. **Adaptability:** Wie gut kann das System des Nutzers intuitives Verständnis

von Mathematik widerspiegeln. Welche Werkzeuge hat es, um den Umgang mit Mathematik zu einer kreativen Begegnung zu machen.

2. **Persuasive Power:** Wie nachhaltig überzeugt das System den Nutzer, daß er es mit einem mächtigen Werkzeug für den *korrekten* Umgang mit Mathematik zu tun hat?
3. **MMGW-quality:** Die Qualität der üblichen MMGW's (mouse, menu, graphics und windows).
4. **Ergonomy:** Wie gut wird des Nutzers Arbeit in Bezug auf Effizienz, Geschwindigkeit und Bequemlichkeit unterstützt?
5. **Expandibility:** Wie gut kann das System mit des Nutzers wachsendem Verständnis und Wissen Schritt halten?

Wegen der großen Bandbreite von Nutzerprofilen, erfordert der erste Punkt ein adaptives System bei dem Datenstruktur und Notation der üblichen Arbeitsweise in der Mathematik (also der ohne Computereinsatz) entsprechen müssen. Gleiche Daten müssen in Abhängigkeit von Geschmack und Intuition des Nutzers verschiedene Darstellungen erlauben. Um ein einfaches Beispiel zu nehmen: Graphische oder auch technische Objekte müssen sowohl durch Formeln wie durch Bilder repräsentierbar sein, und beide Darstellungen müssen den gleichen Informationsgehalt haben. Also muß alles was man durch die üblichen Aktionen interaktiv verändert (Zoomen, Rotieren, etc.) sofort in Formeln rückübersetzbar sein, eine Forderung die von fast keinem System erfüllt wird. Die Ausgabe des Systems darf nicht nur aus schön anzusehenden Formeln bestehen, sondern muß die gesamte Baumstruktur repräsentieren, durch die man sich dann mit dem Cursor bewegen kann. Jeder Nutzer kann eine fast unendliche Liste solcher Forderungen aufstellen, die alle vom System der zukun-

erfüllt sein werden. Die wichtigste Forderung hier ist durch etwas gegeben, was wir schon behandelt haben: Jedes Computeralgebrasystem sollte adequate Möglichkeiten haben, neue mathematische Strukturen zu schaffen, also eine Vielfalt von Ausdrucksmöglichkeiten in der Wissenschaft erlauben, die sich die Beherrschung des unendlichen Universums formaler Strukturen zum Ziel gesetzt hat. Das geht, solange einem nichts besseres einfällt, nur durch nutzerdefinierte Domains und durch Kategorien, welche die Anbindung an allgemeine Eigenschaften erlauben. Natürlich wird der Nutzer, wenn er von diesen Möglichkeiten Gebrauch macht, auch darauf bestehen, über Aussehen und Gestalt dieser Datentypen zu entscheiden. Output, Input und Regeln müssen frei konfigurierbar sein.

Die zweite der aufgeführten Eigenschaften, bringt ein widersprüchliches Element in die Diskussion, da sie mehr psychologischer Natur ist. Denn über Regeln der Vereinfachung und Auswertung mathematischer Ausdrücke gibt es leider keine allgemein akzeptierte Übereinkunft. Der Ausweg hier scheint in der Einführung einer Algebra von Eigenschaften zu bestehen. Persuasive Power verlangt nicht nur die Korrektheit der Software (auch bezüglich einer rigoristischen inhaltlichen Sichtweise) sondern auch die prinzipielle Möglichkeit, alle Aspekte des Systems einzusehen. Ohne eine Offenlegung des gesamten Codes wird auf die Dauer kein System überleben können.

Punkt 3 geht nahtlos in 4 über. Die anspruchsvolle Realisierung dieses Punktes setzt in Zukunft ein lernfähiges System voraus. Gute interaktive help-Möglichkeiten, sowie ein vernünftiger Debugger sind natürlich nur minimale Anforderungen an diesen Punkt.

Der letzte Punkt fordert die nahtlose Einbindung fremder Binärcodeobjekte, die Schaffung verschiedenster Kommunikationsebenen, eine Beschreibung dessen was hier notwendig sein wird, erfordert einen eigenen Artikel.

Sicher sind wir bei dem, was in der Verwirklichung dieser 5 Punkte an Entwicklungen auf uns zukommt heute erst am Anfang der Möglichkeiten Mathematik am Computer zu treiben. Die Schaffung einer allgemeinen mathematisch-technischen Arbeitsumgebung hat gerade erst begonnen.

