Skript zum Kurs
Einführung in das symbolische Rechnen
Wintersemester 1999/2000

H.-G. Gräbe, Institut für Informatik,
`http://www.informatik.uni-leipzig.de/~graebe`

25. Januar 2000

Inhaltsverzeichnis

Kapitel 0

Einleitung

0.1 Das Anliegen dieses Kurses

Nach dem Siegeszug von Taschenrechner und (in klassischen Programmiersprachen geschriebenen) Numerik-Paketen spielen heute Computeralgebra-Systeme (CAS) mit ihren Möglichkeiten, auch stärker formalisiertes mathematisches Wissen in algorithmisch aufbereiteter Form über eine einheitliche Schnittstelle zur Verfügung zu stellen, eine zunehmend wichtige Rolle. Solche Systeme, die im Gegensatz zu klassischen Anwendungen des Computers auch symbolische Kalküle beherrschen, werden zugleich in absehbarer Zeit wenigstens im naturwissenschaftlich-technischen Bereich den Kern umfassenderer Wissensrepräsentationssysteme bilden und die dort heute anzutreffende Zweieinigkeit aus Taschenrechner und Formel- und Tabellensammlung ablösen.

Damit gehört der souveräne Umgang mit solchen Systemen zu einer der Grundfertigkeiten, die von Hochschul-Absolventen wenigstens des mathematisch-naturwissenschaftlichen Bereichs in Zukunft erwartet werden.

Wie für das Programmieren in klassischen Programmiersprachen ist hierbei ein ausgewogenes Verhältnis zwischen Erwerb von eigenen Erfahrungen und methodischer Systematisierung dieser Kenntnisse angezeigt. Hier gibt es viele Parallelen zum Einsatz des Taschenrechners im Schulunterricht. Obwohl Schüler bereits frühzeitig eigene Erfahrungen im Umgang mit diesem "Werkzeug geistiger Arbeit", etwa im Fachunterricht, sammeln, wird auf dessen *systematische* Einführung ab Klasse 8 (sächsischer Lehrplan) nicht verzichtet. Schließlich gehört der qualifizierte Umgang mit dem Taschenrechner, insbesondere die Kenntnis seiner Eigenarten, Möglichkeiten und Grenzen, heute wie Lesen und Schreiben zu den elementaren Fertigkeiten, die jeder Schüler mit Verlassen der Schule beherrschen sollte. Dasselbe gilt in noch viel größerem Maße für die Fähigkeiten von Hochschulabsolventen im Umgang mit CAS, deren Möglichkeiten, Eigenarten und Grenzen ob der Komplexität dieses Instruments viel schwieriger auszuloten sind.

Da ich davon ausgehen kann, dass die Kursteilnehmer in der überwiegenden Zahl bereits eigene Erfahrungen mit einem der großen CAS gesammelt haben, werde ich mich auf eine systematische Darstellung der Möglichkeiten, Grenzen, Formen und Methoden des Einsatzes von Computern zur Ausführung symbolischer Rechnungen konzentrieren. Im Gegensatz zur einschlägigen Literatur soll dabei nicht eines der großen Systeme im Mittelpunkt stehen, sondern deren Gesamtheit Berücksichtigung finden. Eine zentrale Rolle werden die an unserer Universität weit verbreiteten Systeme Maple, MuPAD, Reduce und Mathematica spielen.

Ähnlich wie sich übergreifende Aspekte von Programmiersprachen nur aus deren vergleichender Betrachtung erschließen, ist ein solches Herangehen besser geeignet, die grundlegenden Techniken und Begriffe, die mit der Anwendung des Computers für symbolische Rechnungen verbunden sind, abzuheben.

Ein solcher Zugang erscheint mir auch deshalb sowohl gerechtfertigt als auch wünschenswert, weil es mittlerweile für jedes der großen Systeme eine Fülle von einführender Literatur höchst unterschiedlicher Qualität und verschiedenen Anspruchsniveaus gibt. Mehr noch, die Entwicklerteams der großen Systeme haben in den letzten Jahren viel Energie darauf verwendet, ihre Produkte auch von der äußeren Gestalt her nach modernen softwaretechnischen Gesichtspunkten aufzubereiten[1], so dass selbst ein ungeübter Nutzer in der Lage sein sollte, mit wenig Aufwand wenigstens die Grundfunktionalitäten des von ihm favorisierten Systems eigenständig zu erschließen. Bei Kenntnis grundlegender Techniken, die von all diesen Systemen verwendet werden, sollte diese Einarbeitung noch schneller gelingen.

Für den Nutzer von Computeralgebrasystemen wird es andererseits zunehmend schwieriger, die wirkliche Leistungsfähigkeit der Systeme hinter ihrer gleichermaßen glitzernden Fassade zu erkennen. Auch hierfür soll dieser Kurs ein gewisses Testmaterial zur Verfügung stellen, obwohl bei einem globalen Vergleich der Leistungsfähigkeit der verschiedenen Systeme durchaus Vorsicht geboten ist. Unterschiedliche Herangehensweisen im Design zusammen mit der jeweils spezifischen Entstehungsgeschichte führten dazu, dass die Leistungsfähigkeit unterschiedlicher Systeme in unterschiedlichen Bereichen sehr differiert und, mehr noch, sich verschiedene "mathematische Rigorosa" wie etwa in einem Teil der Wester-Liste [24, ch. 3] oder in Bernardins Übersicht [24, ch. 7] enthalten, mit sehr unterschiedlichem Aufwand im Nachhinein integrieren lassen. Diese Feinheiten führen an einigen Stellen zu unterschiedlichem Verhalten der verschiedenen Systeme bis hin zu sehr unterschiedlichen Antworten. Wir werden uns dabei auf die Frage beschränken, wie konsistent die einzelnen Systeme ihre eigenen Konzepte verfolgen, da gerade Übersichten wie die zitierten die Systementwickler manchmal dazu verleiten, bisher nicht beachtete Problemstellungen nur halbherzig in ihre Systeme zu integrieren.

Diese Konsistenzfrage steht als generelle Einsatzvoraussetzung für den Nutzer eines Computeralgebrasystems an zentraler Stelle. Leider ist sie nicht eindeutig zu beantworten, so lange *ein* Werkzeug *verschiedene* Nutzergruppen adressiert. Aber selbst die Einführung global einstellbarer "Nutzerprofile", wie in [24, ch. 3] vorgeschlagen, würde das Problem kaum lösen. Es ist deshalb in diesem Gebiet noch wichtiger als beim Taschenrechnereinsatz, sich kritisch mit dem Sinn bzw. Unsinn gegebener Antworten auseinanderzusetzen und sich über Art und Umfang möglicher Rechnungen und Antworten in groben Zügen im Klaren zu sein.

Allerdings begegnet man diesem "Zauberlehrlingseffekt", den Weizenbaum in seinem inzwischen klassischen Werk [22] in Bezug auf den Computereinsatz erstmals umfassend artikulierte, im Zusammenhang mit dem Einsatz moderner Technik immer wieder. Dieser als "Janusköpfigkeit" bezeichnete Effekt, dass der unqualifizierte und insbesondere nicht genügend reflektierte Einsatz mächtiger technischer Mittel zu unkontrollierten, unkontrollierbaren und in ihrer Wirkung unbeherrschbaren Folgen führen kann, wissen wir nicht erst seit Tschernobyl. Daraus resultierende Fragen sind inzwischen Gegenstand eines eigenen Wissensgebiets, des "technology assessment", geworden, dessen deutsche Übertragung "Technologiefolgenabschätzung" die zu thematisierenden Inhalte nur unvollkommen wiedergibt. Bezogen auf den Computer kann die Beschäftigung mit Computeralgebra auch hier wertvolle Einsichten vermitteln, die helfen, dessen Wekzeugcharakter gegenüber heute oft anzutreffender Fetischisierung mehr in den Vordergrund zu rücken.

0.2 Die Voraussetzungen

Symbolische Rechnungen sind das wohl grundlegendste methodische Handwerkszeug in der Mathematik und durchdringen alle ihrer Gebiete. Dabei stellt es sich heraus, dass vom methodischen Gesichtspunkt her ein kleines, wiederkehrendes algorithmisches Grundinstrumentarium in den ver-

---

[1] Dass dies nach dem Vorreiter MATHEMATICA mit einer stärkeren Kommerzialisierung auch der meisten anderen großen Systeme verbunden ist, mag der eine oder andere bedauern. Eine wichtige Erkenntnis scheint mir aber zu sein, dass sich im Gegensatz zur unmittelbaren Forschung an Algorithmen hierfür eben keine öffentlichen Mittel allokieren lassen. Auch scheinen die subtilen Mechanismen, die zum Erfolg freier Softwareprojekte führen, hier nicht zu greifen. Andererseits stehen die geforderten Lizenzpreise gerade der Studentenversionen in keinem Verhältnis zur Leistungsfähigkeit der Systeme. Mit diesen Mittel kann man im Wesentlichen wohl nur Supportleistungen, nicht aber tieferliegende algorithmische Untersuchungen refinanzieren.

Die Diskussion der Chancen und Risiken des Zusamenfließens öffentlich geförderter wissenschaftlicher Arbeit und privatwirtschaftlicher Supportleistung an diesem Beispiel erscheint mir auch deshalb wünschenswert, weil ähnliche Entwicklungen in anderen Bereichen der Informatik (Stichwort: LINUX-Distributionen) ebenfalls stattfinden.

schiedensten Situationen variierend zum Einsatz kommt. Dies mag nicht überraschen, ist doch ein ähnliches Universalitätsprinzip programmiersprachlicher Mittel zur Beschreibung von Algorithmen und Datenstrukturen gut bekannt und kann sogar theoretisch begründet werden.

Entsprechend sind Computeralgebrasysteme bereits heute Vielzweckwerkzeuge, die eine gehörige Portion symbolischer Rechnungen in kompliziertesten mathematischen Kontexten übernehmen können. Für einen Einführungskurs stellt dies eine doppelte Schwierigkeit dar, da einerseits die zu demonstrierenden Prinzipien erst in nichttrivialen Anwendungen zu überzeugender Entfaltung kommen, andererseits solche Anwendungen ohne Kenntnis ihres Kontextes nur schwer nachvollziehbar sind. Dies verlangt, eine wohlüberlegte Auswahl zu treffen.

Im folgenden werden wir uns deshalb, neben einer ausführlichen Diskussion der Stellung des symbolischen Rechnens im Wissenschaftsgebäude, beschränken auf die Darstellung wichtiger Designaspekte eines CAS der zweiten Generation und die Diskussion theoretischer und praktischer Aspekte der Simplifikationsproblematik als besonderem Charakteristikum des symbolischen Rechnens. Im letzten Teil des Kurses werden wir schließlich am Beispiel der symbolischen Behandlung algebraischer Zahlen diese Konzepte in ihrem komplexen Zusammenspiel erleben, Anforderungen und Lösungen gegenüberstellen und dabei einige auf den ersten Blick merkwürdige Ansätze besser verstehen. Diesen Teil beschließt ein Überblick über die Algorithmen zur Integration rationaler Funktionen, in denen algebraische Zahlen auf natürliche Weise auftreten.

Nicht berühren werden wir das Feld der *Differentialgleichungen*, da die Schwierigkeiten der damit verbundenen Mathematik in keinem Verhältnis zu den dabei gewinnbaren neuen Einsichten in Zusammenhänge des symbolischen Rechnens steht. Gleichfalls ausgeklammert bleiben Fragen der graphischen Möglichkeiten der CAS, die ein wichtiges Mittel der Visualisierung wissenschaftlicher Zusammenhänge sind und für viele Nutzer den eigentlichen Reiz eines großen CAS darstellen, aber nicht zum engeren Gebiet des symbolischen Rechnens gehören. Dasselbe gilt für die mittlerweile ausgezeichneten Präsentationsmöglichkeiten der integrierten graphischen Oberflächen der meisten der betrachteten Systeme.

Kapitel 1

Computeralgebrasysteme im Einsatz

In diesem Kapitel wollen wir Computeralgebrasysteme in verschiedenen Situationen als Rechenhilfsmittel kennen lernen und dabei erste Besonderheiten eines solchen Systems gegenüber rein numerisch basierten Rechenhilfsmitteln heraus arbeiten. Die folgenden Rechnungen basieren auf MuPAD 1.4, hätten aber mit jedem der großen CAS in ähnlicher Weise ausgeführt werden können.

1.1 Computeralgebrasysteme als Taschenrechner für Zahlen

Mit einem CAS kann man natürlich zunächst alle Rechnungen ausführen, die ein klassischer Taschenrechner beherrscht.

Das umfasst neben den Grundrechenarten auch eine Reihe mathematischer Funktionen:

```
1.23+2.25;
```

$$3.48$$

```
12+23;
```

$$35$$

```
10!;
```

$$3628800$$

```
100!;
```

93326215443944152681699238856266700490715968264381621468592963895217599993
22991560894146397615651828625369792082722375825118521091686400000000000000
0000000000

Am letzten Beispiel erkennen wir eine erste Besonderheit: CAS rechnen im Gegensatz zu Taschenrechnern mit ganzen Zahlen mit *voller* Genauigkeit. Die in ihnen implementierte *Langzahlarithmetik* erlaubt es, die entsprechenden Umformungen *exakt* auszuführen. Dies wird auch bei der Behandlung rationaler Zahlen

```
1/2+2/3;
```

$$7/6$$

```
sum(1/i,i=1..50);
```

$$13943237577224054960759/3099044504245996706400$$

```
float(%);
```

$$4.499205338$$

sowie irrationaler Zahlen

```
sqrt(2);
```

$$\sqrt{2}$$

```
PI;
```

$$\pi$$

deutlich. Diese werden nicht durch numerische Näherungswerte ersetzt, sondern bleiben als *symbolische Ausdrücke* in einer Form stehen, die uns aus einem streng mathematischen Kalkül wohlbekannt ist. Im Gegensatz zu numerischen Approximationen, bei denen sich die Zahl `3.1415926535` qualitativ kaum von der Zahl `3.1426968052` unterscheidet (letzteres ist $2.222222222 \cdot \sqrt{2}$, auf 10 Stellen nach dem Komma genau ausgerechnet), verbirgt sich hinter einem solchen Symbol eine wohldefinierte *mathematische Semantik*. So ist etwa

> $\sqrt{2}$ diejenige (eindeutig bestimmte) positive reelle Zahl, deren Quadrat gleich 2 ist.

Ein CAS verfügt über Mittel, diese Semantik (bis zu einem gewissen Grade) darzustellen. So "weiss" MuPAD einiges über die Zahl π

```
sin(PI);
```

$$0$$

```
sin(PI/4);
```

$$\frac{\sqrt{2}}{2}$$

und sogar

```
sin(PI/5);
```

$$\frac{\sqrt{2}\sqrt{-\sqrt{5}+5}}{4}$$

Ein Programm, das nur einen (noch so guten) Näherungswert von π kennt, kann die Information $\sin(\pi) = 0$ prinzipiell nicht *exakt* ableiten:

```
DIGITS:=20: p:=float(PI); sin(p); unassign(DIGITS):
```

$$3.1415926535897932384$$

$$0.0000000000000000000000000000050487097934144755546$$

Ohne hier auf Details der inneren Darstellung einzugehen, halten wir fest, dass CAS symbolischen Ausdrücken *Eigenschaften* zuordnen, die deren mathematischen Gehalt widerspiegeln.

Wie eben gesehen, kann eine dieser Eigenschaften insbesondere darin bestehen, dass dem CAS auch ein Verfahren zur Bestimmung eines *numerischen Näherungswerts* bekannt ist. Dieser kann mit einer beliebig vorgegebenen Präzision berechnet werden, die softwaremäßig auf der Basis der Langzahlarithmetik operiert. Damit ist die Numerik zwar oft deutlich langsamer als die auf Hardware-Operationen zurückgeführte Numerik klassischer Programmiersprachen, erlaubt aber genauere und insbesondere adaptive Approximationen.

Ähnlich wie auf dem Taschenrechner stehen auch wichtige mathematische Funktionen und Operationen zur Verfügung, allerdings in einem wesentlich größeren Umfang als dort. So wissen CAS, wie man von ganzen Zahlen den größten gemeinsamen Teiler, die Primfaktorzerlegung oder die Teiler bestimmt, die Primzahleigenschaft testet, nächstgelegene Primzahlen ermittelt und vieles mehr.

```
gcd(2^30-1,3^20-1);
```

$$11$$

```
Factor(12!);
```

$$2^{10} \cdot 3^5 \cdot 5^2 \cdot 7 \cdot 11$$

```
isprime(12!-1);
```

$$TRUE$$

1.2 Computeralgebrasysteme als Taschenrechner für Formeln und symbolische Ausdrücke

Die wichtigste Besonderheit eines Computeralgebrasystems gegenüber Taschenrechner und klassischen Programmiersprachen ist ihre

$$\boxed{\text{Fähigkeit zur Manipulation symbolischer Ausdrücke.}}$$

Dies wollen wir zunächst an symbolischen Ausdrücken demonstrieren, die gewöhnliche Variablen enthalten, also Literale wie x, y, z ohne weitergehende mathematische Semantik. Aus solchen Symbolen kann man mit Hilfe der 4 Grundrechenarten *rationale Funktionen* zusammenstellen. Betrachten wir dazu einige Beispiele:

```
u:= a/((a-b)*(a-c)) + b/((b-c)*(b-a)) + c/((c-a)*(c-b));
```

$$\frac{a}{(a-b)\,(a-c)} + \frac{b}{(-a+b)\,(b-c)} + \frac{c}{(-a+c)\,(-b+c)}$$

Es ergibt sich die Frage, ob man diesen Ausdruck weiter vereinfachen kann. Am besten wäre es, wenn man einen gemeinsamen Zähler und Nenner bildet, welche zueinander teilerfremd sind, und diese in ihrer expandierten Form darstellt. Das ermöglicht die Funktion `normal`.

```
normal(u);
```

$$0$$

Das Ergebnis mag verblüffen; jedenfalls sieht man das dem ursprünglichen Ausdruck nicht ohne weiteres an.

Eine solche normalisierte Darstellung ist nicht immer zweckmäßig, weshalb diese Umformung nicht automatisch vorgenommen wurde. So erhalten wir etwa

```
u:=(x^15-1)/(x-1);
```

$$\frac{x^{15}-1}{x-1}$$

```
normal(u);
```

$$x+x^2+x^3+x^4+x^5+x^6+x^7+x^8+x^9+x^{10}+x^{11}+x^{12}+x^{13}+x^{14}+1$$

Betrachten wir allgemein Ausdrücke der Form

$$u_n := \frac{a^n}{(a-b)*(a-c)} + \frac{b^n}{(b-c)*(b-a)} + \frac{c^n}{(c-a)*(c-b)}$$

und untersuchen, wie sie sich unter Normalformbildung verhalten. Wir verwenden dazu die Funktion `subs`, die lokal eine Variable durch einen anderen Ausdruck, in diesem Fall eine Zahl, ersetzt.

```
normal(subs(u,n=2));
```

$$1$$

```
normal(subs(u,n=3));
```

$$a+b+c$$

```
normal(subs(u,n=4));
```

$$ab+ac+bc+a^2+b^2+c^2$$

```
normal(subs(u,n=5));
```

$$abc+a^3+b^3+c^3+ab^2+a^2b+ac^2+a^2c+bc^2+b^2c$$

Wir sehen, dass sich in jedem der betrachteten Fälle die rationale Funktion u_n zu einem Polynom vereinfacht.

Tragen die in die Formeln eingehenden Symbole weitere semantische Information, so sind auch komplexere Vereinfachungen möglich. So werden etwa Ausdrücke, die die imaginäre Einheit I enthalten, wie erwartet vereinfacht:

```
(1+I)^n $ n=1..10;
```

$$1+i, 2i, -2+2i, -4, -4+-4i, -8i, 8+-8i, 16, 16+16i, 32i$$

```
(3+I)/(2-I);
```

$$1+i$$

Oftmals müssen solche Umformungen durch entsprechende Funktionsaufrufe gezielt angestoßen werden.

```
(sqrt(2)+sqrt(3))^n $ n=2..5;
```

$$\left(\sqrt{2}+\sqrt{3}\right)^2, \left(\sqrt{2}+\sqrt{3}\right)^3, \left(\sqrt{2}+\sqrt{3}\right)^4, \left(\sqrt{2}+\sqrt{3}\right)^5$$

```
expand((sqrt(2)+sqrt(3))^n) $n=2..5;
```

$$2\sqrt{2}\sqrt{3}+5, 11\sqrt{2}+9\sqrt{3}, 20\sqrt{2}\sqrt{3}+49, 109\sqrt{2}+89\sqrt{3}$$

Manchmal ist es nicht einfach, das System zu "überreden", genau das zu tun, was man will. So liefern weder `expand` noch `normal` oder `simplify` eine Form von $(\sqrt{2}+\sqrt{3})^{-1}$ mit rationalem Nenner. Erst die offensichtlich aufwendige Funktion (Laufzeit 1.4 s.)

```
radsimp(1/(sqrt(2)+sqrt(3)));
```

$$\sqrt{3}-\sqrt{2}$$

liefert das erwartete Ergebnis, das man auch schnell im Kopf ausrechnen kann. Der Grund liegt darin, dass es aus Effizienzgründen nicht klug ist, Nenner rational zu machen. Wir kommen auf diese Frage später zurück.

1.3 CAS als Problemlösungs-Umgebungen

Wir haben in obigen Beispielen bereits in bescheidenem Umfang programmiersprachliche Mittel eingesetzt, um unsere speziellen Wünsche zu formulieren.

> Das Vorhandensein einer voll ausgebauten Programmiersprache, mit der man den Interpreter des jeweiligen CAS gut steuern kann, ist ein weiteres Charakteristikum der betrachteten Systeme (nur Derive macht hier bisher eine Ausnahme und orientiert sich stärker an einer Drag-and-Drop-Philosophie). Dabei werden alle gängigen Sprachkonstrukte einer imperativen Programmiersprache unterstützt und noch um einige Spezifika erweitert, die aus der Natur des symbolischen Rechnens folgen und über die weiter unter zu sprechen sein wird.

Mit diesem Instrumentarium und dem eingebauten mathematischen Wissen werden CAS so zu einer vollwertigen Problemlösungs-Umgebung, in der man neue Fragestellungen und Vermutungen (mathematischer Natur) ausgiebig testen und untersuchen kann. Selbst für zahlentheoretische Fragestellungen reichen die Möglichkeiten dabei weit über die des Taschenrechners hinaus.

Betrachten wir etwa Aufgaben der Art:

Bestimmen Sie die letzte Ziffer der Zahl 2^{100},

wie sie in Schülerarbeitsgemeinschaften vor Einführung von CAS zum Kennenlernen des Rechnens mit Resten gern gestellt wurden. Die Originalaufgabe ist für ein CAS gegenstandslos, da man die ganze Zahl leicht ausrechnen kann.

```
2^100;
```

$$1267650600228229401496703205376$$

Erst im Bereich von Exponenten in der Größenordnung 1000000 wird der Verbrauch von Rechenzeit ernsthaft spürbar und die dann über 300000-stelligen Zahlen zunehmend unübersichtlich.

Der ursprüngliche Sinn der Aufgabe bestand darin, zu erkennen, dass man zur Berechnung der letzten drei Ziffern nicht die gesamte Potenz, sondern nur deren Rest (mod 10) berechnen muss und dass Potenzreste 2^k (mod 10) eine periodische Folge bilden. Mit unserem CAS kann man den Rechner für eine wesentlich weitergehende Untersuchung derselben Problematik einsetzen. Da dieses in der Lage ist, die ermüdende Berechnung der Potenzreste zu übernehmen[1], können wir nach Regelmäßigkeiten für die Potenzreste für beliebige Moduln fragen.

```
2^k mod 10 $ k=1..20;
```

$$2, 4, 8, 6, 2, 4, 8, 6, 2, 4, 8, 6, 2, 4, 8, 6, 2, 4, 8, 6$$

```
2^k mod 3 $ k=1..20;
```

$$2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1$$

```
2^k mod 11 $ k=1..20;
```

$$2, 4, 8, 5, 10, 9, 7, 3, 6, 1, 2, 4, 8, 5, 10, 9, 7, 3, 6, 1$$

```
2^k mod 17 $ k=1..20;
```

$$2, 4, 8, 16, 15, 13, 9, 1, 2, 4, 8, 16, 15, 13, 9, 1, 2, 4, 8, 16$$

Diese wenigen Kommandos liefern eine Fülle von Material, das uns schnell verschiedene Regelmäßigkeiten vermuten lässt, die man dann gezielter experimentell untersuchen kann. So taucht für primen Modul in der Folge stets eine 1 auf, wonach sich die Potenzreste offensichtlich wiederholen. Geht man dieser Eigenschaft auf den Grund, so erkennt man die Bedeutung primer Restklassen. Auch die Länge der Folge zwischen dem Auftreten der 1 folgt gewissen Gesetzmäßigkeiten, die ihre Verallgemeinerung in elementaren Aussagen über die Ordnung von Elementen in der Gruppentheorie finden.

Wir wollen diese auch aus didaktischen Gesichtspunkten interessanten Möglichkeiten eines CAS zur Erforschung von Eigenschaften ganzer Zahlen an einem weiteren Beispiel verdeutlichen:

Eine Zahl n heisst *perfekt*, wenn sie mit der Summe ihrer echten Teiler übereinstimmt, d.h. die Summe *aller* ihrer Teiler gerade $2\,n$ ergibt. Die Untersuchung solcher Zahlen kann man bis in die Antike zurückverfolgen. Bereits in der Pythagoräischen Schule im 6. Jh. vor Christi werden solche Zahlen betrachtet. EUKLID kannte eine Formel, nach der man alle gerade perfekte Zahlen findet, die erstmals von EULER exakt bewiesen wurde.

Wir wollen nun ebenfalls versuchen, uns einen Überblick über die perfekten Zahlen zu verschaffen. Das MuPAD-Paket `numlib` enthält die Funktion `sigma(n)`, mit der wir die Teilersumme der natürlichen Zahl n berechnen können. Mit der folgenden Schleife verschaffen wir uns erst einmal einen Überblick über die perfekten Zahlen bis 800:

[1] Notwendige Nebenbemerkung für den Einsatz in der Schule: Nachdem dieser Gegenstand genügend geübt worden ist.

```
export(numlib):
for i from 2 to 800 do
        if 2*i=sigma(i) then print(i) end_if
end_for:
```

6
28
496

Betrachten wir die gefundenen Zahlen näher:

$$6 = 2 \cdot 3, \quad 28 = 4 \cdot 7, \quad 496 = 16 \cdot 31.$$

Alle diese Zahlen haben die Gestalt $2^{k-1}(2^k - 1)$. Wir wollen deshalb versuchen, perfekte Zahlen dieser Gestalt zu finden:

```
for k from 2 to 40 do n:=2^(k-1)*(2^k-1):
        if 2*n=sigma(n) then print(k,n,yes)
        else print(k,n,no)
        end_if
end_for:
```

k	n	perfekt ?
2	6	yes
3	28	yes
4	120	no
5	496	yes
6	2016	no
7	8128	yes
8	32640	no
9	130816	no
10	523776	no
11	2096128	no
12	8386560	no
13	33550336	yes
14	134209536	no
15	536854528	no
16	2147450880	no
17	8589869056	yes
18	34359607296	no
19	137438691328	yes
20	549755289600	no
...		

Die Ausgabe ist als Tabelle zusammengestellt und wegen ihrer Länge gekürzt. Sie bietet umfangreiches experimentelles Material, auf dessen Basis sich qualifizierte Vermutungen über bestehende Gesetzmäßigkeiten aufstellen lassen. Es scheint insbesondere so, als ob perfekte Zahlen nur für prime k auftreten. Jedoch liefert nicht jede Primzahl k eine perfekte Zahl n.

Derartige Beobachtung kann man nun versuchen, mathematisch zu untermauern, was in diesem Fall nur einfache kombinatorische Überlegungen erfordert: Die Teiler der Zahl $n = p_1^{a_1} \cdot \ldots \cdot p_m^{a_m}$ haben offensichtlich genau die Gestalt $t = p_1^{b_1} \cdot \ldots \cdot p_m^{b_m}$ mit $0 \le b_i \le a_i$. Ihre Summe beträgt

$$\sigma(n) = (1 + p_1 + \ldots + p_1^{a_1}) \cdot \ldots \cdot (1 + p_m + \ldots + p_m^{a_m}).$$

In der Tat, multipliziert man den Ausdruck aus, so hat man aus jeder Klammer einen Summanden zu nehmen und zu einem Produkt zusammenzufügen. Alle möglichen Auswahlen ergeben genau die beschriebenen Teiler von n.

Die Teilersumme $\sigma(n)$ ergibt sich also unmittelbar aus der Kenntnis der Primfaktorzerlegung der Zahl n. Ist insbesondere $n = a \cdot b$ eine zusammengesetzte Zahl mit zueinander teilerfremden Faktoren a, b , so gilt offensichtlich $\sigma(n) = \sigma(a) \cdot \sigma(b)$.

Wenden wir das auf die gerade perfekte Zahl $n = 2^{k-1} b$ $(k > 1, b$ ungerade) an, so gilt wegen $\sigma(2^{k-1}) = 1 + 2 + 2^2 \ldots + 2^{k-1} = 2^k - 1$

$$2n = 2^k\, b = \sigma(n) = (2^k - 1)\sigma(b).$$

Also ist $2^k - 1$ ein Teile von $2^k\, b$ und als ungerade Zahl damit auch von b. Es gilt folglich $b = (2^k - 1)c$ und somit $\sigma(b) = 2^k\, c = b + c$. Da b und c Teiler von b sind und $\sigma(b)$ *alle* Teiler von b aufsummiert, hat b nur die Teiler b und $c = 1$, muss also eine Primzahl sein. Da auch umgekehrt jede solche Zahl eine perfekte Zahl ist haben wir damit folgenden Satz bewiesen:

Satz 1 *Jede gerade perfekte Zahl hat die Gestalt* $n = 2^{k-1}(2^k - 1)$, *wobei* $P := 2^k - 1$ *eine Primzahl sein muss.*

Ungerade perfekte Zahlen sind bis heute nicht bekannt, ein Beweis, dass es solche Zahlen nicht gibt, aber auch nicht gefunden worden.

1.4 Computeralgebrasysteme als Expertensysteme

Neben den bisher betrachteten Rechenfertigkeiten und der Programmierbarkeit, die ein CAS auszeichnen, spielt das

$$\boxed{\text{in ihnen gespeicherte mathematische Wissen}}$$

für Anwender die entscheidende Rolle. Dieses deckt, wenigstens insofern wir uns auf die dort vermittelten algorithmischen Fertigkeiten beschränken, weit mehr als den Stoff der gymnasialen Oberstufe ab und umfasst die wichtigsten symbolischen Kalküle der Analysis (Differenzieren und Integrieren, Taylorreihen, Grenzwertberechnung, Trigonometrie, Rechnen mit speziellen Funktionen), der Algebra (Matrizen- und Vektorrechnung, Lösen von linearen und polynomialen Gleichungssystemen, Rechnen in Restklassenbereichen), der Kombinatorik (Summenformeln, Lösen von Rekursionsgleichungen, kombinatorische Zahlenfolgen) und vieler anderer Gebiete der Mathematik.

Für viele Kalküle aus mathematischen Teildisziplinen, aber zunehmend auch solche aus verwandten Bereichen anderer Natur- und Ingenieurwissenschaften, ja selbst der Finanzmathematik, gibt es darüber hinaus spezielle Pakete, auf die man bei Bedarf zugreifen kann. Dieses sprunghaft wachsende Expertenwissen ist der eigentliche Kern eines CAS als Expertensystem. Man kann mit gutem Gewissen behaupten, dass heute bereits große Teile der algorithmischen Mathematik in dieser Form verfügbar sind und auch algorithmisches Wissen aus anderen Wissensgebieten zunehmend erschlossen wird. In dieser Richtung spielt das System MATHEMATICA seit Jahren eine Vorreiterrolle.

In dem Zusammenhang ist die im deutschen Sprachraum verbreitete Bezeichnung "Computeralgebra" irreführend, denn es geht durchaus nicht nur um algebraische Algorithmen, sondern auch um solche aus der Analysis und anderen Gebieten der Mathematik. Entscheidend ist allein der *algebraische Charakter* der entsprechenden Manipulationen als endliche Kette von Umformungen symbolischer Ausdrücke. Und genau so geht ja etwa der Kalkül der Differentialrechnung vor: die konkrete Berechnung von Ableitungen elementarer Funktionen erfolgt nicht nach der (auf dem Grenzwertbegriff aufbauenden) Definition, sondern wird durch geschicktes Kombinieren verschiedener *Regeln*, wie der Ketten-, der Produkt- und der Quotientenregel, auf entsprechende Grundableitungen zurückgeführt.

```
diff(x^n,x);
```

$$\frac{n x^n}{x}$$

```
diff(x^2*sin(x)*ln(x+a),x$2); # Zweite Ableitung #
```

$$2\sin(x)\log(a+x)+4x\cos(x)\log(a+x)+\frac{4x\sin(x)}{a+x}-x^2\sin(x)\log(a+x)+\frac{2x^2\cos(x)}{a+x}-\frac{x^2\sin(x)}{(a+x)^2}$$

Beim Integrieren kommen darüber hinaus auch prozedurale Verfahren zum Einsatz, wie etwa die Partialbruchzerlegung, die ihrerseits die Faktorzerlegung des Nennerpolynoms und das Lösen von (linearen) Gleichungssystemen verwendet, die aus einem Ansatz mit unbestimmten Koeffizienten gewonnen werden. Auf diese Weise, und das ist ein weiteres wichtiges Moment des Einsatzes von CAS,

spielen algorithmische Fertigkeiten aus sehr verschiedenen Bereichen der Mathematik nahtlos miteinander zusammen.

```
p:=(2*x^3+2*x^2+2*x-2)/(x^4+x^2+1);
```

$$\frac{2\,x^3+2\,x^2+2\,x-2}{x^4+x^2+1}$$

```
int(p,x);
```

$$-\frac{1}{2}\,\ln(x^2+x+1)-\frac{\sqrt{3}}{3}\,\arctan(\frac{\sqrt{3}(2\,x+1)}{3})+\frac{3}{2}\,\ln(x^2-x+1)+\frac{\sqrt{3}}{3}\,\arctan(\frac{\sqrt{3}(2\,x-1)}{3})$$

```
diff(%,x);
```

$$\frac{3\,(2x-1)}{2\left((x-1/2)^2+3/4\right)}-\frac{2x+1}{2\left((x+1/2)^2+3/4\right)} \\ +\frac{1}{2\,(x-1/2)^2\left(\frac{3}{4\,(x-1/2)^2}+1\right)}-\frac{1}{2\,(x+1/2)^2\left(\frac{3}{4\,(x+1/2)^2}+1\right)}$$

```
normal(%-p);
```

$$0$$

```
int(1/(x^4-1),x);
```

$$\frac{\log(x-1)}{4}-\frac{\log(x+1)}{4}+\frac{i}{4}\log(x-i)-\frac{i}{4}\log(x+i)$$

Expertenwissen wird zum Lösen von verschiedenen Problemklassen benötigt, wobei der Nutzer in den seltensten Fällen wissen (und wissen wollen) wird, welche konkreten Algorithmen im jeweiligen Fall genau anzuwenden sind. Deshalb bündeln die CAS die algorithmischen Fähigkeiten zum Auflösen gewisser Sachverhalte in einem oder mehreren

`solve`-Operatoren,

die den Typ eines Problems erkennen und geeignete Lösungsalgorithmen aufrufen.

So bündelt der `solve`-Operator von MuPAD Wissen über Algorithmen zum Auffinden der Nullstellen univariater Polynome, von linearen und polynomialen Gleichungssystemen.

```
solve(x^2+x+1,x);
```

$$\left\{-1/2i\sqrt{3}-1/2, 1/2i\sqrt{3}-1/2\right\}$$

```
solve({x+y=2,x-y=1},{x,y});
```

$$\{\{y=1/2, x=3/2\}\}$$

```
solve({x^2+y=2,3*x-y^2=2},{x,y});
```

$$\Big\{[x=1,y=1]\,,[x=2,y=-2]\,,$$
$$\left[x=\tfrac{y-5}{3}, y=3/2i\sqrt{3}+1/2\right],$$
$$\left[x=\tfrac{y-5}{3}, y=-3/2i\sqrt{3}+1/2\right]\Big\}$$

Selbst für kompliziertere Gleichungen, die trigonometrische Funktionen enthalten, werden Lösungen angegeben:

```
s:=solve(sin(x)=1/2);
```

$$\left\{\frac{\pi}{6}, \frac{13\pi}{6}, -\frac{11\pi}{6}, \ldots, \frac{5\pi}{6}, \frac{17\pi}{6}, -\frac{7\pi}{6}, \ldots\right\}$$

Allerdings sieht das Ergebnis netter aus als es ist; es bedarf schon einiger Kenntnis von MuPAD, um es weiterzuverarbeiten. Legen wir Schulkenntnisse an, so wäre als Lösung eher ein Ergebnis der Form $\pm\frac{\pi}{6}+2\,k\,\pi$ zu erwarten gewesen. Dass die Lösung intern in eben dieser Form dargestellt wird, erkennen wir mit

```
map(s,expr);
```

$$\Big\{\text{Dom::DiscreteSet}\left(\tfrac{\pi}{6}+2k\pi, k=-\infty\ldots\infty, TRUE\right),$$
$$\text{Dom::DiscreteSet}\left(\tfrac{5\pi}{6}\;+2k\pi, k=-\infty\ldots\infty, TRUE\right)\Big\}$$

Wir sehen weiterhin, dass auch ein spezieller Datentyp-Konstruktor vor unseren Augen verborgen wurde, dessen Eigenschaften wir aber zur Manipulation von s kennen müssen. Nach Studium der entsprechenden (Online-)Dokumentation wird man mit

```
map(s,_index,1..5);
```

$$\left\{\frac{25\pi}{6}, \frac{5\pi}{6}, \frac{29\pi}{6}, -\frac{7\pi}{6}, -\frac{11\pi}{6}, \frac{13\pi}{6}, \frac{17\pi}{6}, -\frac{19\pi}{6}, -\frac{23\pi}{6}, \frac{\pi}{6}\right\}$$

zum Beispiel die ersten 5 Werte aus beiden Folgen extrahieren können.

Maples Lösung derselben Aufgabe lautet

```
solve(sin(x)=1/2);
```

$$1/6\,\pi$$

Wenn wir uns erinnern, dass die Lösungsmenge trigonometrischer Gleichungen periodisch ist, also mit x auch $x+2\pi$ die Gleichung erfüllt, so haben wir schon die Hälfte der tatsächlichen Lösungsmenge gefunden. Warum kommt Maple nicht selbst auf diese Idee? Auch hier hilft erst ein Blick in die (Online-)Dokumentation weiter; setzt man eine spezielle Systemvariable richtig, so erhalten wir das erwartete Ergebnis, allerdings in einer recht verklausulierten Form:

```
_EnvAllSolutions:=true;
```

$$\texttt{\_EnvAllSolutions} := true$$

```
solve(sin(x)=1/2);
```

$$1/6\,\pi + 2/3\,\pi\,\texttt{\_B1\textasciitilde} + 2\,\pi\,\texttt{\_Z1\textasciitilde}$$

Hier sind _B1 und _Z1 Hilfsvariablen mit den Wertebereichen _B1 $\in \{0,1\}$ (B wie boolean) und _Z1 $\in \mathbf{Z}$.

Mathematica antwortet so:

```
In[1]:= Solve[Sin[x]==1/2,x]

Solve::ifun: Inverse functions are being used by Solve, so some
     solutions may not be found.

                Pi
Out[1]= {{x -> --}}
                6
```

Es gibt keine Möglichkeit, Mathematica zu bewegen, die Lösungsschar in parametrisierter Form auszugeben. Im Handbuch wird dies wie folgt begründet ([27, p. 794]):

> *Obwohl sich alle Lösungen dieser speziellen Gleichung einfach parametrisieren lassen, führen die meisten derartigen Gleichungen auf schwierig darzustellende Lösungsmengen. Betrachtet man Systeme trigonometrischer Gleichungen, so sind für eine Parameterdarstellung diophantische Gleichungssysteme zu lösen, was im allgemeinen Fall als algorithmisch nicht lösbar bekannt ist.*

Wir stoßen mit unserer einfachen Frage also unvermutet an prinzipielle Grenzen der Mathematik, die im betrachteten Beispiel allerdings noch nicht erreicht sind.

Oft haben wir nicht nur mit der Frage der Vollständigkeit der angegebenen Lösungsmenge zu kämpfen, sondern auch mit unterschiedlichen Darstellungen ein und des selben Ergebnisses, die sich aus unterschiedlichen Lösungswegen ergeben. Betrachten wir etwa die Aufgabe

```
solve(sin(x)+cos(x)=1/2,x);
```

Eine mögliche Umformung, die uns die vollständige Lösungsmenge liefert, wäre

$$\sin(x) + \cos(x) = \sin(x) + \sin(\frac{\pi}{2} - x) = 2\sin(\frac{\pi}{4})\cos(x - \frac{\pi}{4}),$$

womit die Gleichung zu $\cos(x - \frac{\pi}{4}) = \frac{1}{2\sqrt{2}}$ umgeformt wurde. Als Ergebnis erhalten wir

$$L = \left\{\frac{\pi}{4} + \arccos(\frac{1}{2\sqrt{2}}) + 2k\pi,\ \frac{\pi}{4} - \arccos(\frac{1}{2\sqrt{2}}) + 2k\pi \;\Big|\; k \in \mathbf{Z}\right\}.$$

MuPADs Antwort lautet

```
s:=solve(sin(x)+cos(x)=1/2,x);
```

$$\Big\{2\arctan\left(\tfrac{\sqrt{7}}{3} + 2/3\right), 2\pi + 2\arctan\left(\tfrac{\sqrt{7}}{3} + 2/3\right), -2\pi + 2\arctan\left(\tfrac{\sqrt{7}}{3} + 2/3\right), \ldots,$$
$$2\arctan\left(-\tfrac{\sqrt{7}}{3} + 2/3\right), 2\pi + 2\arctan\left(\tfrac{-\sqrt{7}}{3} + 2/3\right), -2\pi + 2\arctan\left(-\tfrac{\sqrt{7}}{3} + 2/3\right), \ldots\Big\}$$

während MAPLE die Antwort

$$\arctan(\frac{1/4 - 1/4\sqrt{7}}{1/4 + 1/4\sqrt{7}}), \arctan(\frac{1/4 + 1/4\sqrt{7}}{1/4 - 1/4\sqrt{7}}) + \pi$$

liefert.

Durch Rationalmachen der Nenner ist es noch einfach, die Äquivalenz der Lösungen von MAPLE und MUPAD zu erkennen. Ein Vergleich mit der von uns gefundenen Lösung ist dagegen schwierig. Versuchen wir wenigstens die näherungsweise Auswertung einiger Elemente beider Lösungsmengen:

```
sort(map([op(s,1)[1..5]],float));
```

$$[-10.57154324, -4.288357941, 1.994827366, 8.278012673, 14.56119798]$$

```
[float(PI/4+acos(1/(2*sqrt(2)))+2*k*PI) $ k=-2..2];
```

$$[-10.57154324, -4.288357941, 1.994827366, 8.278012673, 14.56119798]$$

An dieser Stelle wird bereits deutlich, dass die Komplexität eines CAS es oftmals notwendig macht, Ergebnisse in einem gegenüber dem Taschenrechnereinsatz vollkommen neuen Umfang nach- und umzuinterpretieren, um sie an die eigenen Erwartungen an deren Gestalt anzupassen.

1.5 Erweiterbarkeit von Computeralgebrasystemen

Ein weiterer Aspekt, der für den Einsatz eines CAS als Expertensystem wichtig ist, besteht in der

$$\boxed{\text{Möglichkeit der Erweiterung seiner mathematischen Fähigkeiten.}}$$

Die zur Verfügung stehende Programmiersprache kann nicht nur zur Ablaufsteuerung des Interpreters verwendet werden, sondern auch (in unterschiedlichem Umfang), um eigene Funktionen und ganze Pakete zu definieren.

Betrachten wir etwa die Vereinfachungen, die sich bei der Untersuchung des rationalen Ausdrucks u_n ergeben haben. Die dabei entstandenen Polynome, wie übrigens u_n auch, ändern sich bei Vertauschungen der Variablen nicht. Solche Ausdrücke nennt man deshalb *symmetrisch*. Insbesondere spielen symmetrische Polynome in vielen Anwendungen eine wichtige Rolle. Nehmen wir an, dass wir solche symmetrischen Polynome untersuchen wollen, diese Funktionalität aber vom System nicht gegeben wird.

Dazu erst einige Definitionen. Sei $X = (x_1, \ldots, x_n)$ eine endliche Menge von Variablen. Die bekanntesten symmetrischen Polynome sind die *elementarsymmetrischen Polynome* $e_k(X)$, die alle verschiedenen Terme aus k paarweise verschiedenen Faktoren x_i aufsummieren, die *Potenzsummen* $p_k(X) = \sum_i x_i^k$ und die *vollen symmetrischen Polynome* $h_k(X)$, die *alle* Terme in den gegebenen Variablen vom Grad k aufsummieren. So gilt etwa für $n = 4$

$$\begin{aligned}
e_2 &= x_1x_2 + x_1x_3 + x_1x_4 + x_2x_3 + x_2x_4 + x_3x_4 \\
p_2 &= {x_1}^2 + {x_2}^2 + {x_3}^2 + {x_4}^2 \\
h_2 &= x_1x_2 + x_1x_3 + x_1x_4 + x_2x_3 + x_2x_4 + x_3x_4 + {x_1}^2 + {x_2}^2 + {x_3}^2 + {x_4}^2
\end{aligned}$$

Wir wollen das System MUPAD so erweitern, dass durch Funktionen `e(d,vars)`, `p(d,vars)` und `h(d,vars)` zu einer natürlichen Zahl d und einer Liste $vars$ von Variablen diese Polynome erzeugt werden können. Statt der angegebenen Definition wollen wir dazu die rekursiven Relationen

$$e(d, (x_1, \ldots, x_n)) = e(d, (x_2, \ldots, x_n)) + x_1 * e(d-1, (x_2, \ldots, x_n))$$

und

$$h(d, (x_1, \ldots, x_n)) = h(d, (x_2, \ldots, x_n)) + x_1 * h(d-1, (x_1, \ldots, x_n))$$

verwenden. Von der Richtigkeit dieser Formeln überzeugt man sich schnell, wenn man die in der Summe auftretenden Terme in zwei Gruppen einteilt, wobei die erste Gruppe x_1 nicht enthält, die Teilsumme also gerade das entsprechende symmetrische Polynom in $(x_2, \ldots, x_n)$ ist. In der zweiten Gruppe kann man x_1 ausklammern.

Die entsprechenden Funktionsdefinitionen in MuPAD lauten (ohne Typprüfungen der Eingabeparameter)

```
e:=proc(d,vars) local u; begin
  if nops(vars)<d then 0
  elif d=0 then 1
  else u:=[op(vars,2..nops(vars))];
      expand(e(d,u)+op(vars,1)*e(d-1,u))
end_if end_proc;

h:=proc(d,vars) local u; begin
  if d=0 then 1
  elif nops(vars)=1 then op(vars,1)^d
  else u:=[op(vars,2..nops(vars))];
      expand(h(d,u)+op(vars,1)*h(d-1,vars))
end_if end_proc;

p:=proc(d,vars) begin
  _plus(map(op(vars),func(x^d,x)))
end_proc;
```

Wir sehen an diesem kleinen Beispiel bereits, dass die komplexen Datenstrukturen, die in symbolischen Rechnungen auf natürliche Weise auftreten, den Einsatz verschiedener Programmierparadigmen und -praktiken ermöglichen.

Die ersten beiden Blöcke ergänzen die jeweilige Rekursionsrelation durch geeignete Abbruchbedingungen zu einer korrekten Funktionsdefinition für e_d und h_d. Im dritten Block werden intensiv verschiedene Operationen auf Listen in einem funktionalen Programmierstil verwendet. Für

```
vars:=[x,y,z];
```

erhalten wir damit

```
h(2,vars);
```

$$xy + xz + yz + x^2 + y^2 + z^2$$

```
e(3,vars);
```

$$xyz$$

```
h(3,vars);
```

$$xyz + x^3 + y^3 + z^3 + xy^2 + x^2y + xz^2 + x^2z + yz^2 + y^2z$$

Ein Vergleich mit unseren weiter oben vorgenommenen Vereinfachungen zeigt, dass die rationale Funktion u_d offensichtlich gerade mit dem vollen symmetrischen Polynom h_{d-2} zusammenfällt.

Kapitel 2

Die Stellung des symbolischen Rechnens im Wissenschaftsgebäude

In diesem Kapitel wollen wir die Konsequenzen für Wissenschaft und Gesellschaft näher beleuchten, die sich aus der Möglichkeit ergeben, verschiedene Kalküle der Wissenschaft in Computerprogramme zu gießen und sie auf diese Weise einem weiten Anwenderkreis als Black- oder Grey-Box-Verfahren zur Verfügung zu stellen.

Vorab sei bemerkt, dass im deutschsprachigen Raum die Begriffe "symbolisches Rechnen" und "Computeralgebra" gewöhnlich alternativ zur Kennzeichnung des Fachgebiets, um das es in diesem Kurs geht, verwendet werden. In diesem Kapitel werde ich den Begriff "symbolisches Rechnen" (wie auch bei der Bezeichnung des Kurses insgesamt) verwenden, wenn ich stärker auf die Potenzen des Fachgebiets fokussiere, den Begriff "Computeralgebra" dagegen, wenn ich den aktuellen Entwicklungsstand des Fachgebiets stärker ins Auge fasse.

2.1 Zur Genese von Wissenschaft im Industriezeitalter

Sehen wir in einem Nachschlagewerk wie etwa der Brockhaus-Enzyklopädie [1] unter dem Stichwort "Wissenschaft" nach, so erfahren wir von den vielfältigen Aspekten, die sich in diesem Begriff bündeln.

Zunächst sind Stellung und Selbstverständnis von Wissenschaft zu nennen, die im Laufe der Zeit eine deutliche Wandlung erfahren haben. Noch im Mittelalter war die Wissenschaft "das Privileg meist wohlhabender, oft adliger Privatgelehrter" (ebenda, S. 278). Eine zentrale Rolle bei der Weitergabe und Erhaltung von Wissen kam den Klostern und ihren Schulen zu. Im Großen und Ganzen hatte die Beschäftigung mit Wissenschaft zu jener Zeit einen eher elitären Anstrich. Im Alltag spielten wissenschaftliche Kenntnisse eine absolut untergeordnete Rolle, ja selbst (aus heutiger Sicht elementare) Grundfertigkeiten wie Lesen, Schreiben und Rechnen waren kaum verbreitet.