## 6 Perspektiven

Die Tendenzen beim gegenwärtigen Entwurf von Computeralgebrasystemen werden die wesentlichen Eigenschaften der in fünf Jahren verfügbaren Systeme bestimmen. Moduln, dynamisches Linken und der Möglichkeit Subsprachen zu kompilieren werden eine große Rolle spielen.

Die Möglichkeit der Leistungssteigerung durch Verwendung paralleler Systeme auf Multiprozessoren und verteilten Systemen wird einbezogen werden. Die in der Computeralgebra vorherrschende Rechnerarchitektur wird sicher die eines Clusters von Shared Memory Maschinen sein.

Im weiteren Verlauf der Entwicklung werden dem Nutzer im Rahmen von Computeralgebrasystemen Parsergeneratoren und Compiler-Compiler zur Verfügung stehen, alles Komponenten, die er nutzen kann, die aber dem durchschnittlichen Nutzer verborgen bleiben werden, und deren Nutzen er nur durch maßgeschneiderte Arbeitsumgebungen für die verschiedensten Problembereiche erfährt. Der anspruchsvolle Nutzer der Zukunft wird den jetzt noch vorhandenen (und störenden) Unterschied zwischen interpretierendem und kompilierendem System weitgehend verschwinden sehen.

Die im Rahmen der Implementation von Computeralgebra auf verteilten Systemen notwendigen Verbesserungen der Kommunika-

tionsstruktur von Computeralgebrasystemen, die fortschreitende Modularität der Systeme, sowie die Verbesserung der Effizienz bei der Verschickung baumstrukturierter Daten werden zur Entwicklung einer Netz-basierten multimedialen mathematischen Arbeitsumgebung führen, in der das Computeralgebrasystem nur eine von vielen Komponenten sein wird. Die wesentlichen Routinen zur Manipulation symbolischer Daten (nicht nur des eingebunden Computeralgebrasystems) werden auf Prozessorebene realisiert sein und das eigentliche System zur Manipulation symbolischer Daten wird lernfähig sein: Aus mehrfachen Durchläufen ähnlich strukturierter symbolischer Rechnungen wird es ein allgemeines Programm für den zugrunde liegenden Algorithmus ableiten.

## Publikationen

- [1] EDWARDS, R.E. *A Formal Background to Mathematics*. Unitext Springer Verlag, New York - Heidelberg - Berlin, 1979
- [2] B. Fuchssteiner: Nichtlineare Dynamische Systeme: Eine Fallstudie für die Anwendung von Computeralgebramethoden, in: *Geometry and Analysis: Trends in Teaching and Research*, B. Fuchssteiner and W. A. J. Luxemburg, eds., Bibliographisches Institut Mannheim, p.217-239, 1992
- [3] FUCHSSTEINER, B., AND AL. *MuPAD Benutzerhandbuch*. Birkhäuser, Basel, 1993.
- [4] FUCHSSTEINER, B., AND AL. *MuPAD Tutorial*. Birkhäuser, Basel, 1994  
<http://math-www.uni-paderborn.de/MuPAD/>
- [5] K. Drescher: Axioms, Categories and Domains, *mathPAD*, 4, 1, p.24-29, 1994
- [6] O. Kluge: Entwicklung einer Programmierumgebung für die Parallelverarbeitung in der Computer-Algebra, Teubner Verlag, Stuttgart, *MuPAD Reports*, 1997

- [7] H. Naundorf: Ein denotationales Modell für parallele objektbasierte Systeme, Teubner Verlag, Stuttgart, MuPAD Reports, 1997
- [8] The MuPAD Group: MuPAD User's Manual, Wiley - Teubner, p.615, Chichester, New York, Brisbane, Toronto, Singapore, Stuttgart, 1996  
<http://math-www.uni-paderborn.de/MuPAD/>
- [9] A. Sorgatz: Dynamische Module: Eine Verwaltung für Maschinencode-Objekte zur Steigerung der Effizienz und Flexibilität von Computeralgebra-Systemen, Teubner Verlag, Stuttgart, MuPAD Reports, 1996
- [10] WESTER, M. A review of CAS mathematical capabilities. *Computer Algebra Nederland Nieuwsbrief 13* (Dec. 1994), 41–48.  
[http://math.unm.edu/~wester/cas\\_review.html](http://math.unm.edu/~wester/cas_review.html)
- [11] P. Zimmermann: Westers test suite in MuPAD 1.2.2, Computer Algebra Nederland Nieuwsbrief, 14, p.53-64, 1995
- [12] P. Zimmermann: Westers test suite in MuPAD 1.3, The SAC Newsletter, 1, p.53-69, 1996