Das ändert sich grundlegend mit dem Aufbruch ins Industriezeitalter. Neben einem paradigmatischen Bruch in der Wissenschaft selbst (siehe ebenda, S. 279) beginnt Wissenschaft auch im Alltag eine wichtigere Rolle einzunehmen; abzulesen etwa in der Einrichtung von Volksschulen, die die Fertigkeiten des Lesens, Schreibens und Rechnens verbreiten.

Ursache für diese veränderte Stellung von Wissenschaft sind zweifelsohne die gewachsenen Anforderungen, die ein industriell organisierter Arbeitsprozess sowohl an die beteiligten Akteure als auch an die geistige Durchdringung der Prozesse selbst stellt. Wissenschaftliche Bemühungen werden nach [1] nunmehr stärker auf die Fragen des "Wie?" und "Wodurch?", also auf funktionale und kausale Erklärungen der Phänomene ausgerichtet. Ein solches Verständnis ermöglicht erst das “Eingreifenkönnen und Beherrschen natürlicher Prozesse und Dinge” (ebenda, S.278). *Wissenschaftliche Rationalität* wird damit zum beherrschenden Wissenstypus, wenigstens im Bereich der Natur- und Technikwissenschaften, denen wir uns im Weiteren ausschließlich zuwenden werden.

Ein solcher Rationalitätsbegriff prägt denn auch das heutige Selbstverständnis der einzelnen Naturwissenschaften (Physik, Chemie, Biologie, ...) als Fachwissenschaften: sie haben als Ziel, in der Natur ablaufende Prozesse adäquat zu beschreiben und damit Modellvorstellungen zu entwickeln, auf deren Basis man Vorhersagen über diese Prozesse treffen oder sie sogar bewusst ausnutzen oder beeinflussen kann. Letzteres ist mit leicht anderer Schwerpunktsetzung auch Gegenstand der Technikwissenschaften.

Die “wissenschaftliche Strenge”, die für eine solche Rationalität an den Tag zu legen ist, unterliegt fachübergreifenden Standards. Die Existenz derartiger Standards hat ihre Ursache nur zum Teil im gemeinsamen Ursprung der Einzelwissenschaften. Eine wesentlich wichtigere Quelle liegt in der gemeinsamen Methodologie und dem dabei verwendeten erkenntnistheoretischen Instrumentarium:

- aus einer Fülle von experimentell gewonnenem Datenmaterial werden *Regelmäßigkeiten* herausgefiltert,
- diese in *Hypothesen* mit dem bisherigen Kenntnisstand verbunden und verdichtet und
- im Zuge weiterer experimenteller Verfikation zu neuen *Theorien* zusammengefasst.

Für die praktische Anwendung dieser neuen Theorien ist schließlich deren

- Aufbereitung in einem handhabbaren *Kalkül* ausschlaggebend[1], der zugleich
- die Basis für die Gewinnung neuen Datenmaterials auf der nächst höheren Abstraktionsebene bildet.

Eine solche in Richtung zunehmender Abstraktion weisende Erkenntnisspirale ist typisch für die “reinen” Wissenschaften. Um Wissenschaften im Zuge zunehmender Industrialisierung produktiv werden zu lassen, spielt die Anwendbarkeit und Anwendung theoretischen Wissens auf die gesellschaftliche Praxis eine ebenso wichtige Rolle. Diese Domäne der “angewandten” und Technik- oder Ingenieurwissenschaften folgt einem anderen erkenntnistheoretischen Paradigma:

- Reale Prozesse werden mit Hilfe eines geeigneten Kalküls *simuliert*.
- Die Simulation wird auf dem Hintergrund der verwendeten Theorie durch Analyse zu einem *Modell* verdichtet.
- Das Modell wird experimentell überprüft (und gegebenenfalls weiter verfeinert)
- Die gewonnenen Erkenntnisse werden in die Praxis implementiert.

In diesem Kreislauf spielen fertige Theorien und konkrete, bereits entwickelte Kalküle (nicht nur der Mathematik) eine zentrale Rolle.

Übergreifende Gesetzmäßigkeiten dieser Erkenntnisprozesse sind Gegenstand von *Querschnittswissenschaften*, von denen hier vor allem Philosophie, Mathematik und inzwischen auch die Informatik zu nennen sind.

Während die Philosophie die Denk- und Abstraktionsprozesse in ihrer Allgemeinheit zum Gegenstand hat, befasst sich die Mathematik mit übergreifenden Gesetzmäßigkeiten, die beim Quantifizieren von Phänomenen auftreten. Quelle und Target dieser Bemühungen sind die entsprechenden logischen Strukturen der Einzelwissenschaften, die oft erst durch die Anstrengungen der Mathematik eine streng deduktiven Ansprüchen genügende Konsistenz erhalten.

[1] Buchberger [3, S. 808] spricht in diesem Zusammenhang von der “Trivialisierung” einer Problemklasse (der symbolischen Mathematik).

Die Mathematik leistet so einen unverzichtbaren und eigenständigen Beitrag für die methodische Fundierung der Einzelwissenschaften, ohne welchen letztere nur wenig über ein empirisches Verständnis ihres Gegenstands hinauskommen würden. Mathematik und mathematischen Methoden kommt damit besonders in der Phase der Hypothesen- und Theoriebildung, aber auch bei der Modellierung und Analyse realer Prozesse, ein wichtiger Platz für die Leistungsfähigkeit und argumentative Tiefe einzelwissenschaftlicher Erkenntnisprozesse zu. Sie ist außerdem die Grundlage einzelwissenschaftlicher Kalküle, egal, ob diese Quantenphysik, Elektronik, Statik oder Reaktionskinetik heißen. Mathematik ist in diesem Sinne die "lingua franca" der Wissenschaft, was Marx zu der Bemerkung veranlasste, dass "sich eine Wissenschaft erst dann als entwickelt betrachten könne, wenn sie dahin gelangt sei, sich der Mathematik zu bedienen".

Im Gegensatz zu spezielleren Kenntnissen aus einzelnen Bereichen der Natur- oder Ingenieurwissenschaften sind mathematische Kenntnisse und Fertigkeiten damit in unserer technisierten Welt nicht nur in breiterem Umfang notwendig, sondern werden auch an verschiedenen Stellen des (Berufs-)Lebens selbst bei Facharbeitern oder vergleichbaren Qualifikationen schlichtweg vorausgesetzt. Eine gewisse "mathematische Kultur", die über einfache Rechenfertigkeiten hinausgeht, ist damit heute für eine qualifizierte Teilhabe am sozialen Leben unumgänglich.

Jedoch ist nicht nur der Einzelne auf solche Kenntnisse angewiesen, sondern auch die Gesellschaft als Ganzes. Denn erst eine solche "Kultur des Denkens" sichert die Fähigkeit, innerhalb der Gesellschaft auf einem Niveau zu kommunizieren, wie es für die Beherrschung der sozialen Prozesse notwendig ist, die sich aus der immer komplexeren technologischen Basis ergeben. Unter diesem Blickwinkel mag es nicht weiter verwundern, dass der Teil des durch die Mathematik entwickelten methodischen und begrifflichen Rüstzeugs, der inzwischen in die Allgemeinbildung Einzug gehalten hat, stetig wächst.

Obwohl es immer wieder Diskussionen über die Angemessenheit solcher Elemente im Schulunterricht gibt, zeigt sich im Lichte der TIMMS-Studien der letzten Jahre, die die mathematischen Fertigkeiten von Schülern in verschiedenen Ländern vergleicht, dass die allgemeine mathematische Kultur, die die Schule in Deutschland derzeit vermittelt, eher als mittelmäßig einzustufen ist.

Mit der allgegenwärtigen Verfügbarkeit leistungsfähiger Rechentechnik wird diese "Verwissenschaftlichung" gesellschaftlicher Zusammenhänge auf eine qualitativ neue Stufe gehoben. Viele, auch umfangreichere Kalküle können nun mechanisiert oder sogar automatisiert werden und stehen damit für einen breiteren Einsatz zur Verfügung, womit sich zugleich die Reichweite wissenschaftlicher Gedankenführung für einen weiten Kreis von Anwendungen deutlich erhöht. Neben Pflege, Weiterentwicklung und Vermittlung entsprechender *Denk-Kalküle*, dem traditionellen Gegenstand mathematischer Bildung, tritt damit eine weitere Querschnittswissenschaft, die die Erstellung, Pflege, Nutzungsunterweisung und Einbettung für solche technikbasierte Hilfsmittel geistiger Arbeit, kurz, eine sich neu herausbildende "technologische Seite des Denkens", zum Gegenstand hat. Ein solches Verständnis von Informatik[2] lässt Raum für eine

> weitergehende Symbiose von Kalkül und Technologie als Gegenstand eines Faches zwischen Mathematik und Informatik,

dem Grabmeier in [7] den provisorischen Namen "Computermathematik" gegeben hat. Das Symbolische Rechnen ist ein wesentlicher Teil dieses Gebiets.

2.2 Die Erkenntnisspirale der Mathematik

Auch die Mathematik entwickelt sich in ähnlichen Erkenntnisspiralen wie oben für die Naturwissenschaften beschrieben. Das findet seinen Niederschlag nicht zuletzt im Schulcurriculum, wo der Abstraktionsgrad der vermittelten mathematischen Begriffe, Denk- und Arbeitsweisen von

[2] Gängige Definitionen des Gegenstands der Informatik fokussieren stärker auf eine einzelwissenschaftliche Betrachtung, etwa als "Wissenschaft von der systematischen Verarbeitung von Informationen, besonders der automatischen Verarbeitung mit Digitalrechnern" ([5]).

Schuljahr zu Schuljahr wächst. Die Abstraktionsniveau klettert von der Beherrschung der Grundrechenarten über Variablenansätze in Textaufgaben und Lösungsverfahren für einfache lineare und nichtlineare Probleme bis hin zu einem ersten Kontakt mit Begriffen der Differential- und Integralrechnung. Dieses Curriculum zeichnet damit in groben Zügen die wichtigsten Abstraktionsschritte nach, die die Mathematik als Wissenschaft in den letzten Jahrhunderten selbst gegangen ist und die der Logik ihres inneren Aufbaus entsprechen.

Eine solche Zunahme des Abstraktionsniveaus fällt allerdings nicht, wie gelegentlich im Schulunterricht, vom Himmel, sondern ist das Resultat eben dieser Erkenntnisspirale, die die Mathematik wie jede andere (ernstzunehmende) Wissenschaft durchläuft: In einer Fülle von *Experimenten* werden *Regelmäßigkeiten* aufgedeckt, die im Weiteren zu *Gesetzmäßigkeiten* verdichtet werden und schließlich in weiter ausgebauter und systematisierter Form als *Kalkül* für neue Experimente zur Verfügung stehen.

Die Durchführung von Experimenten setzt die Beherrschung des aktuellen Kalküls voraus, die Aufdeckung von Regelmäßigkeiten im so gewonnenen Erfahrungsmaterial zusätzlich eine geschulte Beobachtungsgabe, gepaart mit einem guten Schuss Intuition. Will man die erkannten Regelmäßigkeiten allerdings zu Gesetzmäßigkeiten verdichten, reichen Fertigkeiten im Umgang mit dem aktuellen Kalkül nicht mehr aus, sondern man muss in der Lage sein, *über diesen Kalkül* selbst zu reflektieren. Eine solche Zunahme des Abstraktionsniveaus ist oft damit verbunden, dass man einzelne Elemente des aktuellen Kalküls in Sinneinheiten zusammenfasst und mit diesen als Ganzes operiert, wofür sich symbolische Methoden anbieten.

Erkannte Gesetzmäßigkeiten geben einen Ursache-Wirkungs-Zusammenhang wider, der im Weiteren natürlich dahingehend produktiv werden soll, dass man gezielt Wirkungen durch Erzeugung der entsprechenden Ursachen hervorrufen möchte. Hierfür ist es notwendig, die gefundenen Gesetzmäßigkeiten und die Argumente zu deren Fundierung zu einem neuen Kalkül zu verdichten, womit schließlich ein voller Zyklus der Erkenntnisspirale durchlaufen ist.

Wir wollen die hier beschriebene Erkenntnisspirale der Mathematik einige Windungen weit verfolgen. Grundstock aller mathematischer Fertigkeiten sind zuerst einmal *Rechenfertigkeiten*, wie sie seinerzeit von Rechenmeistern wie Adam Ries verbreitet wurden und heute Gegenstand des Grundschul-Unterrichts sind. Entsprechende Fertigkeiten, deren weite Verbreitung im 18. und 19. Jahrhundert die Herausbildung unserer modernen Gesellschaft erst ermöglichte, verlieren heute scheinbar an Bedeutung, denn jeder Taschenrechner und Computer beherrscht dieses Gebiet ebenfalls und sogar zuverlässiger und schneller als der Mensch.

So kann man etwa, statt dies per Hand zu tun, die Fertigkeiten eines Taschenrechners im Umgang mit Dezimalbrüchen zum Berechnen von Nullstellen eines Polynoms, etwa mit dem Newtonverfahren, nutzen. Wir wollen dies tun, um eine größere Menge von Daten zu erzeugen, um eine Beziehung zwischen den Nullstellen und den Koeffizienten eines Polynoms aufzuspüren. Je nach Komfort, den uns der jeweilige Taschenrechner bietet, müssen wir für die Berechnung jeder solchen Nullstelle mehr oder weniger Aufwand treiben. Auf moderneren Taschenrechnern reicht oft ein einziger Knopfdruck pro Nullstelle. Auf einem Computer könnten wir sogar noch komfortabler die ganze Problemstellung in ein Programm gießen und uns die entsprechenden Ergebnisse gleich als Tabelle ausgeben lassen. Abgesehen von diesem (für praktische Zwecke nicht unwesentlichem) Komfort sind die dafür notwendigen Rechenfertigkeiten allerdings keine anderen als bei einer Handrechnung. Wir wollen diese Rechnungen deshalb mit Maple simulieren, wobei wir die Koeffizienten des entsprechenden Polynoms zweiten Grades jeweils mit der Summe und dem Produkt seiner Nullstellen vergleichen:

```
print('Nullstellen', 'a','b','Summe','Produkt');
for a from 2.0 to 4.0 do for b from -12.0 to -9.0 do
     u:=solve(x^2+a*x+b=0,x);
     print([u], a,b, u[1]+u[2], u[1]*u[2])
od od;
```

Nullstellen	a	b	Summe	Produkt
[-4.605551275, 2.605551275]	2.0	-12.0	-2.000000000	-12.00000000
[-4.464101615, 2.464101615]	2.0	-11.0	-2.000000000	-11.00000000
[-4.316624790, 2.316624790]	2.0	-10.0	-2.000000000	-9.99999999
[-4.162277660, 2.162277660]	2.0	-9.0	-2.000000000	-8.99999999
[-5.274917218, 2.274917218]	3.0	-12.0	-3.000000000	-12.00000000
[-5.140054945, 2.140054945]	3.0	-11.0	-3.000000000	-11.00000000
[-5., 2.]	3.0	-10.0	-3.000000000	-10.00000000
[-4.854101966, 1.854101966]	3.0	-9.0	-3.000000000	-8.99999999
[-6., 2.]	4.0	-12.0	-4.000000000	-12.00000000
[-5.872983346, 1.872983346]	4.0	-11.0	-4.000000000	-11.00000000
[-5.741657387, 1.741657387]	4.0	-10.0	-4.000000000	-10.00000000
[-5.605551276, 1.605551276]	4.0	-9.0	-4.000000000	-9.00000000

Wir erkennen unschwer die bekannte Relation, dass für die Nullstellen x_1 und x_2 gerade $a = -(x_1 + x_2)$ und $b = x_1\,x_2$ gilt. Eine solche Liste von experimentellen Ergebnissen gibt eine gewisse Evidenz, dass ein solcher Zusammenhang besteht, liefert aber, so lang sie auch sein mag, noch keinen Beweis im streng deduktiven Sinn. Diesen wird man für den Fall eines Polynoms zweiten Grades erst erbringen können, wenn man in der Lage ist, die Nullstellen der *allgemeinen Gleichung 2. Grades*, also von

$$x^2 + a\,x + b = 0,$$

symbolisch durch die Koeffizienten a, b darzustellen, also die allgemeine Lösungsformel

$$x_1 = -1/2\,a + 1/2\,\sqrt{a^2 - 4\,b}, \quad x_2 = -1/2\,a - 1/2\,\sqrt{a^2 - 4\,b}$$

kennt. Eine solche Kenntnis geht natürlich weit über die Fähigkeit hinaus, die jeweiligen Nullstellen mit dem Newtonverfahren zu berechnen, ja selbst über die Zusammenhänge, die wir für die allgemeinen Nullstellen von Gleichungen höheren Grades weiter unten verwenden werden. Sie sind bereits Teil eines anderen Kalküls, mit dem man *über die Nullstellen* solcher Gleichungen sprechen kann.

Entsprechend einfach ist es auch, aus diesen Formeln einen *Beweis* für den behaupteten Zusammenhang herzuleiten, wenn man diesen neuen Kalkül, die *Variablenrechnung*, beherrscht. Mehr noch sind die entsprechenden Rechnungen mit einem der Formelmanipulation fähigen System *automatisch* ausführbar, wie in unserem Beispiel mit Maple:

```
a:='a': b:='b': u:=solve(x^2+a*x+b=0,x);
```

$$-1/2\,a + 1/2\,\sqrt{a^2 - 4\,b}, -1/2\,a - 1/2\,\sqrt{a^2 - 4\,b}$$

```
u[1]+u[2];
```

$$-a$$

```
expand(u[1]*u[2]);
```

$$b$$

Dieselben Untersuchungen für ein Polynom dritten Grades lassen einen ähnlichen Zusammenhang vermuten. Die entsprechenden numerischen Rechnungen sind wieder mit Maple simuliert, diesmal aber auf drei Stellen gekürzt:

```
Digits:=3;
print('Nullstellen', 'a','b','c','Summe','Produkt');
for a from 2.0 to 3.0 do
   for b from -12.0 to -11.0 do
   for c from 3.0 to 5.0 do
     u:=solve(x^3+a*x^2+b*x+c=0,x);
     print([u], a,b,c, u[1]+u[2]+u[3], u[1]*u[2]*u[3] )
od od od;
```

Nullstellen	a	b	c	Summe	Produkt
[-4.69, .263, 2.43]	2.0	-12.0	3.0	-2.00	-2.99
[-4.72, .359, 2.36]	2.0	-12.0	4.0	-2.00	-3.99
[-4.75, .460, 2.29]	2.0	-12.0	5.0	-2.00	-5.02
[-4.56, .290, 2.27]	2.0	-11.0	3.0	-2.00	-3.00
[-4.59, .398, 2.19]	2.0	-11.0	4.0	-2.00	-4.01
[-4.62, .515, 2.10]	2.0	-11.0	5.0	-2.01	-5.00
[-5.35, .270, 2.08]	3.0	-12.0	3.0	-3.00	-3.00
[2., -5.37, .373]	3.0	-12.0	4.0	-3.00	-3.99
[-5.40, .485, 1.91]	3.0	-12.0	5.0	-3.01	-5.00
[-5.22, .300, 1.92]	3.0	-11.0	3.0	-3.00	-3.01
[-5.24, .418, 1.83]	3.0	-11.0	4.0	-2.99	-4.01
[-5.27, .554, 1.71]	3.0	-11.0	5.0	-3.01	-4.99

Hier ist der Zusammenhang zur expliziten Lösungsformel zum Beweis der angetroffenen Regelmäßigkeit bereits schwieriger auszunutzen. Für Polynome höheren Grades, wo es solche Lösungsformeln (wenigstens in einer aus dem Schulunterricht gewohnten Form) gar nicht gibt, muss man schließlich tieferliegende Zusammenhänge zwischen den Nullstellen und den Koeffizienten ausnutzen, die sich nur in einer Sprache formulieren lassen, die *über Variablenrechnung* räsoniert: Ein Polynom

$$f := x^n + a_1\,x^{n-1} + a_2\,x^{n-2} + \ldots + a_{n-1}\,x + a_n$$

hat stets n komplexe Nullstellen $x_1, x_2, \ldots, x_n$, wenn man diese mit der entsprechenden Vielfachheit zählt, und es gilt

$$f = (x - x_1) \cdot (x - x_2) \cdot \ldots \cdot (x - x_n).$$

Nun ist es leicht, die Vietaschen Formeln allgemein herzuleiten. Man muss einfach diese Produktdarstellung ausmultiplizieren und die Koeffizienten mit denen in der Ausgangsdarstellung vergleichen.

Satz 2 *(Vietascher Wurzelsatz)*
Zwischen den Koeffizienten des Polynoms

$$f := x^n + a_1\,x^{n-1} + a_2\,x^{n-2} + \ldots + a_{n-1}\,x + a_n$$

und seinen Nullstellen $x_1, x_2, \ldots, x_n$ besteht folgender Zusammenhang:

$$a_i = (-1)^i\, e_i(x_1, x_2, \ldots, x_n), \quad i = 1, 2, \ldots, n,$$

wobei e_i das im letzten Kapitel eingeführte i-te elementarsymmetrische Polynom ist.

Evidenz für diesen Sachverhalt kann man sich wiederum mit einem Computerprogramm wie Maple verschaffen, das den Variablenkalkül, hier insbesondere das *Rechnen mit Polynomen*, beherrscht:

```
for n from 2 to 5 do print(collect(expand(mul(x-x.i,i=1..n)),x)) od;
```

$$x^2 + (-x_2 - x_1)\,x + x_1\,x_2$$

$$x^3 + (-x_1 - x_3 - x_2)\,x^2 + (x_1\,x_3 + x_1\,x_2 + x_2\,x_3)\,x - x_1\,x_2\,x_3$$

$$\begin{aligned}&x^4 + (-x_2 - x_1 - x_4 - x_3)\,x^3 + (x_1\,x_2 + x_2\,x_4 + x_2\,x_3 + x_1\,x_4 + x_1\,x_3 + x_3\,x_4)\,x^2\\ &\qquad + (-x_1\,x_2\,x_4 - x_1\,x_2\,x_3 - x_2\,x_3\,x_4 - x_1\,x_3\,x_4)\,x + x_1\,x_2\,x_3\,x_4\end{aligned}$$

$$\begin{aligned}&x^5 + (-x_1 - x_2 - x_5 - x_3 - x_4)\,x^4\\ &\qquad + (x_2\,x_5 + x_2\,x_4 + x_3\,x_4 + x_2\,x_3 + x_1\,x_2 + x_1\,x_4 + x_3\,x_5 + x_1\,x_5 + x_1\,x_3 + x_4\,x_5)\,x^3\\ &\qquad + \big(-x_1\,x_2\,x_3 - x_3\,x_4\,x_5 - x_2\,x_4\,x_5 - x_2\,x_3\,x_5 - x_1\,x_4\,x_5 - x_1\,x_3\,x_5\\ &\qquad\qquad - x_1\,x_2\,x_5 - x_1\,x_3\,x_4 - x_1\,x_2\,x_4 - x_2\,x_3\,x_4\big)x^2\\ &\qquad + (x_2\,x_3\,x_4\,x_5 + x_1\,x_2\,x_4\,x_5 + x_1\,x_2\,x_3\,x_5 + x_1\,x_2\,x_3\,x_4 + x_1\,x_3\,x_4\,x_5)\,x - x_1\,x_2\,x_3\,x_4\,x_5\end{aligned}$$

Auch diese (endliche) Liste von Ergebnissen ist natürlich noch kein Beweis des Vietaschen Wurzelsatzes. Um einen solchen zu erbringen müssen wir eine weitere Abstraktionsstufe erklimmen, auf der sich die (hoffentlich unendliche) Folge der beobachteten Regelmäßigkeiten einer (endlichen) mathematischen Argumentation erschließt. Die im letzten Kapitel hergeleitete rekursive Definition der elementarsymmetrischen Polynome zusammen mit einem Induktionsargument schließen die noch bestehende Lücke.

2.3 Symbolisches Rechnen und der Computer als Universalmaschine

Interessanterweise wiederholt sich die Entwicklung vom einfachen Kalkül der Arithmetik hin zu komplizierteren symbolischen Kalkülen in der Geschichte des Einsatzes des Computers als Hilfsmittel geistiger Arbeit. Historisch wurde das Wort Computer bekanntlich zuerst mit einer Maschine zur schnellen Ausführung numerischer Rechnungen verbunden. Nachdem dies in der Anfangszeit ebenfalls auf einfache arithmetische Operationen beschränkt war (und für Taschenrechner lange so beschränkt blieb), können auf entsprechend leistungsfähigen Maschinen heute auch kompliziertere Anwendungen wie das Berechnen numerischer Werte mathematischer Funktionen, die Approximation von Nullstellen gegebener Polynome oder von Eigenwerten gegebener Matrizen realisiert werden. Solche numerischen Verfahren spielen (derzeit) die zentrale Rolle in Anwendungen mathematischer Methoden auf Probleme aus Naturwissenschaft und Technik und bilden den Kern einer eigenen mathematischen Disziplin, des *Wissenschaftlichen Rechnens*[3].

All diesen Anwendungen ist gemein, dass sie zwar, unter Verwendung ausgefeilter Programmiersprachen, die Programmierfähigkeit eines Computers ausnutzen, sich letztlich aber allein auf das Rechnen mit (Computer)zahlen zurückführen lassen. Der Computer erscheint in ihnen stets als außerordentlich präzise und schnelle, im übrigen aber stupide *Rechen*maschine, als “number cruncher”.

Genau so kommt der Computer auch in vielen großen numerischen Simulationen praktischer Prozesse zum Einsatz, so dass ein Bild seiner Fähigkeiten entsteht, das sowohl aus innermathematischen als auch informatik-theoretischen Überlegungen heraus eher einer künstlichen Beschränkung seiner Einsatzmöglichkeiten gleichkommt. Zeigt uns doch die Berechenbarkeitstheorie in Gestalt der Church'schen These, dass der Computer eine *Universalmaschine* ist, die, mit einem geeigneten Programm versehen, prinzipiell in die Lage versetzt werden kann, jede nur denkbare algorithmische Tätigkeit auszuüben. Also insbesondere auch in der Lage sein sollte, Symbole und nicht nur Zahlen nach wohlbestimmten Regeln zu verarbeiten.

Dass ein Computer auch zu einer solchen Symbolverarbeitung fähig ist, war übrigens lange vor dem Bau des ersten “echten” (von-Neumann-)Rechners bekannt. Bereits Charles Babbage (1792

[3] Allerdings definiert sich Wissenschaftliches Rechnen normalerweise nicht über die verwendeten Kalküle, sondern in Abgrenzung zur “reinen Mathematik” über das zu Grunde liegende (und weiter oben bereits beschriebene) Anwendungsparadigma.

- 1838), der mit seiner "Analytical Engine" 1838 ein dem heutigen Computer ähnliches Konzept entwickelte, ohne es aber je realisieren zu können, hatte diese Fähigkeiten einer solchen Maschine im Blick. Seine Assistentin, Freundin und Mäzenin, Lady Lovelace, schreibt (zitiert nach [10, S. 1]) :

> Viele Menschen, die nicht mit entsprechenden mathematischen Studien vertraut sind, glauben, dass mit dem *Ziel* von Babbage's Analytical Engine, Ergebnisse in Zahlennotation auszugeben, auch deren *Inneres* arithmetisch-numerischer Natur sein müsse statt algebraisch-analytischer. Das ist ein Irrtum. Die Maschine kann ihre numerischen Eingaben genau so anordnen und kombinieren, als wären es Buchstaben oder andere allgemeine Symbole; und könnte sie sogar *in einer solchen Form ausgeben*, wenn nur entsprechende Vorkehrungen getroffen würden.

Ein solches Computerverständnis ist uns, im Gegensatz zu den Pionieren des Computer-Zeitalters, im Lichte von ASCII-Code und Textverarbeitungssystemen heute allgemein geläufig, wenigstens was den Computer als intelligente Schreibmaschine betrifft. Mit dem Siegeszug der Kleinrechentechnik in den letzten 20 Jahren entwickelte er sich dabei vom Spielzeug und der erweiterten Schreibmaschine hin zu einem unentbehrlichen Werkzeug der Büroorganisation, wobei vor allem seine Fähigkeit, (geschriebene) Information speichern, umordnen und verändern zu können, eine zentrale Rolle spielt. In diesem Anwendungssektor kommt also die Fähigkeit des Computers, *symbolische Information* verarbeiten zu können, bereits unmittelbar auch für den Umgang mit Daten zum Einsatz. Dabei verwischt sich, genau wie von Lady Lovelace vorausgesehen, durch die binäre Kodierung symbolischer Information die Grenze zwischen Zahlen und Zeichen, die zuerst so absolut schien.

Auf dieser Abstraktionsebene ist es auch möglich, die verschiedensten Nachschlagewerke und Formelsammlungen, also in symbolischer Form kodiertes Wissen, mit dem Computer aufzubereiten, in datenbankähnlichen Strukturen vorzuhalten und mit entsprechenden Textanalyseinstrumenten zu erschließen. Es wird sogar möglich, auf verschiedene Weise symbolisch kodierte Informationen in multimedialen Produkten zu verknüpfen, was das ungeheure innovative Potential dieser Entwicklungen verdeutlicht. In der Hand des Ingenieurs und Wissenschaftlers entwickelt sich damit der Computer zu einem sehr effektiven Instrument, das nicht nur den Rechenschieber, sondern auch zunehmend Formelsammlungen abzulösen in der Lage ist. Auf diesem Niveau handelt es sich allerdings noch immer um eine *syntaktische* Verarbeitung von Information, wo der Computer deren *Sinn* noch nicht in die Verarbeitung einzubeziehen vermag.

Aber auch der Einsatz des Computers zu *numerischen* Zwecken enthält bereits eine wichtige symbolische Komponente: Er hat das Programm für die Rechnungen in adäquater Form in seinen Speicher zu bringen und von dort wieder zu extrahieren. Diese Art symbolischer Information ist bereits *semantischer Art*, da die auf diese Weise dargestellten *Algorithmen* inhaltliche Aspekte der verarbeiteten Zahlengrößen erschließen. Dass dies vom Nutzer nicht in gebührender Form wahrgenommen wird, hängt in erster Linie mit der strikten Trennung von (numerischen) Daten und (symbolischem) Programm sowie der Betrachtung des Computers als virtuelle Maschine ("Das Programm macht der Programmierer, die Rechnung der Computer") zusammen.

Bringt man beide Ebenen, die Daten und die Programme, zusammen, ermöglicht also algorithmische Operationen auch auf symbolischen Daten, geht man einen großen Schritt in die Richtung, semantische Aspekte auch symbolischer Information einer automatischen Verarbeitung zu erschließen. Denken lernt der Computer damit allerdings nicht, denn auch die Algorithmik symbolischer Informationsverarbeitung benötigt zunächst den in menschlicher Vorleistung erdachten *Kalkül*, den der Computer dann in der Regel schneller und präziser als der Mensch auszuführen vermag.

In diesem Schnittpunkt moderner Entwicklungen befindet sich die *Computeralgebra*. Mit ihrem ausgeprägten Werkzeugcharakter und einer starken Anwendungsbezogenheit steht sie paradigmatisch dem (klassischen Gegenstand des) Wissenschaftlichen Rechnen nahe und wurde lange Zeit nur als Anhängsel dieser sich aus der Numerik heraus etablierten mathematischen Disziplin verstanden. Ihre Potenzen sind aber vielfältiger. Zunächst steht sie in einer Reihe mit anderen nichtnumerischen Applikationen einer "Mathematik mit dem Computer" wie z.B. Anwendungen der diskreten Mathematik (Kombinatorik, Graphentheorie) oder der diskreten Optimierung, die *endliche* Strukturen untersuchen, die sich *exakt* im Computer reproduzieren lassen. Der Unterschied zwischen "Wissenschaftlichem Rechnen" und "angewandter Mathematik" wird damit deutlicher.

Allerdings hat die Computeralgebra mathematische Strukturen zum Gegenstand, die syntaktisch endlich (und damit *exakt* im Computer darstellbar) sind, aber semantisch oft unendliche Strukturen repräsentieren. Sie kommt damit mathematischen Arbeitstechniken näher als die anderen bisher genannten Gebiete. Siehe [3] für weitergehende Überlegungen zur Abgrenzung des Gegenstands des symbolischen Rechnens von numerischer und diskreter Mathematik.

Neben konkreten Implementierungen wichtiger mathematischer Verfahren reicht die Bedeutung der Computeralgebra aber über den Bereich der algorithmischen Mathematik hinaus. Die Vielzahl mathematischer Verfahren, die in einem modernen CAS unter einer *einheitlichen* Oberfläche verfügbar sind, machen dieses zu einem $\boxed{\text{metamathematischen Werkzeug}}$ für Anwender, ähnlich den Numerikbibliotheken, die heute schon im Wissenschaftlichen Rechnen eine zentrale Rolle spielen.

In einer zu etablierenden "Computermathematik" ([7]) als Symbiose dieser Entwicklungen werden computergestützte numerische, diskrete und symbolische Methoden gleichberechtigt nebeneinander stehen und in praktischen Applikationen sich gegenseitig befruchtend ineinander greifen.

2.4 Numerische versus strukturelle Mathematik

Beim Versuch der Etablierung einer solchen "Computermathematik" stoßen allerdings paradigmatisch unterschiedliche Welten aufeinander: Während die symbolischen Methoden der Computeralgebra, wenigstens im Prinzip, streng mathematisch deduktiver Natur sind, haben numerische Verfahren meist approximativen Charakter und sind damit zunächst auf ein quantitatives Bild der Realität ausgerichtet. Wir wollen deshalb nun versuchen, wichtige Unterschiede zwischen numerischen und symbolischen Methoden herauszuarbeiten.

Eine solcher Vergleich ist auch deshalb interessant, weil die Erfahrungen, über die die meisten Leser mit automatischen Rechnungen auf dem Computer verfügen, gerade aus dem Gebiet der Numerik kommen. Auch der größte Teil der Rechnungen, die heutzutage auf großen Mainframes und Rechnerverbänden ausgeführt werden, sind numerischer Natur und erfahren bestenfalls noch eine Ergänzung durch Visualisierungstools, die sich auf eben solche Grundlagen stützen.

Den typischen Unterschied zwischen beiden Sichtweisen beschreibt Pavelle in [13] wie folgt:

> Betrachten wir den einfachen Ausdruck $\frac{3\pi^2}{\pi}$. Jeder Algebrastudent weiss, dass sich dieser Bruch zu 3π vereinfachen lässt, wenn man Zähler und Nenner durch π kürzt. Der numerische Wert von 3π kann interessieren, es kann aber auch sein, dass es ausreicht oder vielleicht sogar günstiger ist, diesen Ausdruck in seiner symbolischen, nichtnumerischen Form zu belassen. Mit einem nur auf numerische Operationen getrimmten Computer *muss* der Ausdruck $\frac{3\pi^2}{\pi}$ ausgewertet werden; wird dies mit einer Präzision von 10 Stellen ausgeführt, erhält man als Wert 9.424777958. Diese Zahl, abgesehen von der Tatsache, dass es sich dabei um eine eher uninformative Folge von Ziffern handelt, ist nicht dasselbe wie die Zahl, die man auf demselben Wege (d.h. durch Auswertung mit einer Genauigkeit von 10 Stellen) aus 3π bekommt. Letzteres liefert nämlich die Zahl 9.424777962, wobei die Differenz in den letzten beiden Stellen aus unumgänglichen Rundungsfehlern des Computers herrührt. Die Äquivalenz von $\frac{3\pi^2}{\pi}$ und 3π würde von solch einem Computer wohl nicht einmal festgestellt werden.

Der Verlust struktureller Information, der im Rahmen numerischer Verfahren durch die Abbildung der Überabzählbarkeit der reellen Welt auf die Endlichkeit des Computers unvermeidlich ist, führt zu dem Ergebnis, dass mit diesen Daten streng deduktiv basierte, also dem Wissenschaftlichkeitsanspruch einer "reinen" Wissenschaft genügende, Argumentationen nicht mehr möglich sind.

Symbolische Verfahren können dagegen ob der Endlichkeit ihrer Begriffswelt adäquat in die Endlichkeit eines Computers übertragen werden, ohne dadurch Teile ihrer strukturellen Aussagekraft einzubüßen.

Numerische Verfahren kann man einsetzen, um *praktisch relevante Zahlenwerte auszurechnen*, symbolische Verfahren dagegen auch, um *mathematische Sachverhalte zu beweisen.*

Mit diesen Argumenten soll die vielfältig unter Beweis gestellte Bedeutung numerischer Verfahren und Ergebnisse nicht in Abrede gestellt werden. Allerdings ergeben sich im Vergleich von symbolischen und numerischen Methoden eine Reihe von Vorteilen, die eine strukturelle Sicht auf mathematische Fragestellungen gegenüber einer rein quantitativen bietet. Pavelle ([13]) listet die folgenden auf:

- *Erstens* gestattet eine präventive symbolische Analyse es oftmals, einen Ausdruck effektiver numerisch auszuwerten.

  Dieser einmalige vorherige Aufwand spielt besonders für Funktionen, die mit vielen verschiedenen Parameterbelegungen auszuwerten sind, eine entscheidende Rolle. Nach [15] werden moderne CAS zu 90 % gerade zu diesem Zweck, d.h. zur Generierung effizienten Codes, eingesetzt.

- *Zweitens* sind die gewonnenen Aussagen durch das prinzipielle Vermeiden von numerischen Approximationen exakt in einem streng mathematisch-deduktiven Sinn. Damit werden Fehler- und Rundungsanalysen, die bei der Ergebnisverifikation numerischer Verfahren oftmals den Löwenanteil des Aufwands ausmachen, unnötig.

  Die Frage der Sicherung der Relevanz numerischer Simulationen ist eine zentrale Frage, die bei vielen praktischen Applikationen unbeantwortet im Raum stehen bleibt. Sie erfordern gewöhnlich umfangreiche Stabilitätsuntersuchungen, bei denen ihrerseits symbolische Methoden hilfreich sein können.

- Und *drittens* impliziert ein Ergebnis in einer symbolischen Form ein qualitativ tieferes Problemverständnis als es selbst durch eine große Menge rein numerischer Daten ausgedrückt werden kann.

  Zwar ermöglichen es (z.B. auf numerischen Verfahren basierende) Simulationen, größere Datenmengen zu erzeugen, die man nach *Regelmäßigkeiten* absuchen kann, jedoch erst eine (nur symbolisch mögliche) Analyse in einem streng deduktiven Verständnis von Mathematik vermag diese in *Gesetzmäßigkeiten* zu verwandeln, die unserem *Pool gesicherten Wissens* hinzugefügt werden können.

In *Theorien* fixierte symbolische Ergebnisse vermitteln einen wesentlich tieferen strukturellen Einblick in Zusammenhänge als Daten aus (rein numerischen) Simulationen, erfordern aber zugleich einen komplizierteren mathematischen Apparat. Sicher kann man längst nicht alles von praktischer Relevanz bis zu einer solchen strukturellen Sicht theoretisch aufarbeiten und verdichten. Sie ist und bleibt jedoch das eigentliche Ziel mathematischer (und nicht nur mathematischer) Erkenntnis. [13] zitiert in diesem Zusammenhang R.W.Hamming:

The purpose of computing is insight, not numbers.

Es gilt also, die Besonderheiten beider Gebiete (gegenseitig) ernst zu nehmen und zu einem neue Horizonte eröffnenden Zusammenspiel zu führen.

2.5 Was ist Computeralgebra ?

Was ist nun Computeralgebra? Oben hatten wir gesehen, dass sie eine spezielle Art von Symbolverarbeitung zum Gegenstand hat, in der, im Gegensatz zur Textverarbeitung, die Symbole mit

Inhalten, also einer *Semantik*, verbunden sind. Auch geht es bei der Verarbeitung um Manipulationen eben dieser Inhalte und nicht primär der Symbole selbst.

Von der Natur der Inhalte und der Form der Manipulationen her können wir den Gegenstand der Computeralgebra also in erster Näherung als

symbolisch-algebraische Manipulationen mathematischer Inhalte

bezeichnen.

Gehen wir eher von der syntaktischen Form aus, in der uns diese Inhalte entgegentreten, so lässt sich der Gegenstand grob als

Rechnen mit Symbolen, die mathematische Objekte repräsentieren

umreißen. Diese Objekte können neben ganzen, rationalen, reellen oder komplexen Zahlen (beliebiger Genauigkeit) auch algebraische Ausdrücke, Polynome, rationale Funktionen, Gleichungssysteme oder sogar noch abstraktere mathematische Objekte wie Gruppen, Ringe, Algebren und deren Elemente sein.

Das Adjektiv *symbolisch* bedeutet, dass das Ziel der mathematischen Problemstellung nicht die Suche nach numerischen Werten, sondern nach einer geschlossenen oder approximativen Formel im Sinne des deduktiven Mathematikverständnisses ist. *Algebraisch* bedeutet, dass eine *exakte* mathematische Ableitung aus den Ausgangsgrößen durchgeführt wird, anstatt näherungsweise Fließkommaarithmetik einzusetzen. Beispiele für solche algebraisch-symbolischen Umformungen sind die Polynomfaktorisierung, die (unbestimmte) Differentiation und Integration, die Reihenentwicklung von Funktionen, analytische Lösungen von Differentialgleichungen, exakte Lösungen polynomialer Gleichungssysteme oder die Simplifikation mathematischer Formeln.

Der Begriff "algebraisch" beinhaltet dabei keine Einschränkung auf spezielle Teilgebiete der Mathematik, sondern eine der verwendeten Methoden. Diese sind als mathematische Schlussweise weit verbreitet, denn auch Anwendungen aus der Analysis, die, wie z.B. Grenzwert- oder Integralbegriff, per definitionem Näherungsprozesse untersuchen, verwenden in ihrem eigenen Kalkül solche algebraischen Umformungen. Wir hatten dies bereits am Unterschied zwischen der Ableitungsdefinition und dem Vorgehen bei der praktischen Bestimmung einer solchen Ableitung gesehen. Vgl. auch [11].

In den letzten 30 Jahren hat man große Fortschritte in der theoretischen Fundierung dieser symbolischen Methoden erreicht. Es sind Werkzeuge entstanden, die es erlauben, ein großes Stück mathematischen und zunehmend auch naturwissenschaftlichen und ingenieurtechnischen Knowhows Anwendern als black box unmittelbar zugänglich zu machen.

Historisch stand und steht dabei das effiziente Rechnen mit aus mathematischen Symbolen aufgebauten polynomialen und rationalen Ausdrücken am Ausgangspunkt, wobei die wichtigste Problemstellung im Erkennen semantischer Gleichwertigkeit syntaktisch unterschiedlicher Ausdrücke liegt. Die hierbei verwendeten Methoden unterscheiden sich oftmals sehr von der durch "mathematische Intuition" gelenkten und stärker heuristisch geprägten Schlussweise des Menschen und greifen verstärkt die Entwicklungslinien der konstruktiven Mathematik mit ihrer vorerst letzten Blüte in den 20er Jahren dieses Jahrhunderts wieder auf, vgl. R.Loos in [11].

In diesem Sinne beschreibt J.Grabmeier in [7], auf R.Loos zurückgehend,

> *Computeralgebra als den Teil der Informatik und Mathematik, der algebraische Algorithmen entwickelt, analysiert, implementiert und anwendet.*

"Algebraisch" bezeichnet dabei, wie oben erläutert, eine Methode und kein mathematisches Teilgebiet, so dass z.B. auch analytische Algorithmen, insofern sie auf algebraische Umformungen hinauslaufen, erfasst werden.

Buchberger kommt in [3, S. 799] ganz ohne dieses Adjektiv aus:

> *Symbolisches Rechnen* (dort als Synonym für Computeralgebra, vgl. ebenda S. 800 – HGG) *ist der Teil der algorithmischen Mathematik, der sich mit dem exakten algorithmischen Lösen von Problemen in abstrakten mathematischen Strukturen befasst.*

Im Weiteren unterstreicht Buchberger die Bedeutung der Algebraisierung und Algorithmisierung mathematischer Fragestellungen, um sie einer computeralgebraischen Behandlung im engeren Sinne zugänglich zu machen, und schlägt diesen Aufwand dem symbolischen Rechnen zu. Die Übergänge zu anderen Gebieten der algorithmischen Mathematik sind fließend, so dass wir uns im Folgenden stärker auf die Symbiose zwischen Mathematik und Computer konzentrieren wollen, die im symbolischen Rechnen zum Ausdruck kommt und das Wort *Computeralgebra* in diesem Sinne verwenden wollen.

Mit Blick auf die zunehmende Kompliziertheit der entstehenden Werkzeuge sind dafür obige Definitionen sicher noch zu erweitern um

> *die Entwicklung des zu Implementierung und Management solcher Systeme notwendigen informatik-theoretischen und -praktischen Instrumentariums.*

Die Computeralgebra befindet sich damit an der Schnittstelle zentraler Entwicklungen verschiedener Gebiete sowohl der Mathematik als auch der Informatik.

In ihrem Computeralgebra-Report [8] definiert die Fachgruppe Computeralgebra der Gesellschaft für Informatik (GI) ihr eigenes Fachgebiet etwas ausführlicher so:

> *Die Computeralgebra ist ein Wissenschaftsgebiet, das sich mit Methoden zum Lösen mathematisch formulierter Probleme durch symbolische Algorithmen und deren Umsetzung in Soft- und Hardware beschäftigt. Sie beruht auf der exakten endlichen Darstellung endlicher oder unendlicher mathematischer Objekte und Strukturen und ermöglicht deren symbolische und formelmäßige Behandlung durch eine Maschine. Strukturelles mathematisches Wissen wird dabei sowohl beim Entwurf als auch bei der Verifikation und Aufwandsanalyse der betreffenden Algorithmen verwendet. Die Computeralgebra kann damit wirkungsvoll eingesetzt werden bei der Lösung von mathematisch modellierten Fragestellungen in zum Teil sehr verschiedenen Gebieten der Informatik und Mathematik sowie in den Natur- und Ingenieurwissenschaften.*

In diesem Spannungsfeld zwischen Mathematik und Informatik, das R.Loos in [11] etwas pessimistisch als einen Bereich umreisst, der solche Probleme zum Gegenstand hat,

> die zu rechenintensiv sind, um von Mathematikern anerkannt zu werden und zu mathematisch, um als Informatik durchzugehen,

findet die Computeralgebra zunehmend ihren eigenen Platz. Sie nimmt damit zugleich eine wichtige Stellung für Entwicklungsimpulse aus beiden Gebieten ein. So mag es nicht verwundern, dass die großen Durchbrüche der letzten Jahre sowohl in der Mathematik als auch in der Informatik die von der Computeralgebra produzierten Werkzeuge wesentlich beeinflusst haben und umgekehrt.

Mit der Computerisierung mathematischer Verfahren erweitert sich zugleich der Kognitionsbereich "durchschnittlicher" Mathematiker, so dass mathematische Ergebnisse, die noch vor einigen Jahrzehnten intellektuelle Vorposten darstellten, zum Allgemeingut werden. Eines der kuriosen Beispiele, das in diesem Zusammenhang immer wieder erwähnt wird, ist die Wiederholung der Mondbahnberechnungen, für die der französische Astronom C.Delaunay im 19. Jahrhundert 20 Jahre seines Lebens opferte und die mit modernen Computern in wenigen Stunden nachvollziehbar sind, vgl. [13]. Die mathematische Glaubwürdigkeit des Computerergebnisses ist zudem unvergleichlich höher, wie in der anerkennenden Feststellung zum Ausdruck kommt, dass sich Delaunay nur an drei Stellen in Termen untergeordneter Bedeutung verrechnet habe (und nicht umgekehrt der Computer).

Zusammenfassend lässt sich sagen, dass die Computeralgebra nicht eine weitere Computeranwendung schlechthin unter vielen anderen ist, sondern *in natürlicher Weise* Entwicklungen, die die Informatik als Ganzes hervorgebracht haben, weiterführt: Der Computereinsatz für symbolische Rechnungen eröffnet einen neuen Abschnitt auf dem Weg des Computers vom primitiven Bitknipser zu einem Universalwerkzeug für geistige Arbeit. Es beginnt damit eine neue Etappe auf dem Weg der praktischen Realisierung des theoretischen Anspruchs, den die Church'sche These impliziert.

Die von der Computeralgebra produzierten Werkzeuge spielen bereits heute eine stark zunehmende Rolle sowohl in den Natur- als auch in den Ingenieurwissenschaften. Dies dokumentiert sowohl die wachsende Zahl von Anwenderpaketen der verschiedenen großen Systeme als auch eine beeindruckende Zahl von Büchern zu dieser Thematik. Für einen vollständigeren Überblick über Tendenzen und Anwendungen der Computeralgebra sei auf den schon erwähnten Computeralgebra-Report [8] verwiesen.

Ein pikantes Detail liegt in der Ignoranz dieser Entwicklungen durch Teile der etablierten Informatik selbst, denn Darlegungen zur Computeralgebra sucht man in verschiedenen Quellen, die sich eine umfassende Darstellung der Informatik vorgenommen haben, vergebens. So enthalten weder der "Duden Informatik" [5] noch das "Lexikon Informatik" [17] ein Stichwort *symbolisches Rechnen* oder *Computeralgebra*. Aber auch hier scheinen sich Gewichte zu verschieben, wie ein Blick in das neue "Handbuch Informatik" [16] belegt, in dem ein ganzer Abschnitt dem symbolischen Rechnen gewidmet ist (aus dem wir weiter oben bereits zitiert haben).

In der Tat kann man den Einfluss dieser neuen Arbeitsmittel auf den Umbruch unserer technisierten Arbeitswelt insbesondere im ingenieurtechnischen Bereich kaum überschätzen. Dort, wo heute noch dicke Formelsammlungen und Tafelwerke das Berufsbild prägen, die trotz ihrer Dicke genau wie unser Berg numerischer Daten immer nur eine sehr beschränkte Sicht auf *Fakten* und keine *Einsichten* vermitteln können, werden Werkzeuge, die auf symbolischen Fähigkeiten im beschriebenen Sinne aufsetzen, diese Bereiche geistiger Arbeit in vielleicht noch nachhaltigerer Weise revolutionieren als dies mit der Erfindung des Buchdrucks geschah.

Schließlich eröffnen die damit verbundenen Möglichkeiten, nun auch algorithmisches Know How in großem Umfang zu vergegenständlichen, vollkommen neuen Dimensionen der Wissensrepräsentation.

J.Grabmeier beschreibt die Perspektiven eines solchen

Übergangs von einer fakten- zu einer stärker algorithmenorientierten Wissensrepräsentation

in [7] wie folgt:

> Viele Probleme aus der Ingenieurwelt, den Naturwissenschaften und den Wirtschaftswissenschaften sind heute ohne massiven Einsatz von Computern nicht lösbar. Die dahinterliegenden Probleme werden mit den Methoden des Wissenschaftlichen Rechnens angegangen. Dabei werden mehr und mehr die traditionellen numerischen Rechnungen durch symbolisches Rechnen mit dem Computer ersetzt bzw. ergänzt...
>
> Der Siegeszug der Computeralgebra in den letzten Jahren ist eng gekoppelt mit der stürmischen Entwicklung von immer neuen Rechnergenerationen, die es erst möglich gemacht hat, die besonders rechen- und speicherintensiven Programme und Systeme zum symbolischen Rechnen zu realisieren.
>
> Aber der Aufwand lohnt sich: Wenn man statt einer Zahl eine parameterabhängige Formel als Ergebnis erzielt, hat man nicht nur ein Problem gelöst, sondern eine Klasse von möglicherweise *unendlich vielen* Problemen erledigt. Dadurch wird ein Qualitätssprung möglich, denn die Formel erlaubt es nun z.B., die Parameter zu optimieren oder schnell auf Veränderungen zu reagieren. ...
>
> Wie heute ein Taschenrechner zum Alltag gehört, wird künftig jeder Ingenieur und jeder, zu dessen Aufgaben das Lösen, Erlernen oder Lehren mathematischer Probleme gehört, Zugriff auf ein Computeralgebra-System haben. Die verschiedenen schon heute verfügbaren Komponenten für numerisches und symbolisches Rechnen, für Statistik und andere mathematische Gebiete, für Graphik und Animation, Textverarbeitung und Dokumentation mit Hypertext-Systemen und vieles mehr sehe ich in nicht allzu ferner Zukunft über entsprechende Schnittstellen zu individuell kombinierbaren Computermathematik-Systemen für das Wissenschaftliche Rechnen zusammenwachsen. Die Computeralgebra leistet damit einen wesenlichen Beitrag für eine der Schlüsseltechnologien unserer technikbestimmten Gesellschaft.

2.6 Computeralgebrasysteme (CAS) – Ein Überblick

Die ersten Anfänge der Entwicklung von Programmen der Computeralgebra reichen bis in die frühen 50er Jahre zurück. [21] nennt in diesem Zusammenhang Arbeiten aus dem Jahre 1953 von H.G.Kahrimanian (Temple University in Philadelphia) und von Norton (?) (Massachusetts Institute of Technology) zum analytischen Differenzieren. Kahrimanian schrieb in diesem Rahmen auch ein Assemblerprogramm für die Univac I, das damit als Urvater der CAS gelten kann.

Ende der 50er und Anfang der 60er Jahre wurden am MIT große Forschungsanstrengungen unternommen, symbolisches Rechnen mit eigenen Hochsprachen zu entwickeln (Formula ALGOL, ABC ALGOL, ALADIN, ...). Von diesen Sprachen hat sich bis heute vor allem LISP als die Grundlage für die meisten Computeralgebrasysteme erhalten, weil in dessen Sprachkonzept die strenge Trennung von Daten- und Programmteil, deren Überwindung wir als für die Computeralgebra wesentlich herausgearbeitet hatten, bereits aufgehoben ist.

Die Wurzeln der ersten allgemeinen Systeme liegen in Versuchen verschiedener Fachwissenschaften, die im jeweiligen Kalkül anfallenden umfangreichen symbolischen Rechnungen einem Computer als Hifsmittel zu übertragen. Schwardmann [18] weist auf die Programme CAYLEY für Algorithmen in der Gruppentheorie und SAC zum Rechnen mit multivariaten Polynomen und rationalen Funktionen (Mathematik) sowie SHEEP für die Relativitätstheorie und MOA für die Himmelsmechanik (Physik) hin. [21] enthält einen detaillierteren Überblick über die Entwicklungen und Systeme jener Zeit.

Die ersten Systeme allgemeiner Ausrichtung, die nicht (nur) für spezielle Anforderungen eines Faches konzipiert wurden, die Systeme Reduce und Macsyma, basieren auf LISP und sind seit Mitte der 60er Jahre im Einsatz. Reduce wurde von A. Hearn aus einem System für Berechnungen in der Hochenergiephysik (Feynman-Diagramme, Wirkungsquerschnitte) entwickelt, während Macsyma am MIT im Zusammenhang mit Untersuchungen zur künstlichen Intelligenz im Rahmen des DARPA-Programms entstand. Grabmeier klassifiziert diese in [7] als *Allzwecksysteme der ersten Generation* und weist darauf hin, dass die programmiertechnischen Restriktionen jener Zeit (Lochstreifen, Stapelbetrieb) für symbolische Rechnungen, die aus noch darzulegenden Gründen meist stärker dialogorientiert sind, denkbar ungeeignete Bedingungen boten.

Mit der Entwicklung stärker dialogorientierter Rechentechnik in der 70er und 80er Jahren erhielten diese Entwicklungen neue Impulse. Sie beginnen ebenfalls, wie auch die des Computers als "number cruncher", bei der Arithmetik, hier allerdings der *Arithmetik symbolischer Ausdrücke*. Entsprechend bilden bei den Computeralgebrasystemen der zweiten Generation eine interaktiv zugängliche Polynomarithmetik zusammen mit einem regelbasierten Simplifikationssystem den Kern des Systemdesigns, um den herum mathematisches Wissen in Anwenderbibliotheken gegossen wurde und wird. Diese Systeme sind durch ein Zwei-Ebenen-Modell gekennzeichnet, in dem die Interpreterebene zwar gängige Programmablaufstrukturen unterstützt, jedoch keine Datentypen kennt, sondern es prinzipiell erlaubt, alle im Kernbereich, der *ersten Ebene*, implementierten symbolischen Algorithmen mit allen möglichen Daten zu kombinieren. Weitergehende programmiersprachliche Elemente wie insbesondere Typisierung werden nur rudimentär unterstützt.

Neben neuen Versionen der "alten" Systeme Reduce und Macsyma entstanden dabei eine Reihe neuer Systeme, von denen besonders Maple, Mathematica und Derive zu nennen sind.

Mathematica entwickelte sich aus dem von Steven Wolfram Ende der 70er Jahre entworfenen System SMP, das wiederum aus der Notwendigkeit geboren wurde, komplizierte algebraische Manipulationen in bestimmten Bereichen der theoretischen Physik effektiv und zuverlässig auszuführen. Es ist das erste System, das unmittelbar in der sich zu dieser Zeit im Compilerbereich durchsetzenden Sprache C geschrieben wurde. In der weiteren Entwicklung spielte dieses System eine Vorreiterrolle in der konsequenten Vermarktung von Software aus dem symbolischen Bereich, was bis heute in einer äußerst restriktiven Lizenzpolitik der inzwischen speziell für die Weiterentwicklung und den Vertrieb gegründeten Firma Wolfram Research, Inc. zum Ausdruck kommt. Der Versuch, schwerpunktmäßig über rein marktorientierte Mechanismen die Mittel für die Weiterentwicklung der algorithmischen und softwaretechnischen Seite eines solch komplexen Produkts aufzubringen, hat Mathematica, besonders mit seinen neuen Version 3.0 (Sommer 1996) und

4.0 (Frühjahr 1999), in die vorderste Front der "Großen" gebracht. Gleichwohl werden diese Aktivitäten, ähnlich derer von Bill Gates, von der weltweiten Gemeinschaft der Computeralgebraiker teilweise mit großen Vorbehalten verfolgt.

Einen anderen Weg der Kommerzialisierung gingen A.D. Rich und D.R. Stoutemyer mit der Gründung von Soft Warehouse, Inc. in Honolulu (Hawaii) ebenfalls im Jahre 1979. Neben Mainframes begannen zu dieser Zeit Arbeitsplatzcomputer mit geringen technischen Ressourcen eine zunehmend wichtige Rolle zu spielen, so dass die Frage entstand, ob man CAS mit ihren traditionell hohen Hardware-Anforderungen auch für solche Plattformen "zuschneiden" kann. Mit dem System muMATH-79 und der (wiederum LISP-basierten) Sprache muLISP-79 wurde darauf eine überzeugende Antwort gefunden. Nach fast zehnjähriger "Ehe" mit Microsoft nahm die Firma die weitere Entwicklung in die eigenen Hände und brachte 1988 ein Nachfolgeprodukt unter dem Namen DERIVE – *A Mathematical Assistant* auf den Markt, das mit minimalen Hardware-Anforderungen unter dem Betriebssystem DOS gute symbolische Fähigkeiten entwickelt. Nachdem in den letzten Jahren durch die rasante Erweiterung der Hardware-Ressourcen von Arbeitsplatzrechnern dieser Anwendungsbereich auch von den anderen CAS zunehmend erobert wird, konzentrierte sich die Firma auf das Taschenrechner-Geschäft und hat in Zusammenarbeit mit HP und TI zwei interessante Kleinstrechner mit symbolischen Fähigkeiten mitentwickelt, den HP-486X (1991) und den TI-92 (1995). Heute ist DERIVE ein Produkt, das mit großem Erfolg im schulischen Bereich eingesetzt wird. Um auf diesem Markt (dessen wirtschaftliche Potenzen die des gesamten wissenschaftlichen Bereichs um Größenordnungen übersteigen) erfolgreicher agieren zu können, wurde DERIVE im August 1999 von TI aufgekauft. Wie weit solche Produkte mit Laptops, die die volle Leistungsfähigkeit "großer" Systeme anbieten, konkurrieren können, wird die (nahe) Zukunft erweisen.

Ähnliche Überlegungen des "Downsizing" der bis dahin nur auf Mainframes laufenden großen Systeme lagen der Entwicklung von MAPLE an der University of Waterloo zugrunde. In nur drei Wochen wurde Ende 1980 eine erste Version (mit beschänkten Fähigkeiten) entwickelt, die auf B, einer ressourcen- und laufzeitfreundlichen Sprache aus der BCPL-Familie aufsetzte, aus der sich C als heutiger Standard entwickelt hat. Seit 1983 sind Versionen von MAPLE auch außerhalb der University of Waterloo in Gebrauch. Zur Gewährleistung einer effizienten Portabilität auf immer neue Computergenerationen wurde im Gegensatz zu den großen LISP-Systemen Wert auf einen kleinen, heute in C geschriebenen Systemkern gelegt, der die erforderlichen Elemente einer symbolischen Hochsprache implementiert, in der seinerseits die verschiedenen symbolischen Algorithmen geschrieben werden können.

Auch hier stellte sich schnell heraus, dass die Anforderungen, die Vertrieb, Support einer Nutzergemeinde sowie die Portierung auf immer neue Rechnergenerationen stellen, die Möglichkeiten einer akademischen Anbindung sprengen und (unter heutigen Bedingungen) nur über eine entsprechende Firma stabil zu gewährleisten ist. Diese Rolle spielt seit Ende 1987 Waterloo Maple Inc., die im Gegensatz zu Wolfram Research aber nach wie vor mit einer sehr engen Bindung an die universitäte Gruppe um K. Geddes und G. Labahn an der University of Waterloo, Ontario (Kanada), über ein ausgebautes akademisches Hinterland verfügt, das ein arbeitsteiliges Vorgehen in der softwaretechnischen und algorithmischen Weiterentwicklung des Systems ermöglicht.

Auch IBM als eine der großen Softwarefirmen hat sich seit den Anfangstagen der Computeralgebra mit eigenen Aktivitäten an den wichtigsten Entwicklungen beteiligt. Das betrifft insbesondere nach einigen Sprachentwicklungen in den 60er Jahren das System SCRATCHPAD, mit dem im *IBM T.J. Watson Research Center at Yorktown, NY,* seit den 70er Jahren diese Untersuchungen fortgesetzt wurden. SCRATCHPAD verwendet als bisher einziges System im Design ein strenges Typkonzept. Bis zum Beginn der 90er Jahre standen diese Entwicklungen ausschließlich auf IBM-Rechnerplattformen und nur in experimentellen Versionen zur Verfügung und fanden damit trotz ihrer Exzellenz keine weite Verbreitung. Auch mit der ersten offiziellen Version von AXIOM im Jahre 1991 änderte sich daran wenig. Das mag IBM bewogen haben, 1992 im Zuge einer generellen "Flurbereinigung" des Firmenprofils die Rechte an AXIOM der Firma NAG Ltd. zu übertragen, einer Non-profit-Firma, die sich bereits auf dem Gebiet wissenschaftlicher Software ausgewiesen hatte. Um eine Schnittstelle zu ihrem Hauptprodukt, der NAG Numerik-Bibliothek, erweitert, hat NAG in den letzten Jahren AXIOM auf eine Vielzahl von Plattformen portiert und auch den zugehörigen Standalone-Compiler ALDOR weiterentwickelt. Im Sommer 1998 ließ jedoch die Ankündigung aufhorchen, dass sich NAG strategisch auf MAPLE orientieren wird, so dass die Zukunft dieses Systems im ungewissen liegt.

AXIOM ist mit seinem streng getypten Ansatz ein

$$\boxed{\text{Computeralgebrasystem der dritten Generation,}}$$

mit denen versucht wird, ins Zentrum des Designs moderne programmiertechnische Ansätze der Typisierung, Modularisierung und Datenkapselung zu setzen und diese konsequent in einem symbolisch orientierten Kontext zu realisieren. Computeralgebrasysteme bieten hierfür denkbar gute Voraussetzungen, denn ein solches Typsystem ist ja seinerseits symbolischer Natur mit einem stark algebraisierten Hintergrund, sollte also prinzipiell mit den vorhandenen sprachlichen Mitteln eines CAS modelliert werden können. Konsequente Versuche, ein solches Typ-System mit den Mitteln eines CAS der zweiten Generation zu modellieren, wurden auch mit dem Maple-Paket Gauss ([9]) sowie dem Domain-Konzept in MUPAD vorgelegt. Die *Integration* eines solchen Typsystems in die Programmiersprache bedeutet jedoch, diese symbolischen Manipulationen unterhalb der Interpreterebene zu vollziehen, also gegenüber Systemen der zweiten Generation eine weitere Ebene zwischen Interpreter und Systemkern einzufügen, die diese Typinformationen in der Lage ist auszuwerten. Dieses Konzept wird von AXIOM auch in einer Standalone-Version, dem Programmiersystem ALDOR, mit beeindruckenden Ergebnissen verfolgt. Es zeigt sich, dass die algebraische Natur des Targets dieser Bemühungen mit der algebraischen Natur der theoretischen Fundierung von programmiertechnischen Entwicklungen gut harmoniert und etwa mit parametrisierten Datentypen und virtuellen Objekten (bzw. abstrakten Datentypen) bereits zu einer Zeit experimentiert wurde, in der an die bis heute rudimentären Versuche mit "Templates" und Ähnlichem in den "großen Sprachfamilien" C/C++ bzw. PASCAL/MODULA/OBERON noch nicht zu denken war. Besonders deutlich wird dies im Design von $\boxed{\text{MAGMA,}}$ einem Nachfolger von CAYLEY, das für komplexe Fragestellungen aus den Bereichen Algebra, Zahlentheorie, Geometrie und Kombinatorik entworfen wurde und Konzepte der universellen Algebra und Kategorientheorie in einem objektorientierten Ansatz direkt umsetzt.

System	neueste Version	Lizenzhalter	Preis Einzellizenz	Preis Stud.-version
AXIOM	2.2	NAG Ltd.	1710 DM	—
Derive		Texas Instruments	398 DM	159 DM
Macsyma	2.4.	Macsyma Inc.	450 DM	120 DM
Magma		The Computational Algebra Group at the University of Sydney	1000 $	—
Maple	V.5	Waterloo Maple Inc.	1200 $	180 DM
Mathematica	4.0	Wolfram Research Inc.	2500 DM	299 DM
MuPAD	1.4.2	MuPAD group Univ. Paderborn, SciFace Paderborn	498 DM	kostenlos
Reduce	3.7.	A. Hearn, Rand Corp. Santa Monica	835 DM	180 DM

Die großen Computeralgebrasysteme allgemeiner Ausrichtung im Überblick
(Stand September 1999)

In den letzten fünf bis zehn Jahren wurden alle diese Systeme, die bis dahin nur in einer mehr oder weniger experimentellen Fassung vorlagen, mit größerem Aufwand unter marktorientierten Gesichtspunkten weiterentwickelt. Schwerpunkt waren dabei vor allem die Einbindung von bequemeren, fensterbasierten Ein- und Ausgabetechniken auf Notebook-Basis, hypertextbasierte Hilfesysteme und leistungsfähige Graphikmoduln. Sie besitzen damit heute in der Regel eine ausgereifte Benutzeroberfläche, sind gut dokumentiert und auf den verschiedensten Plattformen (bis hin zu leistungsfähigeren Personalcomputern unter den gängigen Betriebssystemen Win 3.11, Win'95 oder OS/2 sowie LINUX) verfügbar[4]. Zu den Systemen gibt es einführende Bücher und Bücher zum vertieften Gebrauch, für spezielle Anwendungen und zum Einsatz in der Lehre. Zudem erscheinen regelmäßig Nutzerinformationen und Informationen über frei zugängliche Anwenderpakete. Weiterhin haben sich Benutzergruppen gebildet, und es werden Anwendertagungen durchgeführt.

Zugleich entstanden und entstehen auch neue Systeme allgemeiner Ausrichtung. Da CAS jedoch einen hohen Entwicklungsaufwand erfordern (mehr als 100 Mannjahre Programmierarbeiten), sind diese neuen Systeme in der Regel nicht sehr leistungsfähig oder haben nur einen sehr eingeschränkten Anwendungsbereich. Eine Ausnahme bildet hier das System MuPAD, dessen Entwicklung im Jahre 1989 von einer Arbeitsgruppe an der Uni-GH Paderborn unter der Leitung von Prof. B. Fuchssteiner begonnen und in den letzten Jahren unter aktiver Beteiligung einer großen Zahl von Studenten, Diplomanden und Doktoranden intensiv vorangetrieben worden ist. Auch hier stellte es sich heraus, dass ein solches System, wenn es eine gewisse Dimension erreicht hat, nicht allein aus dem akademischen Bereich heraus gewartet und gepflegt werden kann. Seit dem Frühjahr 1997 hat deshalb die eigens dafür gegründete Firma *SciFace* einen Teil dieser Aufgaben insbesondere aus dem software-technischen Bereich übernommen.

Auch das System MAGMA hat diesen Sprung inzwischen geschafft. Es wird von einer Gruppe um John Cannon an der School of Mathematics and Statistics an der University of Sydney entwickelt. Obwohl es die Entwickler selbst als "nichtkommerzielles System" bezeichnen, werden die Arbeiten zu einem Teil über Lizenzgebühren finanziert, die wissenschaftliche Einrichtungen und andere Nutzer für den Gebrauch des Systems zu zahlen haben.

Schließlich gibt es mehrere Entwicklungen, deren Anstrengungen nicht auf ein komplett anwendungsfähiges System, sondern eine Art Toolbox für symbolisches Rechnen ausgerichtet sind, einer Umgebung also, die es gestattet, neue Algorithmen und programmiertechnische Ansätze auf der Basis einer soliden Programmbibliothek zu untersuchen. Der Prototyp dieser Herangehensweise war das System ALDES/SAC-2 von G.COLLINS und R.LOOS, das inzwischen mehrere Migrationen auf neue Plattformen und Computerarchitekturen erfahren hat, etwa MAS (Modula), SACLIB (C), PARSAC und PACLIB (parallele Architekturen). In diesem Zusammenhang ist auch AXIOMs Standalone-Compiler ALDOR zu nennen, der einen zunehmenden Teil der mathematischen Fähigkeiten von AXIOM in ebendieser Weise zur Verfügung stellen und damit deren Integration auf Bibliotheksniveau in andere Programme ermöglichen soll.

Im Zusammenhang mit der Entwicklung paralleler Plattformen und verteilter Systeme auf Client-Server-Basis wird auch zunehmend über die Bereitstellung von symbolischer Rechenleistung in einem solchen Kontext nachgedacht. Hierfür sind in die gängigen Computeralgebrasysteme neue Kommunikationskonzepte zu integrieren, die an die besonderen Anforderungen an Struktur und Umfang des Datenaustauschs bei symbolischen Rechnungen angepasst sind.

Daneben gibt es eine große Anzahl kleinerer experimenteller Systeme mit eingeschränkter Ausrichtung. Sie wurden für spezielle Kalküle auf begrenzten Gebieten der Mathematik oder Physik entwickelt und erreichen durch angepasste Datenstrukturen und Algorithmen oftmals eine erstaunliche Leistungsfähigkeit. Sie sind meist das Produkt kleiner Gruppen von Spezialisten, die das Topwissen ihres Fachgebiets mit guten Programmierkenntnissen verbinden konnten. Beispiele für solche Systeme sind in der Physik SCHOONSHIP, FORM (Hochenergiephysik), CAMAL (Himmelsmechanik), SHEEP und STENSOR (allgemeine Relativitätstheorie) sowie in der Mathematik MAGMA und GAP (Gruppentheorie), PARI, KANT, SIMATH (Zahlentheorie), Macaulay, Macaulay-2, CoCoA und Singular (Algebra und algebraische Geometrie) und LIE (Lietheorie).

[4]Einige der Systeme haben sich aber nicht mehr der Mühe unterzogen, ihre Implementierung der 16-Bit-Welt von Win 3.11 anzupassen.

2.7 Und wie wird es weitergehen ?

Im Zuge der Herausbildung einer "Computermathematik", wie in [7] prognostiziert, entsteht zunehmend die Frage nach der Integration von Softwareentwicklungen, die bisher in anderen Kreisen vorangetrieben und gepflegt wurden. Dies betrifft insbesondere die Interaktion zwischen

- symbolischen Verfahren der Computeralgebra im engeren Sinne,
- in sehr umfangreichen Programmpaketen vorhandene ausgefeilte numerische Applikationen des "wissenschaftlichen Rechnens" sowie
- Entwicklungen der Computergraphik.

Die heute existierenden Computeralgebrasysteme haben sowohl numerische als auch graphische Fähigkeiten, die durchaus beeindrucken, jedoch von der Leistungsfähigkeit der genannten "professionellen" Entwicklungen weit entfernt sind. Aktuelle Bemühungen sind deshalb darauf gerichtet, in Zukunft die Vorteile arbeitsteiligen wissenschaftlichen Vorgehens stärker zu nutzen, statt das Fahrrad mehrfach zu erfinden. Auf der Softwareseite finden sich solche Bemühungen in Designkonzepten wieder, die sich stärker an der eigenen *Kernkompetenz* orientieren und den "Rest" durch Anbindung an andere Kernkompetenzen abdecken. Dies erfolgte für die großen CAS zuerst im Graphikbereich, wo sich AXIOM (Open Inventor) und REDUCE (Gnuplot) bereits seit einiger Zeit an eigenständige Entwicklungen im Graphiksektor angekoppelt haben. Weiterhin spielen Verbindungen zu Numerikpaketen wie die AXIOM-Anbindung an die Fortranbibliothek f90 oder die IRENA-Anbindung von REDUCE eine zunehmend wichtige Rolle. Auch von der Numerikseite her gibt es derartige Aktivitäten, wie die Einbindung von symbolischen Fähigkeiten von MAPLE in das im ingenieurtechnischen Bereich stark verbreitete MATHCAD 6.0 belegt.

Inzwischen ist dies zu einem deutlich sichtbaren Trend geworden und eigene Protokolle wie etwa MathLink für die Kommunikation zwischen C und MATHEMATICA entwickelt worden. Eine solche C-Einbindungsmöglichkeit liegt auch der Integration von MAPLE in MATHCAD 6.0 zu Grunde. Darüber hinaus gibt es inzwischen im Rahmen des OpenMath-Projekts Bemühungen, ein eigenes Protokoll für symbolische Rechnungen zu entwickeln, das von allen CAS unterstützt werden soll, um den Datenaustausch zwischen verschiedenen Applikationen aus diesem Gebiet generell zu erleichtern.

Der dritte große Bereich des Zugriffs auf Fremdkompetenz liegt in der Gestaltung des Ein- und Ausgabedialogs der großen Systeme. Mathematische Formeln sind oft auch von drucktechnisch komplizierter Gestalt und verwenden eine Fülle extravaganter Symbole in den verschiedensten Größen. Hierfür hat sich im Bereich der mathematischen Fachliteratur das Satzsystem TEX von D.E.Knuth und dessen etwas komfortablere Umgebung LATEX als ein de facto Standard weitgehend durchgesetzt. Deshalb bieten die meisten großen CAS auch eine Ausgabe in dieser Form an, die es erlaubt, Formeln direkt in mathematische Texte zu übernehmen bzw. diese direkt mit entsprechenden Wiedergabewerkzeugen in optisch ansprechender Form auszugeben. Ein großer Schritt vorwärts in dieser Richtung wurde durch die Notebook-Oberflächen von Mathematica 3.0 und Maple V.4 erreicht, die als direktes Graphik-Frontend für eine solche Darstellungsart ausgelegt sind.

> Computeralgebrasysteme werden damit zunehmend bequeme Werkzeuge für die eigene geistige Arbeit, die lokal auf dem Schreibtisch des Wissenschaftlers oder Ingenieurs einen immer größeren Teil des globalen Know Hows verschiedener Fachrichtungen in einer auch algorithmisch leicht zugänglichen Form bereithalten.

In Verbindung mit Client-Server-Techniken eröffnen sich für diese Entwicklungen nochmals vollkommen neue Perspektiven, die derzeit verstärkt untersucht werden: Ein "Kontrollzentrum" mit im wesentlichen nur Notebook- und Kommunikationsfähigkeiten als Mensch-Maschinensystem-Schnittstelle hat Zugang zu einer Vielzahl von Spezialprogrammen unterschiedlicher Kompetenz, die die gestellten Aufgaben arbeitsteilig lösen. So kann z.B. ein Programm den symbolischen Teil der Aufgabe bearbeiten, das Ergebnis (über das Kontrollzentrum) zur numerischen Auswertung einem zweiten Programm zur Verfügung stellen, dieses daraus Daten für eine graphische Auswertung erzeugen, die einem dritten Programm zur Graphikausgabe weitergereicht werden.

Solche Systeme würden es dann auch gestatten, auf die global und dezentral verteile Kompetenz anderer *unmittelbar* zuzugreifen, indem (wenigstens für ausgewählte Fragen, denn das Ganze wäre auch teuer) jeweils die neuesten und fortgeschrittensten Algorithmen und die leistungsfähigsten Implementierungen auf besonders diffizile Fragen angewendet werden könnten.

Wagen wir einen

Blick in die Zukunft:

Die Möglichkeiten, die sich aus einer zunehmenden Vernetzung auf der einen Seite und Allgegenwart von Computern auf der anderen abzeichnen, haben wir bisher noch gar nicht erfasst. In naher Zukunft wird es mit diesen Instrumenten möglich sein, zunehmend *selbst* in einem heterogenen Informationsraum zu operieren, statt mit vorgefertigten Instrumenten vor Ort eine *vorab generierte Kompetenz* zu konsultieren. Man wird statt dessen

1. weltweit auf von Spezialisten an verschiedenen Orten gepflegte Kompetenz unmittelbar zugreifen können
2. dezentral Messdaten mit Monitoring-Verfahren erfassen können (Brückenprüfung, Patientenüberwachung,...)
3. die damit generierten Erfahrungen selbst unmittelbar in diesem Informationsraum einbringen und damit anderen zugänglich machen können (World Wide Web)

Die Entstehung derartiger "kollektiver Vernunftformen" wird den Weg in die viel beschworene Wissensgesellschaft ganz entscheidend prägen.

Kapitel 3

Aufbau und Arbeitsweise eines Computeralgebrasystems der zweiten Generation

In diesem Kapitel wollen wir uns näher mit dem Aufbau und der Arbeitsweise eines Computeralgebrasystems der zweiten Generation vertraut machen und dabei insbesondere Unterschiede und Gemeinsamkeiten mit ähnlichen Arbeitsmitteln aus dem Bereich der Programmiersysteme herausarbeiten.

3.1 Interpreter oder Compiler ?

Programmiersysteme werden zunächst grob in Interpreter und Compiler unterschieden. CAS haben wir bisher ausschließlich über eine interaktive Oberfläche, also als Interpreter, kennengelernt. Es erhebt sich damit die Frage, ob es gewichtige Gründe gibt, symbolische Systeme gerade so auszulegen.

> Interpreter und Compiler sind Programmiersysteme, die dazu dienen, die in einer Hochsprache fixierte Implementierung eines Programms in ein Maschinenprogramm zu übertragen und auf dem Rechner auszuführen. Beide Arten durchlaufen dabei die Phasen *lexikalische Analyse, syntaktische Analyse* und *semantische Analyse* des Programms.
>
> Ein **Interpreter** bringt den aktuellen Befehl nach dieser Prüfung *sofort* zur Ausführung.
>
> Ein **Compiler** führt in einer ersten Phase, der *Übersetzungszeit*, die Analyse des vollständigen Programms aus, übersetzt dieses in eine maschinennahe Sprache und speichert es ab.
>
> Ein Compiler durchläuft damit die weiteren Phasen *Codegenerierung* und evtl. *Codeoptimierung.*
>
> In einer zweiten Phase, zur *Laufzeit*, wird das übersetzte Programm von einem *Lader* zur Abarbeitung in den Hauptspeicher geladen.

Ein **Compiler** betreibt also einen größeren Aufwand bei der Analyse des Programms, der sich in der Abarbeitungsphase rentieren soll. Das ist nur sinnvoll, wenn dasselbe Programm mit mehreren Datensätzen ausgeführt wird. Das zentrale Paradigma des Compilers ist also das des *Programmflusses*, längs dessen Daten geschleust werden.

Compiler werden deshalb vorwiegend dann eingesetzt, wenn ein und dasselbe Programm mit einer Vielzahl unterschiedlicher Datensätze abgearbeitet werden soll und die (einmaligen) Übersetzungszeitnachteile durch die (mehrmaligen) Laufzeitvorteile aufgewogen werden. Dies erfolgt besonders bei einer Sicht auf den Computer als "virtuelle Maschine", d.h. als programmierbarer Rechenautomat.

Compiler sind typische Arbeitsmittel einer stapelorientierten Betriebsweise.

Ein **Interpreter** dagegen zeichnet sich durch eine höhere Flexibilität aus, da auch noch während des Ablaufs in das Geschehen eingegriffen werden kann. Werte von (globalen) Variablen sind während der Programmausführung abfrag- und änderbar und einzelne Anweisungen oder Deklarationen im Quellprogramm können geändert werden. Ein Interpreter ist also dort von Vorteil, wo man verschiedene Daten auf unterschiedliche Weise kombinieren und modifizieren möchte. Das zentrale Paradigma des Interpreters ist also das der *Datenlandschaft*, die durch Anwendung verschiedener Konstruktionselemente erstellt und modifiziert wird.

Als entscheidender Nachteil schlagen längere Rechenzeiten für einzelne Operationen zu Buche, da z.B. die Adressen der verwendeten Variablen mit Hilfe der Bezeichner ständig neu gesucht werden müssen. Ebenso ist Code-Optimierung nur beschränkt möglich.

Interpreter werden deshalb vor allem in Anwendungen eingesetzt, wo diese Nachteile zugunsten einer größeren Flexibilität des Programms, der Möglichkeit einer interaktiven Ablaufsteuerung und der Inspektion von Zwischenergebnissen in Kauf genommen werden.

Interpreter sind typische Arbeitsmittel einer dialogorientierten Betriebsweise.

Eine solche Flexibilität ist eine wichtige Anforderung an ein CAS, wenn es als *Hilfsmittel für geistige Arbeit* eingesetzt werden soll. Dies ist folglich auch der Grund, weshalb alle großen CAS wenigstens in ihrer obersten Ebene Interpreter sind. Dies trifft selbst auf die erwähnten *Toolbox-Systeme* zu, die meist einen mit klassischen Analysewerkzeugen erstellten *Parser* als Minimaloberfläche anbieten.

Von dieser Dialogfläche aus werden einzelne Datenkonstruktoren (Funktionen, Unterprogramme) aufgerufen, für die jedoch eine andere Spezifik gilt: Bei diesen handelt es sich um standardisierte, auf einer höheren Abstraktionsebene optimierte algorithmische Lösungen, die während des Dialogbetriebs mit einer Vielzahl unterschiedlicher Datensätze aufgerufen werden (können). Sie gehören also in das klassische Einsatzfeld von Compilern.

Es ist deshalb nur folgerichtig, diese Teile eines CAS in vorübersetzter (und optimierter) Form bereitzustellen. Allerdings trifft dies nicht nur auf Systemteile selbst zu. Auch der Nutzer sollte die Möglichkeit haben, selbstentwickelte Programmteile, die mit mehreren Datensätzen abgearbeitet werden sollen, zu übersetzen, um in den Genuss der genannten Vorteile zu kommen.

Entsprechend dieser Anforderungsspezifikation sind große CAS allgemeiner Ausrichtung, wie übrigens heute alle größeren Interpretersysteme,

top-level interpretiert, low-level compiliert.

Bei der Übersetzung von Programmteilen gibt es drei Ebenen, die unterschiedliche Anforderungen an die Qualität der Übersetzung stellen:

- Während der Systementwicklung erstellte Übersetzungen, die in Form von Bibliotheken grundlegender Algorithmen mit dem System mitgeliefert (oder nachgeliefert) werden.
- Eigenentwicklungen, die mit geeigneten Instrumenten übersetzt und archiviert und damit in nachfolgenden Sitzungen in bereits compilierter Form verwendet werden können.
- Im laufenden Betrieb erzeugte Übersetzungen, die für nachfolgende Sitzungen nicht mehr zur Verfügung stehen (müssen).

Beispiele für die **erste Ebene** sind die Implementierungen der wichtigsten mathematischen Algorithmen, die die Leistungskraft eines CAS bestimmen. Hier wird man besonderen Wert auf den Entwurf geeigneter Datenstrukturen sowie die Effizienz der verwendeten Algorithmen legen, wofür neben entsprechenden programmiertechnischen Kenntnissen auch profunde Kenntnisse aus dem jeweiligen mathematischen Teilgebiet erforderlich sind.

Beispiele für die **zweite Ebene** sind Anwenderentwicklungen für spezielle Zwecke, die auf den vom System bereitgestellten Algorithmen aufsetzen und so das CAS zum Kern einer (mehr oder weniger umfangreichen) speziellen Applikation machen. Solche Applikationen reichen von kleinen Programmen, mit denen man verschiedene Vermutungen an entsprechendem Datenmaterial testen kann, bis hin zu umfangreichen Sammlungen von Funktionen, die Algorithmen aus einem bestimmten Gebiet von Mathematik, Naturwissenschaft oder Technik zusammenstellen.

Beispiele für die **dritte Ebene** sind schließlich zur Laufzeit entworfene Funktionen, die mit umfangreichem Datenmaterial ausgewertet werden müssen, wie es etwa beim Erzeugen einer Graphikausgabe anfällt.

Natürlich sind die Grenzen zwischen den verschiedenen Ebenen fließend. So muss im Systemdesign der ersten Ebene beachtet werden, dass nicht nur eine Flexibilität hin zu komplexeren Algorithmen zu gewährleisten ist, sondern auch eine Flexibilität nach unten, damit das jeweilige CAS möglichst einfach an unterschiedliche Hardware und Betriebssysteme angepasst werden kann. Hierfür ist es sinnvoll, das Prinzip der *virtuellen Maschine* anzuwenden, d.h. Implementierungen komplexerer Algorithmen in einer maschinenunabhängigen Zwischensprache zu erstellen, die dann von leistungsfähigen Werkzeugen plattformabhängig übersetzt und optimiert werden.

Die einzelnen CAS sind deshalb so aufgebaut, dass in einem Systemkern (unterschiedlichen Umfangs) Sprachelemente und elementare Datenstrukturen eines leistungsfähigen Laufzeitsystems implementiert werden, in dem dann seinerseits komplexere Algorithmen codiert sind.

Auch zwischen der ersten und zweiten Ebene ist eine strenge Abgrenzung nicht möglich, da die Implementierung guter konstruktiver mathematischer Verfahren eine gute Kenntnis des zugehörigen theoretischen Hintergrunds und damit oft eine akademische Einbindung dieser Entwicklungen wünschenswert macht oder gar erfordert. Außerdem werden in der Regel von Nutzern entwickelte besonders leistungsfähige spezielle Applikationen neuen Versionen des jeweiligen CAS hinzugefügt, um sie so einem weiteren Nutzerkreis zugänglich zu machen. In beiden Fällen ist es denkbar und in Einzelfällen auch schon geschehen, dass auf diese Weise entstandene Programmstücke aus Performance-Gründen Auswirkungen auf das Design tieferliegender Systemteile haben.

Auf diese Weise entstanden aus den Versionen der verschiedenen CAS, die bis Mitte der 80er Jahre von überschaubaren Personengruppen meist an einzelnen wissenschaftlichen Einrichtungen entwickelt wurden, Softwareprodukte, die vielfältigen "in Bytes gegossenen" Sachverstand bereithalten und deren weitere Entwicklung von einem schwer zu überschauenden Kreis von Nutzern und Entwicklern vorangetrieben wird.

> Das ist aber nur möglich, wenn große Teile des Systemdesigns selbst offenliegen und auch die technischen Mittel, die Nutzern und Systementwicklern zur Übersetzung und Optimierung von Quellcode zur Verfügung stehen, in ihrer Leistungsfähigkeit nicht zu stark voneinander differieren. Aus einer solchen Interaktion ergibt sich als weitere Forderung an Design *und* Produktphilosophie, dass Computeralgebrasysteme **offene Systeme** sein müssen.

Diese inhaltliche und oft auch personelle Interaktion zwischen Systementwicklern und Nutzern ist die wohl entscheidende Quelle der enormen Leistungskraft heutiger CAS. Dabei sind die Rollen zwischen Entwicklern und Anwendern von CAS nicht so klar verteilt wie bei vielen anderen Programmpaketen, sind doch gerade die *mathematischen* Fähigkeiten der großen CAS aus einer Vielzahl von Aktivitäten zur programmtechnischen Fixierung mathematischen Know-Hows entstanden, an denen sich Arbeitsgruppen beteiligen, die rund um die Welt verteilt sind und nur sehr lose Kontakte zueinander und zum engeren Kreis der Systementwickler halten. Letztere tragen hauptsächlich die Verantwortung für die Weiterentwicklung der entsprechenden programmiertechnischen Instrumentarien. Die Kommunikation zwischen diesen Gruppen erfolgt über Mailing-Listen, News-Gruppen, Anwender- und Entwicklerkonferenzen, Artikel in Zeitschriften, Bücher etc., insgesamt also mit den im üblichen Wissenschaftsbetrieb anzutreffenden Mitteln. Natürlich ergeben sich für ein konsistentes Management der Anstrengungen eines solch heterogenen Personenkreises im Umfeld des jeweiligen CAS

hohe Anforderungen auch im organisatorischen Bereich, die weit über Fragen des reinen Software-Managements hinausgehen

und mit den Anforderungen, die ein klassisches Produkt stellt, bei dem Nutzer und Entwicklerkreis streng getrennt sind, kaum zu vergleichen sind. Auf die von den einzelnen Systemen entwickelten Standards, wie die *Maple Shared Library*, die REDUCE-Bibliothek *RedLib* oder *MathSource* von MATHEMATICA werden wir hier jedoch nicht weiter eingehen.

Der Umfang des bereits implementierten mathematischen Wissens macht es aus Effizienzgründen weiterhin erforderlich, über eine $\boxed{\text{flexible Konfigurierbarkeit}}$ der Systeme entsprechend der jeweiligen Aufgabenstellung nachzudenken. In diesem Zusammenhang spielen insbesondere Konzepte des *dynamischen Ladens* eine wichtige Rolle. Dabei wird eine Startkonfiguration des Systems in den Hauptspeicher geladen, zu der je nach den Erfordernissen der auszuführenden Rechnungen zur Laufzeit weitere Codestücke hinzugeladen bzw. entfernt werden, womit der Hauptspeicherbedarf begrenzt werden kann.

Die (konsistente) Entwicklung und Betreuung solcher Systeme stellt sehr hohe Anforderungen an die verwendeten $\boxed{\text{Konzepte des Software-Designs.}}$ Alle modernen Entwicklungen, begonnen von Modularisierungs- und Datenkapselungskonzepten, über dynamisches Laden und Client-Server-Modelle bis hin zu Modellen verteilten Rechnens finden deshalb in diesem Bereich sowohl ein sehr komplexes Target, in dem die Leistungsfähigkeit neuer Konzepte geprüft werden kann, als auch eine Fülle von Anregungen und Fragestellungen, wie sie in dieser Komplexität von kaum einem zweiten Gebiet in der Informatik aufgeworfen werden.

3.2 Der prinzipielle Aufbau eines Computeralgebrasystems

Nach dem Start des entsprechenden Systems meldet sich die $\boxed{\text{Interpreterschleife}}$ und erwartet eine Eingabe, typischerweise einen in *linearisierter Form* einzugebenden symbolischen Ausdruck, der vom System ausgewertet wird. Das Ergebnis wird in einer stärker an mathematischer Notation orientierten *zweidimensionalen Ausgabe* angezeigt.

Neben derartigen Eingaben sowie einem `quit`-Befehl zum Verlassen der Interpreterschleife gibt es noch eine Reihe von Eingaben, die offensichtlich andere Reaktionen hervorrufen. Dies sind in MAPLE zum Beispiel

- Eingaben der Form `?topic` zur Aktivierung des Hilfesystems und
- Eingaben der Form `plot(...)`, worauf im Rahmen der X-Windows-Umgebung ein Graphik-Ausgabefenster geöffnet wird.

Offensichtlich werden hierbei andere als die unmittelbaren symbolischen Fähigkeiten des CAS herangezogen.

Die *Interpreterschleife* des CAS bringt also verschiedene Teile des Systems zusammen, wovon nur eines der unmittelbar symbolische Rechnungen ausführende Kern ist, den wir als den *Systemkern* bezeichnen wollen. Neben den beiden Komponenten *Hilfesystem* und *Graphik-Interface* sind dies noch die der Ein- und Ausgabe dienende *Nutzerschnittstelle* (front end) sowie ein *numerisches Interface*, das die numerische Auswertung symbolischer Ausdrücke übernimmt. Letzteres ist oftmals enger in den Systemkern intergriert. Wir wollen es trotzdem an dieser Stelle einordnen, da die entsprechende Funktionalität nicht direkt mit symbolischen Rechnungen zu tun hat.

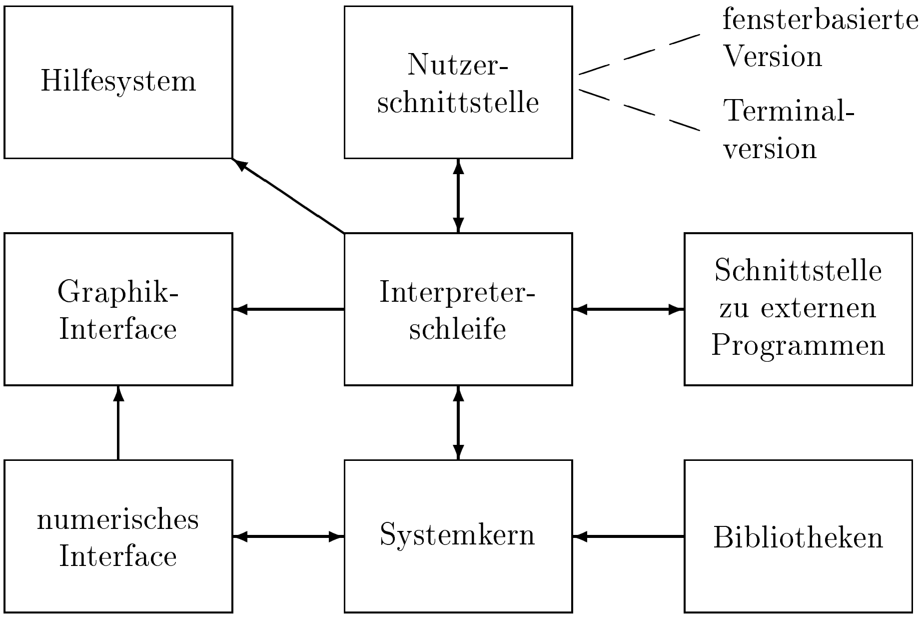

Bild 1: Prinzipieller Aufbau eines Computeralgebra-Systems

Ehe wir uns Aufbau und Arbeitsweise des Systemkerns zuwenden, in dem die symbolischen Fähigkeiten des jeweiligen Systems konzentriert sind, wollen wir einen kurzen Blick auf die anderen Komponenten werfen.

Das äußere Erscheinungsbild des Systems wird in erster Linie durch die Nutzerschnittstelle bestimmt. Genereller Bestandteil derselben ist ein *Formatierungssystem* zur Herstellung zweidimensionaler Ausgaben, das sich stärker an der mathematischen Notation orientiert als die Systemeingaben. Man unterscheidet *Terminalversionen*, in denen die Ausgabe im Textmodus erfolgt und die neben Ein- und Ausgabe meist einen rudimentären Zeileneditor besitzen, und *fensterbasierte Versionen*, in denen die Ausgabe im Graphikmodus erfolgt. Bei der Ausgabeformatierung mathematischer Formeln spielt die von D. Knuth entwickelte Seitenbeschreibungssprache TEX bzw. deren komfortablere Erweiterung LATEX eine zentrale Rolle, die sich inzwischen als defacto-Standard für die elektronische Form mathematischer Publikationen etabliert hat. Alle CAS können deshalb ihre Ergebnisse in diesem Format ausgeben.

Graphikbasierte Ausgabesysteme sind heute meist in der Lage, *integrierte Dokumente* zu produzieren, die Texte, Ergebnisse von Rechnungen und Graphiken verbinden und in gängigen Formaten (HTML, TEX) exportieren können. Eine besondere Vorreiterrolle spielen auf diesem Gebiet aktuelle Versionen von Mathematica und Maple mit dem Konzept des *Arbeitsblatts*. Hier treffen sich Entwicklungen aus dem Bereich des wissenschaftlichen Publizierens (mit Produkten wie *Scientific Word*[1]), der Gestaltung interaktiver Oberflächen und Methoden des symbolischen Rechnens, womit sich zugleich vollkommen neue Horizonte in Richtung der (elektronischen) Publikation interaktiver mathematischer Texte eröffnen.

Hilfesysteme sind bei den einzelnen CAS sehr unterschiedlich entwickelt. Generell ist jedoch auch hier ein Trend hin zur Verwendung gängiger Techniken zum Entwurf von Hilfesystemen in Form hypertextbasierter Dokumente und entsprechender Analysewerkzeuge zu verspüren. Dabei wird zunehmend, wie im letzten Kapitel bereits für das Graphik-Interface beschrieben, nicht nur

[1] Siehe `http://www.tcisoft.com`

auf entsprechende Ansätze, sondern auch auf bereits fertige Software-Komponenten zurückgegriffen. Hilfesysteme sind meist hierarchisch oder/und nach Schlagworten sortiert, so dass man relativ genaue Kenntnisse benötigt, wie konkrete Kommandos oder Funktionen heißen bzw. an welcher Stelle in der Hierarchie man relevante Informationen findet.

Ein besonders interessantes Hilfesystem, das System *MathTips*, ist von MACSYMA entwickelt worden. Es basiert auf assoziativen Zugriffen auf die darunterliegende Informationsbasis, so dass man Fragen in "natürlichem Englisch" eingeben kann. Das System versucht dazu relevante Informationen zu finden.

Obwohl es auch große und leistungsfähige Programmpakete zur Numerik gibt, treiben die CAS die Entwicklung ihres $\boxed{\text{numerischen Interface}}$ in zwei Richtungen voran. Zum einen ist zu bedenken, dass ein CAS nur dann zu einem nützlichen Werkzeug in der Hand eines Wissenschaftlers oder Ingenieurs wird, wenn es die volle "compute power" bereitstellt, die von dieser Clientel im Alltag benötigt wird. Dazu gehören neben symbolischen Rechenfertigkeiten und Visualisierungstools auch die Möglichkeit, ohne weitergehenden Aufwand symbolische Ergebnisse numerisch auszuwerten. Deshalb hat jedes der großen CAS eigene Routinen für numerische Berechnungen etwa von Nullstellen oder bestimmten Integralen "für den Hausgebrauch". Dabei wird stark von den speziellen Möglichkeiten adaptiver Präzision einer bigfloat-Arithmetik Gebrauch gemacht, die auf der Basis der vorhandenen Langzahlarithmetik leicht implementiert werden kann. Diese Fähigkeiten, die etwa beim Bestimmen nahe beieinanderliegender Nullstellen von Polynomen oder beim Berechnen numerischer Näherungswerte hoher Präzision eine Rolle spielen, sind stärker in den Systemkern integriert.

Andererseits ist eine wichtige, wenn nicht gar die wichtigste[2] Anwendung von Computeralgebra die Aufbereitung symbolischer Daten zu deren nachfolgender numerischer Weiterverarbeitung. Deshalb werden neben Codegeneratoren auch zunehmend $\boxed{\text{explizite Schnittstellen und Protokolle}}$ entworfen, über die die Programme mit externem Sachverstand kommunizieren können. Über solche Schnittstellen kann insbesondere mit vorhandenen Numerikbibliotheken kommuniziert werden. Allerdings kann eine solche Schnittstelle umfangreichere Funktionen erfüllen, etwa Web-basierte Kommunikation mit Klienten organisieren oder kooperative Prozesse mehrerer CAS oder mehrerer Prozesse eines CAS koordinieren.

Betrachten wir nun die Anforderungen näher, die beim Design des $\boxed{\text{Systemkerns}}$ zu berücksichtigen sind, in dem die uns in diesem Kurs interessierenden symbolischen Rechnungen letztendlich ausgeführt werden. Diese Anforderungen kann man grob folgenden sechs Komplexen zuordnen:

- Es ist ein Konzept für eine *Datenrepräsentation* zu entwickeln, das es erlaubt, heterogen strukturierte Daten, wie sie typischerweise im symbolischen Rechnen auftreten, mit der notwendigen Flexibilität, aber doch nach einheitlichen Gesichtspunkten zu verwalten und zu verarbeiten. Damit verbunden ist die Frage nach einer entsprechend leistungsfähigen *Speicherverwaltung*.

  Dabei ist zu berücksichtigen, dass symbolische Ausdrücke sehr unterschiedlicher und im Voraus nicht bekannter Größe auftreten, also als *dynamische Datentypen* mit einer ebensolchen *dynamischen* Speicherverwaltung anzulegen sind, natürlich ohne auf Nutzerebene explizit Pointer und Speicher-Allokations-Operationen zuzulassen.

  Im Lichte statischer Typkonzepte wie im klassischen Pascal, das für diese Zwecke nur Pointer kennt, klingt eine solche Anforderung auf den ersten Blick sehr anspruchsvoll. Wer jedoch über einige Erfahrung im objektorientierten Programmieren dynamischer Strukturen verfügt, weiss, dass dynamische Datentypen nichts ungewöhnliches sind.

- Es wird eine *Programmiersprache* benötigt, mit der man den Ablauf der Rechnungen steuern kann. Bekanntlich reicht dafür ein kleines Instrumentarium an Befehlskonstrukten aus. Es muss deshalb möglich sein, diese gängigen Programmablaufkonstrukte (Schleifen, Verzweigungen, Anweisungsverbünde) auch in dieser Sprache zu formulieren. Weiterhin muss

[2] Nach [15] werden moderne CAS zu 90 % zur Generierung effizienten Codes eingesetzt.

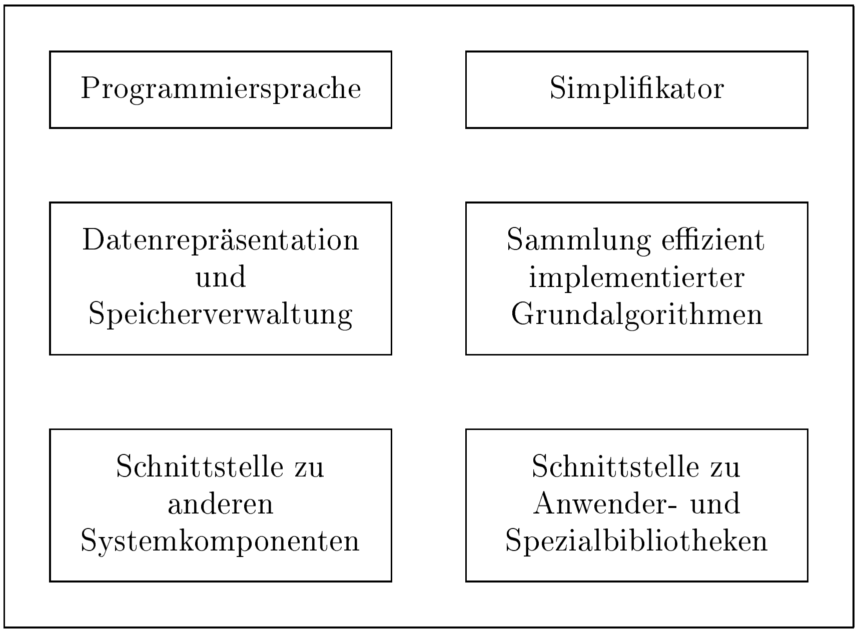

Bild 2: Komponenten des Systemkern-Designs

sie Methoden des strukturierten Programmierens (Prozeduren und Funktionen, Modularisierung) unterstützen, vermehrt um Instrumente, die sich aus der Spezifik des symbolischen Rechnens ergeben.

- Es ist ein Konzept für den *Simplifikator* zu entwickeln, mit dessen Hilfe Ausdrücke gezielt in zueinander äquivalente Formen nach unterschiedlichen Gesichtspunkten umgeformt werden können.

  Als Minimalanforderung muss dieser Simplifikator wenigstens in der Lage sein, in gewissem Umfang die semantische Gleichwertigkeit syntaktisch unterschiedlicher Ausdrücke festzustellen.

  Dabei handelt es sich meist um ein zweistufiges System, das aus einer *Polynomarithmetik* besteht, die in der Lage ist, rationale Ausdrücke umzuformen, und einem *Simplifikationssystem*, das die Navigation in der transitiven Hülle der dem System bekannten elementaren Umformungsregeln gestattet.

- Es werden *effiziente Implementierungen grundlegender Algorithmen* (Langzahl- und Polynomarithmetik, Rechnen in modularen Bereichen, zahlentheoretische Algorithmen, Faktorisierung von Polynomen, Bigfloat-Arithmetik, Numerikroutinen, Differenzieren, Integrieren, Rechnen mit Reihen und Summen, Grenzwerte, Lösen von Gleichungssystemen, spezielle Funktionen ...) benötigt, die in verschiedenen Kontexten des symbolischen Rechnens immer wieder auftreten.

  Diese in Form von black-box-Wissen vorhandene Kompetenz ist die Kernkompetenz des Systems. Die Implementierung dieser Algorithmen baut wesentlich auf anderen Teilen des Designkonzepts auf, die damit darüber entscheiden, wie effektiv Implementierungen überhaupt sein können. Gewöhnlich sind Teile dieser Sammlung von Funktionen aus Gründen der Performance nicht in der Programmiersprache des jeweiligen Systems, sondern maschinennäher ausgeführt.

  Die Anzahl und die Komplexität der eingebauten Funktionen ist mit klassischen Programmiersprachen nicht vergleichbar.

- Es ist ein Konzept für das Zusammenwirken der verschiedenen *Spezial- und Anwenderbibliotheken* mit dem Systemkern zu entwickeln, um das in ihnen gespeicherte mathematisch-algorithmische Wissen zu aktivieren.

  Spezialbibliotheken sind Sammlungen von Implementierungen algorithmischer Verfahren aus mathematischen Teildisziplinen, die in der Sprache des jeweiligen Systems geschrieben sind und speziellere Kalküle (Tensorrechnung, Gruppentheorie) zur Verfügung stellen. Derartige Spezialbibliotheken werden oftmals von der jeweiligen Nutzergemeinde als Gemeineigentum entwickelt und gepflegt und von den Lizenzhaltern der CAS nur gesammelt und weitergegeben (z.B. MAPLE's Shared Library, MATHEMATICA's MathSource, REDUCE's RedLib)

  Anwenderbibliotheken sind Sammlungen von Anwendungen mathematischer Methoden in anderen Wissenschaften, die auf den symbolischen Möglichkeiten des jeweiligen CAS aufsetzen. Solche Anwendungsbibliotheken (z.B. die MATHEMATICA Application Library) werden von den Firmen meist stärker kommerziell vertrieben.

- Schließlich ist ein Konzept für das *Zusammenwirken des Systemkern mit den anderen Systemkomponenten* zu entwickeln.

3.3 Klassische und symbolische Programmiersysteme

Das klassische Konzept einer imperativen Programmiersprache geht von der Vorstellung eines Programms als

> *schrittweise Transformation einer Menge von* **Eingabedaten** *in eine Menge von* **Ausgabedaten** *nach einem vorgegebenen* **Algorithmus**

aus ([5]).

Diese Transformation erfolgt durch *Abarbeiten einzelner Programmschritte,* in denen die Daten entsprechend den angegebenen Instruktionen verändert werden. Den Zustand der Gesamtheit der durch das Programm manipulierten Daten bezeichnet man auch als den *Programmstatus.*

Die Programmschritte werden gewöhnlich in *Anweisungen und Deklarationen* unterteilt, wobei Anweisungen den Programmstatus ändern, Deklarationen dagegen nicht, sondern lediglich Bedeutungen von Bezeichnern festlegen.

Anweisungen setzen sich aus elementaren Anweisungen zusammen, die durch entsprechende *Steuerstrukturen* miteinander verbunden sind. Elementare Anweisungen können *Zuweisungen* oder *Prozeduraufrufe* sein. Durch erstere wird direkt der Wert einer Variablen geändert, durch zweitere erfolgt die Änderung des Programmstatus als Seiteneffekt.

Bei **Deklarationen** unterscheidet man gewöhnlich zwischen Deklarationen von *Datentypen*, *Variablen*, *Funktionen* und *Prozeduren.* Jede solche Deklaration verbindet mit einem *Bezeichner* eine gewisse Bedeutung. Neben Variablenbezeichnern gibt es (z.B. in Pascal oder C) noch Bezeichner für Funktionen, Typen, Marken und Konstanten, die jedoch nur zur Compilezeit eine Bedeutung haben, zur Laufzeit dagegen maschinenabhängig durch konkrete Speicheradressen ersetzt sind.

Hinter demselben Bezeichner können sich in einem Programm allerdings unterschiedliche Objekte verbergen, was durch die Begriffe *Sichtbarkeit* und *Lebensdauer* beschrieben wird. Zur Übersetzung wird deshalb eine Tabelle, die $\boxed{\text{Symboltabelle,}}$ angelegt, in der die jeweils gültigen Kombinationen von Bezeichner und Bedeutung gegenübergestellt sind.

Obwohl diese Referenzen zur Laufzeit aufgelöst sind, verbleibt im Zusammenhang mit Variablen eine andere Referenz, die zwischen ihrer *Adresse* und dem dort hinterlegten *Wert.* Die Adresse spielt dabei die Rolle des Bezeichners, über den man das entsprechende Datum eindeutig identifizieren kann.

In einem Interpreter, der ja Syntaxanalyse und Abarbeitung kombiniert, existieren beide Referenzmechanismen nebeneinander, d.h. es gibt eine Symboltabelle, die z.B. einem Variablenbezeichner Adresse (Speicherplatz) und Typ (Speichergröße) zuordnet, und die Speicheradressen selbst, unter der man den jeweiligen Wert findet.

Symboltabellen, die im klassischen Fall symbolische Informationen über *programmrelevante Eigenschaften* verschiedener Symbole vorhalten, kommt für symbolische Rechnungen eine zentrale Rolle für die Speicherung auch darüber hinausgehender semantischer Informationen zu.

Ausdrücke

In klassischen Programmiersprachen ergibt sich der Wert, der einer Variablen zugeordnet wird, als Ergebnis eines *Ausdrucks* oder *Funktionsaufrufs*. Auch in Funktionsaufrufen selbst spielen (zulässige) Ausdrücke als Aufrufparameter eine wichtige Rolle, denn man kann sie statt Variablen an all den Stellen verwenden, an denen ein *call by value* erfolgt.

Der Begriff des Ausdrucks spielt in klassischen Programmiersprachen eine zentrale Rolle, wobei Funktionsaufrufe als spezielle Postfix-Ausdrücke aufgefasst werden, siehe etwa [20, R.5.2]. Zulässige Ausdrücke werden rekursiv als nach bestimmten Regeln zusammengesetzte Zeichenketten definiert, in denen neben Konstanten, Variablenbezeichnern und Funktionsaufrufen auch noch verschiedene *Operationszeichen* auftreten. So sind etwa $a + b$ oder $b - c$ ebenso zulässige Ausdrücke wie $a + gcd(b, c)$, wenn a, b, c als `integer`-Variablen und `gcd` als zweistellige Funktion `gcd:(int,int)` $\mapsto$ `int` vereinbart wurden.

Solche zweistelligen Operatoren unterscheiden sich allerdings nur durch ihre spezielle Notation von zweistelligen Funktionen. Man bezeichnet sie als *Infix-Operatoren* im Gegensatz zu der gewöhnlichen Funktionsnotation als *Präfix-Operator*. In manchen Sprachen spielen auch *Postfix-Notationen* von Funktionen (etwa das Fakultätssymbol) eine Rolle.

Wir wollen umgekehrt Ausdrücke als spezielle Funktionsaufrufe verstehen, da intern die Darstellungen in einer Präcompilierphase sofort in die entsprechende Präfix-Notation umgewandelt werden, so dass die obigen Ausdrücke in der Form `+(a,b), -(b,c), +(a,gcd(b,c))` erscheinen. Dabei sind eine Reihe weiterer Besonderheiten wie Ausdrücke der Form $a + b + c$ aufzulösen. Bei dieser Kurzschreibweise wird implizit die Tatsache verwendet, dass die Addition assoziativ ist. Um eine solche Summe aufzulösen, sind entsprechende Klammern zu setzen, um festzulegen, in welcher Reihenfolge die Addition auszuführen ist. Vereinbart man zum Beispiel, dass der Operator `+` linksassoziativ ist, so wird $a+b+c$ als $(a+b)+c$ berechnet. Ähnliche Fragen treten für Ausdrücke der Gestalt $a+b*c$ auf, bei deren Auswertung zu berücksichtigen ist, dass Punkt- vor Strichrechnung geht, d.h. die Multiplikation eine größere Bindungskraft hat als die Addition. Diese Tatsache wird gewöhnlich durch die Festlegung von *Vorrangzahlen* für die einzelnen Operatoren berücksichtigt, wobei Operatoren mit einer höheren Vorrangzahl zuerst angewendet werden.

Ausdrücke bestehen damit für einen klassischen Compiler (und Interpreter) nach der Auflösung dieser Diversitäten ausschließlich aus einer rekursiven Schachtelung von Funktionsaufrufen,

in die auf der untersten Ebene **Konstanten und Variablen** eingehen. Von Seiteneffekten einmal abgesehen (reference by name), ist der Unterschied zwischen Konstanten und Variablen unerheblich, da jeweils ausschließlich der *Wert* in die Berechnung des entsprechenden Funktionswerts eingeht.

Zur Analyse solcher Ausdrücke wird vom Compiler gewöhnlich ein Wurzelbaum aufgebaut, dessen Wurzel dem Funktionssymbol der obersten Ebene entspricht und dessen Blätter selbst wieder Wurzelbäume sind, die den einzelnen Ausdrücken entsprechen, die als Argumente auftreten. Die Blätter dieses Baums sind einzelne Konstanten und Variablen. Aus diesem Baum wird durch den Compiler ein Programm generiert, das den Wert des entsprechenden Funktionsausdrucks bestimmt. Auch ein Interpreter einer klassischen Sprache generiert ein solches Programm, so dass sich hier der entsprechende Wurzelbaum ebenfalls nur als Zwischenprodukt in der Auswertungsphase eines Ausdrucks ergibt.

In symbolischen Rechnungen bleiben allerdings “Formeln” als Ergebnisse stehen, für die eine Darstellung durch ebensolche oder ähnliche Wurzelbäume sinnvoll ist. Solche Konzepte spielen eine zentrale Rolle bei der Darstellung symbolischer Ausdrücke.

Die Prozesse, die in einem CAS zur *Laufzeit* ablaufen, haben damit viele Gemeinsamkeiten mit den Analyseprozessen, die in einem Compiler zur *Übersetzungszeit* stattfinden.

Datentypen und Polymorphie

Beim Auswerten von Ausdrücken spielt in klassischen Programmiersprachen *Polymorphie* eine wichtige Rolle, d.h. die Möglichkeit, Funktions- und insbesondere Operatorsymbolen unterschiedliche Bedeutung in Abhängigkeit von der Art der Argumente zu geben. Funktionen auf verschiedenen Strukturen mit gemeinsamen Eigenschaften durch dasselbe Symbol zu bezeichnen ist in der mathematischen Notation weit verbreitet und auch notwendig, um geeignete Abstraktionen überhaupt formulieren zu können.

Gleichwohl muss man diese Ambiguitäten in konkreten Implementierungen wieder auflösen, da natürlich z.B. zur Berechnung von $a + b$ für ganze und reelle Zahlen unterschiedliche Verfahren aufzurufen sind. Diese Unterscheidung vermögen Compiler klassischer Sprachen selbstständig zu treffen. Sie sind darüberhinaus auch in der Lage, etwa bei der Addition einer ganzen und einer reellen Zahl selbständig eine passende Typkonversion auszuführen.

Beschränkt sich Polymorphie in klassischen Ansätzen noch auf von den Systemdesignern vorgesehene Typambiguitäten, so hat sie inzwischen mit dem Paradigma der Objektorientierung auch in allgemeinerer Form in die Informatik Einzug gehalten. Grundlage der Polymorphie ist dabei die Möglichkeit, Referenzen auf denselben Funktionsnamen an Hand der *Typinformationen*, die die Parameter des jeweiligen Funktionsaufrufes tragen, aufzulösen.

Typsysteme sind damit ein wichtiges Instrument in modernen Programmiersprachen und wären sicher auch für das symbolische Rechnen von Bedeutung. Warum fehlt den großen CAS, die wir (und andere Autoren) unter dem Begriff “2. Generation” zusammenfassen, trotzdem ein solches Typkonzept bzw. ist ein solches nur in rudimentären Ansätzen vorhanden?

Untersuchen wir dazu zuerst einmal, welche Ansprüche an ein Typsystem im symbolischen Rechnen zu stellen sind. Datentypen sind (nach [5]) ein

> *grundlegender Begriff der Informatik, unter dem man die Zusammenfassung von Wertebereichen und Operationen zu einer Einheit*

versteht. Betrachen wir etwa den Ausdruck $f := 2 * x + 3$, so müsste ein passender Datentyp `Polynomial` die für Polynome übliche Signatur zur Verfügung stellen. Da Polynomoperationen sich aus Operationen auf Koeffizienten und auf Termen zusammensetzen, müsste ein qualifiziertes Typsystem wenigstens die unterschiedlichen Koeffizientenbereiche berücksichtigen. In einer Sprache wie Pascal oder bzw. C++ müssten wir also unterscheiden zwischen

```
class IntegerPolynomial {
    int coeff;
    Term exp; ...
}
```

und

```
class FloatPolynomial {
    float coeff;
    Term exp; ...
}
```

da diese unterschiedliche Additionen und Multiplikationen der Koeffizienten verwenden. Da jedoch viele andere Koeffizientenbereiche wie etwa rationale Zahlen, komplexe Zahlen und selbst Matrizen gelegentlich benötigt werden, ist es sinnvoller, Polynome als einen *parametrisierten Datentyp* anzulegen. `Polynomial(R)` konstruiert dann als Datentyp Polynome mit Koeffizienten aus dem Bereich R, für den selbst die entsprechenden Ringoperationen definiert sein müssen. Im Gegensatz zu Templates in C++,

```
<template R>
class Polynomial {
    R coeff;
    Term exp; ...
}
```

die für verschiedene Parameterwerte R vollständige Kopien der Implementierung des Datentyps anlegen, ist es allerdings sinnvoller, die Auflösung der Referenzen auf die Koeffizientenoperationen über den Aufrufparameter R vorzunehmen, der also ein *abstrakter Datentyp* sein müsste. Neben der Reduktion des entsprechenden Codes erlaubt ein solches Vorgehen auch, Polynomringe *dynamisch* zu erzeugen, d.h. als Koeffizienten Bereiche zu verwenden, an die zur Compilezeit des Polynomcodes nicht gedacht worden ist bzw. die vielleicht noch nicht einmal zur Verfügung standen.

Bereits diese einfache Überlegung zu Datentypkonzepten im symbolischen Rechnen führt uns also unvermittelt an die vorderste Front der Entwicklung programmiersprachlicher Instrumente.

Auch die Inferenz von Datentypen birgt erhebliche Probleme in sich. Um etwa den Ausdruck $f := 2 * x + 3$ als `Polynomial(Integer)` zu interpretieren, ist in einem ersten Schritt $2 * x$ zu berechnen, d.h. das Produkt aus `2:Integer` und `x:String` zu bilden. Dazu muss der gemeinsame Oberbereich erst gefunden werden, in den beide Datentypen transformiert werden können, damit die Multiplikation ausgeführt werden kann. Noch komplizierter wird es bei der Berechnung von $f+g$ mit $g = 2/3$. Für g ergibt sich bei entsprechender Analyse der Datentyp `Fraction(Integer)`, für f dagegen `Polynomial(Integer)`. Das Ergebnis könnte entweder die Einbettung `Integer`⊂ `Polynomial(Integer)` verwenden und damit den Typ `Fraction(Polynomial(Integer))` einer rationalen Funktion oder aber die Einbettung `Integer`⊂ `Fraction(Integer)` verwenden und damit den Typ `Polynomial(Fraction(Integer))` eines Polynoms mit rationalen Koeffizienten haben. Beiden Fällen ist gemein, dass der entsprechende Obertyp, in den eine Typkonversion erfolgen muss, aus einer Obertyprelation des Parameters zu inferieren ist.

Diese kleine Liste von Fragestellungen mögen als Begründung dienen, weshalb CAS der zweiten Generation auf ein konsistentes Typsystem im Sinne der Theorie der Programmiersprachen weitestgehend verzichten und Typinformationen nur insoweit verwenden, wie sie sich aus der *syntaktischen Struktur* der entsprechenden Ausdrücke extrahieren lassen.

Die zentrale Datenstruktur des symbolischen Rechnens

Für ein weitgehend typloses System sind alle Ausdrücke gleichartig, so dass es sinnvoll ist, eine geeignete Datenstruktur zu finden, mit der man Ausdrücke (also in unserem Verständnis geschachtelte Funktionsaufrufe) uniform darstellen kann. Anderenfalls hätte man für Polynome, Matrizen, Operatoren, Integrale, Reihen usw. jeweils eigene Datenstrukturen zu erfinden, was das Speichermanagement wesentlich erschweren würde. Die klassische Lösung des Ableitungsbaums mit dem Funktionsnamen in der Wurzel und den Argumenten als Söhne hat noch immer den Nachteil geringer Homogenität, da Knoten mit unterschiedlicher Anzahl von Söhnen unterschiedliche Speicherplatzanforderungen stellen.

Sehen wir uns deshalb an, wie Ausdrücke in Maple, Mathematica und Reduce intern dargestellt werden. Die folgenden Prozeduren gestatten es jeweils, diese innere Struktur sichtbar zu machen.

Mathematica:

Die Funktion `FullForm` gibt die interne Darstellung eines Ausdruck preis.

Maple:

```
structure:=proc(u) op(0..nops(u),u) end;
```

```
totalstructure := proc(u) local i;
     if nops(u)=1 and not type(u, function) then u
     else [seq(totalstructure(op(i,u)), i = 0 .. nops(u))]
     fi
end;
```

`structure` extrahiert die oberste Ebene, `totalstructure` stellt die gesamte rekursive Struktur der Ausdrücke dar.

REDUCE:

```
procedure struct(u); lisp prettyprint u;
```

`struct` stellt die gesamte rekursive Struktur des entsprechenden Ausdrucks dar.

Wir betrachten folgende Beispiele:

$$(x+y)^5$$

MATHEMATICA:	`Power[Plus[x, y], 5]`
MAPLE:	`[^,[+,x,y],5]`
REDUCE:	`(plus (expt x 5) (times 5 (expt x 4) y) (times 10 (expt x 3) (expt y 2)) (times 10 (expt x 2) (expt y 3)) (times 5 x (expt y 4)) (expt y 5))`

REDUCE überführt den Ausdruck sofort in eine expandierte Form. Dies können wir mit `expand` auch bei den anderen beiden Systemen erreichen. Die innere Struktur der expandierten Ausdrücke hat jeweils die folgende Gestalt:

MATHEMATICA:	`Plus[Power[x, 5], Times[5, Power[x, 4], y], Times[10, Power[x, 3], Power[y, 2]], Times[10, Power[x, 2], Power[y, 3]], Times[5, x, Power[y, 4]], Power[y, 5]]`
MAPLE:	`[+, [^, x, 5], [*, 5, [^, x, 4], y], [*, 10, [^, x, 3], [^, y, 2]], [*, 10, [^, x, 2], [^, y, 3]], [*, 5, x, [^, y, 4]], [^, y, 5]]`

Ähnliche Gemeinsamkeiten findet man auch bei der Struktur anderer Ausdrücke:

$1/2$	MATHEMATICA:	`Rational[1, 2]`
	MAPLE:	`[fraction, 1, 2]`
	REDUCE:	`(quotient 1 2)`
$1/x$	MATHEMATICA:	`Power[x, -1]`
	MAPLE:	`[^, x, -1]`
	REDUCE:	`(quotient 1 x)`
$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$	MATHEMATICA:	`List[List[1, 2], List[3, 4]]`
	MAPLE:	`[array, 1 .. 2, 1 .. 2, [(1, 1) = 1, (2, 1) = 3, (2, 2) = 4, (1, 2) = 2]]`
	REDUCE:	`(mat (1 2) (3 4))`
$\sin(x)^3+\cos(x)^3$	MATHEMATICA:	`Plus[Power[Cos[x], 3], Power[Sin[x], 3]]`
	MAPLE:	`[+, [^, [sin, x], 3], [^, [cos, x], 3]]`
	REDUCE:	`(plus (expt (cos x) 3) (expt (sin x) 3))`
Liste [a,b,c]	MATHEMATICA:	`List[a, b, c]`
	MAPLE:	`[list, a, b, c]`
	REDUCE:	`(list a b c)`

Wir sehen, dass in allen drei Systemen die interne Darstellung symbolischer Daten durch geschachtelte Listen erfolgt. Diese Darstellung orientiert sich eng an der durch Ableitungsbäume. Allerdings steht in MAPLE und REDUCE das entsprechende Funktionssymbol nicht an einer Extraposition in der Wurzel den entsprechenden Teilbaums, sondern als erstes Element, als *Kopf*, gleichrangig mit allen anderen Elementen in der Liste. Dasselbe gilt intern auch für MATHEMATICA, wo man auf die einzelnen Argumente eines Ausdruchs s mit `Part[s,i]` und auf das Kopfsymbol mit `Part[s,0]` zugreifen kann:

```
s=Sin[x+y]
s//FullForm

     Sin[Plus[x, y]]

s[[0]]

     Sin

s[[1]]

     x+y
```

Eine solche

> Darstellung von Funktionsaufrufen durch (geschachtelte) Listen, in denen das erste Listenelement den Funktionsnamen und die restlichen Elemente die Parameterliste darstellen,

ist typisch für die Sprache LISP, die Urmutter aller modernen CAS. CAS verwenden das erste Listenelement darüber hinaus auch zur Unterscheidung von einfachen Datenstrukturen (Listen, Mengen, Matrizen) sowie zur Speicherung von Typangaben atomarer Daten, also für ein

> syntaktisches Typsystem.

Ein solches Datendesign erlaubt eine hochgradig homogene Datenrepräsentation, denn jedes Listenelement besteht aus einem Pointerpaar ("dotted pair"), wobei der erste Pointer auf das entsprechende Listenelement, der zweite auf die Restliste zeigt. Nur auf der Ebene der Blätter tritt eine überschaubare Zahl anderer (atomarer) Datenstrukturen wie ganze Zahlen, Floatzahlen oder Strings auf. Eine solche homogene Datenstruktur erlaubt trotz der sehr unterschiedlichen Größe der verschiedenen symbolischen Daten die effektive dynamische Allokation und Reallokation von Speicher, da einzelne Zellen jeweils dieselbe Größe haben.

Listen als abstrakte Datentypen werden dabei allerdings anders verstanden als in den meisten Grundkursen "Algorithmen und Datenstrukturen" (etwa [12] oder [26]). Die dort eingeführte *destruktiv veränderbare Listenstruktur* mit den zentralen Operationen `Einfügen` und `Entfernen` ist für unsere Zwecke ungeeignet, da Ausdrücke gemeinsame Teilstrukturen besitzen können und deshalb ein destruktives Listenmanagement zu unüberschaubaren Seiteneffekten führen würde. Stattdessen werden Listen als *rekursiv konstruierbare Objekte* betrachtet mit Zugriffsoperatoren `first` (erstes Listenelement) und `rest` (Restliste) sowie dem Konstruktor `cons`, der ein neues Pointerpaar erstellt, mit dem (ein Pointer auf) ein Listenelement und (ein Pointer auf) eine Restliste zu einer neuen Liste zusammengefügt werden. Dieses für die Sprache LISP typische Vorgehen erlaubt es, auf aufwendiges Duplizieren von Teilausdrücken zugunsten des Duplizierens von Pointern zu verzichten, indem beim Kopieren einer Liste die Referenzen übernommen, die Links aber kopiert werden. Diese Sprachelemente prädestinieren einen funktionalen und rekursiven Umgang mit Listen, der deshalb in LISP und damit in CAS weit verbreitet ist. Für einen aus einer imperativen Welt kommenden Nutzer ist das gewöhnungsbedürftig.

3.4 Das Variablenkonzept des symbolischen Rechnens

Wir hatten bereits gesehen, dass die Analyse symbolischer Ausdrücke *zur Laufzeit*, die ja im Mittelpunkt des Interpreters eines CAS steht, weniger Ähnlichkeiten zum Laufzeitverhalten einer klassischen Programmiersprache hat als vielmehr zu den Techniken, die ein Compiler oder Interpreter derselben zur Übersetzungszeit verwenden.

Das trifft insbesondere auf das Variablenkonzept zu, in dem statt der klassischen Bestandteile *Bezeichner, Wert, Speicherbereich* und *Datentyp*, von denen in symbolischen Umgebungen sowieso nur die ersten beiden eine Bedeutung besitzen, neben dem Bezeichner *verschiedenartige* symbolische Informationen eine Rolle spielen, die wir als **Eigenschaften** dieses Bezeichners zusammenfassen wollen.

Die entscheidende Besonderheit in der Verwendung von Variablen in einem symbolischen Kontext besteht aber darin, dass sich **Namensraum und Wertebereich überlappen.** In der Tat ist in einem Ausdruck wie x^2+y nicht klar, ob etwa x hier als *Symbol* für sich selbst steht oder als *Variable* Container eines anderen Werts ist. Mehr noch kann sich die Bedeutung je nach Kontext unterscheiden. So wird etwa in einem Aufruf

```
> u:=int(1/(sin(x)*cos(x)-1),x);
```

x symbolisch gemeint sein, selbst wenn diese Variable bereits einen Wert besitzt.

Sehen wir uns diese Besonderheit des Variablenkonzepts zunächst in einigen Maple-Beispielen an.

```
p:=9*x^3-37*x^2+47*x-19;
```

$$9\,x^3-37\,x^2+47\,x-19$$

```
x:=2;
p;
```

$$-1$$

Das ist das erwartete Verhalten. Aus dem *Symbol* x wurde durch die Wertzuweisung die *Variable* x, deren *Wert* in die nachfolgende Auswertung von p eingeht.

Versuchen wir nun, diese Umwandlung rückgängig zu machen, etwa mit dem Versuch

```
x:=unknown;
p;
```

$$9\,unknown^3-37\,unknown^2+47\,unknown-19$$

x ist noch immer eine Variable, also Container eines nunmehr allerdings symbolischen Wertes, und kein Symbol.

```
x:=x;
```

$$x:=unknown$$

Auch diese "Verzweiflungstat" half nicht weiter, denn bei einer Zuweisung wird die rechte Seite ausgewertet und deren *Wert* der linken Seite zugewiesen.

Um x wieder zu einem Symbol zu machen, müssen wir diese Auswertung verhindern, was man in Maple wie folgt erreicht:

```
x:='x';
```

$$x := x$$

Wir haben auf diese Weise (scheinbar) der Variablen x als Wert das Symbol x zugewiesen.

```
p;
```

$$9\,x^3 - 37\,x^2 + 47\,x - 19$$

Wir sehen, dass aus einem Symbol durch eine Wertzuweisung automatisch eine (globale) Variable wird. Die *Deklaration* einer solchen globalen Variablen ist nicht notwendig, da ja kein Typ vereinbart werden muss.

Variablen werden in klassischen Compilern und Interpretern in einer *Symboltabelle* erfasst, um sie an allen Stellen, an denen sie im Quellcode auftreten, in einheitlicher Weise zu verarbeiten. Für jede Variable existiert **genau** ein Eintrag in der Tabelle, unter dem auch weitere Informationen über *Eigenschaften* der Variablen, wie ihr Datentyp, ihre Speicheradresse, ihr Auftreten in lokalen Kontexten usw. vermerkt sind.

Da in einem CAS jedes in irgendeinem Ausdruck auftretende Symbol später als Variable verwendet werden kann, muss dieser *Mechanismus der eindeutigen Referenz* in einem solchen Kontext auf **alle** auftretenden Symbole ausgedehnt werden. Dementsprechend wird bei der Analyse der einzelnen Eingaben jeder String, der ein Symbol darstellen könnte, daraufhin geprüft, ob er bereits an anderer Stelle aufgetreten ist und in diesem Fall eine Referenz an die entsprechende Stelle der Symboltabelle eingetragen. Andernfalls ist die Symboltabelle um einen neuen Eintrag zu erweitern.

CAS unterscheiden damit nicht zwischen Symbolen und Variablen.

In einigen CAS kann man sich diese Symboltabelle ganz oder teilweise anzeigen lassen. MAPLE verfügt sogar über zwei solche Funktionen. `unames()` listet alle bereits verwendeten Symbole auf, die keinen Wert besitzen (unassigned names), `anames()` die Symbole mit "Wert", die also Variablen im klassischen Sinne entsprechen[3].

Betrachten wir, wie sich diese beiden Mengen im Zuge von Wertzuweisungen ändern. Mit den folgenden Anweisungen kann man MAPLE zurücksetzen und sich damit den ursprünglichen Zustand beider Sequenzen ansehen:

```
restart: [unames()]; [anames()];
```

Eine ganze Reihe von Informationen, die beim Start von MAPLE geladen werden, liegen als Strings vor und sind deshalb nach dem beschriebenen Mechanismus in diese Tabelle aufgenommen worden. Die zweite Menge ist dagegen noch leer.

Betrachten wir einen Teil aus diesem Raum, die Namen der Länge 1, und beobachten, wie sich dieser Namensraum durch die Verwendung von Variablen verändert:

```
[anames()]; select(s->length(s)=1,[unames()]);
```

```
[]

[!, ., <, =, I, O, ^, s, x, y]
```

```
u; [anames()]; select(s->length(s)=1,[unames()]);
```

```
u
```

[3]Die in der Dokumentation gegebene Spezifikation ist allerdings nicht ganz korrekt, da nicht alle Symbole, die einen Wert haben, angezeigt werden, wie man sich leicht überzeugt, wenn man sich etwa mit `anames(integer)` alle Symbole mit Integer-Werten anzeigen lässt.

```
                         []

         [!, ., <, =, I, O, ^, s, u, x, y]
```

```
u:=1; [anames()]; select(s->length(s)=1,[unames()]);
```

```
                       u := 1

                         [u]

           [!, ., <, =, I, O, ^, s, x, y]
```

```
u:='u';[anames()]; select(s->length(s)=1,[unames()]);
```

```
                       u := u

                         []

         [!, ., <, =, I, O, ^, s, u, x, y]
```

In der letzten Zeile ist `u` aus der Liste der Variablen wieder verschwunden, d.h. die angegebene Zuweisung "löscht" den Wert von `u`. Symbole können also einen Wert besitzen, müssen es aber nicht. Hilfsweise kann man sich vorstellen, dass Variablen, die keinen Wert haben, zu sich selbst auswerten, was aber nicht vollkommen korrekt ist. So ist etwa eine Zuweisung

```
> x:=x;
```

in den meisten Systemen nicht zulässig.

In CAS besitzt nicht jedes Symbol einen Wert. Variablen müssen deshalb nicht initialisiert werden.

Die Eigenschaft eines Symbols, einen Wert zu besitzen, kann im Laufe seiner Lebenszeit mehrfach wechseln. Der für das symbolische Rechnen typische Ausgangszustand kann in den Systemen auf unterschiedlichem Weg hergestellt werden:

Axiom	)clear value x
Macsyma	kill(x)
Maple	x:='x'
Mathematica	Clear[x]
MuPAD	unassign(x)
Reduce	clear x

Tabelle 3: Symbole "wertlos" setzen.

Der Wert eines Symbols wird ähnlich wie andere Eigenschaften dem Symbol direkt zugeordnet. Dies kann man z.B. in Reduce verfolgen, indem man sich mit der Funktion `prop` die Eigenschaften des entsprechenden Symbols (in interner Notation) auflisten lässt:

```
prop 'x; prop 'y;
```

Anfangs verfügen beide Symbole über keine Eigenschaften.

```
x:=2$ prop 'x;

    ((avalue scalar 2))
```

Der zugewiesene Wert wurde unter dem Symbol x als eine Eigenschaft `avalue` gespeichert.

```
x:=y^2+z$ prop 'x;

    ((avalue scalar (!*sq ((((y . 2) . 1) ((z . 1) . 1)) . 1) t)))
```

Der geänderte Wert wurde auf dieselbe Weise, diesmal in einer bereits übersetzten Form, gespeichert.

```
prop 'y;

    (used!*)
```

Für y wurde zusätzlich die Information gespeichert, dass es als Symbol in einem anderen Ausdruck verwendet wurde.

```
clear x; prop 'x;

prop 'y;

    (used!*)
```

Löscht man x, so ist die Wertzuweisung verschwunden, die Zusatzinformation für y bleibt dagegen erhalten.

Dieses einheitliche Management der Symbole führt dazu, dass auch zwischen Variablenbezeichnern und Funktionsbezeichnern nicht unterschieden wird. Wir hatten beim Auflisten der dem System bekannten Symbole bereits an verschiedenen Stellen Funktionsbezeichner in trauter Eintracht neben Variablenbezeichnern stehen sehen.

CAS unterscheiden nicht zwischen Variablen- und Funktionsbezeichnern.

Da andererseits Funktionsdefinitionen ebenfalls symbolischer Natur sind und "nur" einer entsprechenden Interpretation bedürfen, verwenden einige Systeme wie etwa MAPLE sogar das Zuweisungssymbol, um Funktionsdefinitionen mit einem Symbol zu verbinden.

Eine solche einheitliche Behandlung interner und externer Symbole erlaubt es auch, Systemvariablen und selbst -funktionen zur Laufzeit zu überschreiben oder neue Funktionen zu erzeugen und (selbst in den compilierten Teil) einzubinden. Da dies zu unvorhersagbarem Systemverhalten führen kann, sind die meisten internen Funktionen allerdings vor Überschreiben geschützt. Hier einige Beispiele in MAPLE:

```
> Pi:=3.14;

Error, attempting to assign to `Pi` which is protected

> unprotect(Pi);
> Pi:=1;
                                   Pi := 1

> eval(arctan(1));
                                     1/4

> gcd:=2;

Error, attempting to assign to `gcd` which is protected

>  unprotect(gcd);
```

```
> gcd:=5;
                              gcd := 5

> gcd(2,3);
                                  5

> gcd(12,13);
                                  5
```

Wir sehen, dass es MAPLE nach Entfernen des Überschreibschutzes selbst erlaubt, Funktionsnamen als Variablennamen zu verwenden. Die meisten CAS unterscheiden jedoch zwischen Wert- und Funktionsdefinitionen und ordnen sie als *unterschiedliche* Eigenschaften dem jeweiligen Symbol zu.

Auswerten von Ausdrücken

Die Tatsache, dass der Wert von Variablen symbolischer Natur sein kann, in den seinerseits Symbole eingehen können, die selbst einen Wert besitzen, führt zu einer weiteren Besonderheit von CAS. Betrachten wir die folgenden Zuweisungen in MAPLE

```
a:=b;b:=c;c:=3;
                               a := b

                               b := c

                               c := 3
```

und sehen uns an, welche Wertzuweisungen für a vorgenommen worden ist:

```
a;
                                  3
```

Es ist also die gesamte Evaluationskette durchlaufen worden: Der Wert von a ist das Symbol b, das seinerseits als Wert das Symbol c hat, welches den Wert 3 besitzt. Denkbar wäre auch eine eingeschränkte Evaluationstiefe, etwa nur Tiefe 1. In MAPLE kann man diese Tiefe mit speziellen Kommandos variieren:

```
seq(eval(a,i),i=1..6);
                         b, c, 3, 3, 3, 3

b:=d;
                               b := d

a;
                                  d
```

Indem b ein anderer Wert zugewiesen wurde, ändert sich auch die Evaluationskette für a.

Die Variablen, die in den bisherigen Zuweisungen als rechte Seiten auftraten, hatten selbst noch keinen Wert. Bei Variablen mit Wert müssen wir unterscheiden, ob wir deren Wert oder aber das Symbol selbst meinen.

```
u:=3;a:=u;b:='u';
                               u := 3
```

```
                                   a := 3

                                   b := u
a;b;
                                     3

                                     3
```

An dieser Stelle ist der Unterschied noch nicht zu erkennen. Betrachten wir jedoch

```
u:=5;a;b;
                                   u := 5

                                     3

                                     5
```

so erkennen wir, dass sich Änderungen des Werts von u auf b auswirken, auf a dagegen nicht.

```
seq(eval(a,i), i=1..5);
                              3, 3, 3, 3, 3

seq(eval(b,i), i=1..5);
                              u, 3, 3, 3, 3
```

Geht in einen Ausdruck eine Variable x durch ihren *Wert* ein, so spricht man von *früher Auswertung*, wird sie dagegen nur als *Symbol* verwendet, so spricht man von *später Auswertung*, da sich ein evtl. vorhandener Wert erst bei späteren Auswertungen auswirkt. CAS benötigen also Mechanismen, um Variablen in beiden Bedeutungen, Wert und Symbol, einzusetzen. Dafür gibt es zwei gängige Herangehensweisen: Variablen, die häufiger in ihrer symbolischen Gestalt benötigt werden, weist man durch einen **Substitutionsoperator** *lokal* einen Wert zu, der nur in dem entsprechenden Ausdruck wirksam ist. Für Variablen, denen global ein Wert zugewiesen worden ist, gibt es einen Schutzmechanismus, der sie vor der Auswertung bewahrt. Dies geschieht entweder wie in Maple durch eine spezielle *Hold*-Funktion oder wie in Axiom und Mathematica durch zwei Zuweisungsoperatoren, wovon einer die in die rechte Seite eingehenden Variablen auswertet (*frühe Auswertung*), der andere dagegen nicht (*verzögerte Auswertung*). Mathematica verfügt sogar über beide Mechanismen.

System	Substitutions-operator	frühe Zuweisung	späte Zuweisung	Hold-Operator
AXIOM	subst(f(x),x=a)	x:=a	x==a	–
MACSYMA	subst(x=a,f(x))	x:a	–	'x
MAPLE	subs(x=a,f(x))	x:=a	–	'x'
MATHEMATICA	f(x) /.x↦a	x=a	x:=a	Hold[x]
MuPAD	subs(f(x),x=a)	x:=a	–	hold(x)
REDUCE	subst(x=a,f(x))	x:=a	let x=a	–

Tabelle 4: Substitutions- und Zuweisungsoperatoren der verschiedenen Systeme

Diese Besonderheiten sind in einer klassischen Programmiersprache, in der Namensraum und Wertebereich getrennt sind, unbekannt.

Bei dem bisherigen Evaluierungsverfahren maximaler Tiefe kann es eintreten, dass ein zu evaluierendes Symbol nach endlich vielen Evaluationsschritten selbst wieder in dem entstehenden

Ausdruck auftritt und damit der Evaluationsprozess stets von Neuem durchlaufen wird wie in folgendem Beispiel (MAPLE, ähnlich MUPAD):

```
x:=y+1;
                                    x := y + 1

y:=x;
Warning, recursive definition of name
                                    y := y + 1

y;
Error, too many levels of recursion
```

Solche rekursiven Verkettungen können leicht eintreten, wenn man die Konsequenzen der getroffenen Wertzuweisungen nicht genau überblickt. Auch aus diesem Grund spielen lokale Wertzuweisungen, die nur innerhalb eines einzigen Aufrufs Gültigkeit haben, eine wichtige Rolle.

REDUCE erlaubt eine solche rekursive Zuweisung nicht, sondern bricht mit einer Fehlermeldung ab:

```
x:=x+1;

***** x improperly defined in terms of itself
```

MATHEMATICA besitzt eine interne Variable `$RecursionLimit`, die die Anzahl der Evaluationsschritte, die wirklich ausgeführt werden, begrenzt.

Die beiden Systeme AXIOM und MACSYMA verwenden standardmäßig nur Evaluationen der Tiefe 1, d.h. ersetzen standardmäßig nach dem ersten Evaluationsschritt auftretende Symbole nicht weiter durch ihre Werte, wodurch dieses Problem der rekursiven Wertzuweisung ebenfalls vermieden wird. Allerdings unterscheidet sich damit das Auswertungsverhalten dieser beiden Systeme von dem der anderen CAS.

Zuweisung	Axiom Macsyma	Maple Mathematica MuPAD Reduce
a:=b+1	b+1	b+1
b:=c+1	c+1	c+1
c:=d+1	d+1	d+1
a	b+1	d+3
a:=b+1	c+2	d+3

Tabelle 5: Unterschiedliches Auswertungsverhalten der verschiedenen Systeme

3.5 Funktionen, Ausdrücke und Funktionssymbole

Funktionen bezeichnen im mathematischen Sprachgebrauch *Abbildungen* $f : X \longrightarrow Y$ von einem Definitionsbereich X in einen Wertevorrat Y.

In klassischen Programmiersprachen steht der konstruktive Aspekt einer solchen Betrachtung im Vordergrund, d.h. der *Algorithmus*, nach welchem die Abbildungsvorschrift f jeweils realisiert werden kann. Eine Funktion ist in diesem Sinne eine (beschreibungs-)endliche Berechnungsvorschrift, die man auf Elemente $x \in X$ anwenden kann, um entsprechende Elemente $f(x) \in Y$ zu produzieren. Wir unterscheiden dazu $\boxed{\text{Funktionsdefinitionen}}$ und $\boxed{\text{Funktionsaufrufe.}}$

Die algorithmische Beschreibung in einer Funktionsdefinition verwendet formale Parameter als Platzhalter, die in einem Funktionsaufruf mit konkreten Werten (call by value) initialisiert und danach abgearbeitet werden. Wird eine nicht definierte Funktion aufgerufen, so wird (in Compilern meist bereits zur Übersetzungszeit) ein Fehler gemeldet.

Symbolische Systeme kennen dagegen auch $\boxed{\text{Funktionssymbole,}}$ d.h. "Funktionen ohne Funktionsdefinition". Dies ist vollkommen analog zum Wechselverhältnis zwischen Variablen und Symbolen, da wir letztere als "Variablen ohne Wert" kennengelernt haben.

Die Auswertung eines Funktionsaufrufs folgt dem klassischen Schema, wobei als Wert eines Aufrufparameters ein symbolischer Ausdruck im weiter oben definierten Sinne auftritt. Existiert keine Funktionsdefinition, so wird allerdings nicht mit einer Fehlermeldung abgebrochen, sondern aus den Werten der formalen Parameter und dem Funktionssymbol als "Kopf" ein neuer Ausdruck gebildet. Einen solchen symbolischen Ausdruck, dessen Kopf ein Funktionssymbol ist, wollen wir auch als $\boxed{\text{Funktionsausdruck}}$ bezeichnen[4].

Es kann sogar der Fall eintreten, dass eine Funktion nur partiell (im informatischen Sinne) definiert ist, d.h. für spezielle Argumente eine Auswertung stattfindet, für andere dagegen ein Funktionsausdruck gebildet wird. So wird etwa in den folgenden MAPLE-Konstrukten das Funktionssymbol `sin` zur Konstruktion symbolischer Ausdrücke verwendet, die für Sinus-Funktionswerte stehen, die nicht weiter ausgewertet werden können.

```
sin(x); sin(2);
```

$$\sin(x)$$

$$\sin(2)$$

Im Gegensatz dazu sind

```
sin(Pi/4);
```

$$1/2\,\sqrt{2}$$

```
sin(2.55);
```

$$0.5576837174$$

klassische Funktionsaufrufe, die eine Funktionsdefinition von sin verwenden.

Ähnlich den Symbolen könne auch neue Funktionssymbole eingeführt werden, von denen zunächst nichts bekannt ist.

```
u:=f(x)+2*x+1;
```

$$f(x)+2\,x+1$$

```
subs(x=2,u);
```

$$f(2)+5$$

Aus dem neuen Funktionssymbol f wurden zwei Ausdrücke `f(x)` und `f(2)` konstruiert.

Eine besondere Art von Funktionen stellen z.B. die MAPLE-Funktionen `expand, collect, normal` dar, die symbolische Ausdrücke $\boxed{\text{transformieren,}}$ d.h. Funktions aufrufe darstellen, deren Parameter selbst wieder Funktionsausdrücke sind und deren Wirkung auf einen Umbau dieser Ausdrücke ausgerichtet ist. Dies widerspricht dem Auswertungsverhalten von klassischen Funktionen, das "von innen nach außen" fortschreitet.

Oftmals kann man diese Trennung in Funktionsaufrufe, Funktionssymbole und Funktionstransformationen nicht sauber vornehmen, wie etwa das folgende Beispiel für die `exp`-Funktion in MAPLE zeigt:

[4]Allerdings ist in unserem Sprachgebrauch (fast) jeder symbolische Ausdruck ein solcher Funktionsausdruck.

```
exp(x);
```

$$\exp(x)$$

```
exp(2.0);
```

$$7.389056099$$

```
exp(ln(x+y));
```

$$x + y$$

Im ersten Fall erhalten wir einen Funktionsausdruck mit dem Symbol `exp` als Kopf, im zweiten Fall eine Float-Zahl, die mit einem entsprechenden Näherungsverfahren in einem Funktionsaufruf berechnet wurde, im letzten Fall dagegen ist das Ergebnis eine Funktionstransformation, in dem sich *exp* und *ln* gegenseitig "gefressen" haben.

Dasselbe Verhalten, das "mathematische" Funktionen wie `exp` oder `sin` zeigen, ist auch bei stärker "algorithmischen" Funktionen wie etwa `diff` zu beobachten, der Funktion, mit der man in MAPLE Ableitungen berechnen kann:

```
diff(sin(x),x);
```

$$\cos(x)$$

```
diff(f(x),x);
```

$$\frac{d}{dx} f(x)$$

```
h:=diff(f(g(x)),x);
```

$$\mathrm{D}(f)(g(x)) \frac{d}{dx} g(x)$$

Im ersten Beispiel wird (wenigstens auf den ersten Blick) ein Funktionsaufruf ausgeführt. Im zweiten Fall entsteht ein Funktionsausdruck, denn die etwas andere Ausgabegestaltung hat nur mit der speziellen zweidimensionalen Ausgabeformatierung von `diff(f(x),x)` zu tun. Im dritten Beispiel schließlich wird eine Funktionstransformation ausgeführt, denn hinter der angegebenen zweidimensionalen Ausgabe verbirgt sich der Ausdruck `D(f)(g(x))*diff(g(x),x)`.

Im letzten Ergebnis tritt mit dem Symbol D sogar eine Funktion auf, die keinen Funktions*ausdruck*, sondern ein reines Funktions*symbol* als Argument hat, weil an der Stelle die Ableitung der Funktion f an der Stelle $y = g(x)$ einzusetzen ist. In der Tat, ersetzen wir g durch ein "bekanntes" Funktionssymbol, so erhalten wir aus der Kettenregel gewohnte Formeln.

```
eval(subs(f=sin,h));
```

$$\cos(g(x)) \frac{d}{dx} g(x)$$

Solche "Funktionen von Funktionen" entstehen in verschiedenen mathematischen Zusammenhängen auf natürliche Weise, da auch Funktionen Gegenstand mathematischer Kalküle (etwa der Analysis) sind. Die Ableitung einer Funktion $f(x)$, die in MAPLE als `D(f)(x)` dargestellt wird, kann in MATHEMATICA und MUPAD näher an der üblichen mathematischen Notation als `f'[x]` bzw. `f'(x)` eingegeben werden. In beiden Fällen verbirgt sich dahinter aber ein Ausdruck, der dieselbe Struktur wie der aus MAPLE hat, denn `f'` ist keine neue syntaktische Einheit, sondern das Ergebnis der Anwendung des Postfix-Operators `'` auf f, wie eine Strukturanalyse zeigt.

In allen drei Systemen ist damit der Kopf des Ausdrucks `D(f)(x)` kein Symbol, sondern selbst wieder ein Ausdruck, womit die einheitliche Behandlung der Aufrufparameter und des Kopfes eines Funktionsaufrufs bei der Darstellung in Form einer geschachtelten Liste eine weitere Rechtfertigung erfährt.

"Funktionen von Funktionen" können auch ein Eigenleben führen:

```
D(sin);
```

$$\cos$$

```
D(exp+ln);
```

$$\exp\ +\ \left(a \mapsto a^{-1}\right)$$

Um die Antwort im letzten Beispiel zu formulieren, wurde eine weitere Funktion benötigt, von der nur die Zuordnungsvorschrift bekannt ist, die aber keinen Namen besitzt. Eine solche Funktion wird auch als *reine Funktion* (pure function) bezeichnet. Sie ist das Gegenteil eines Funktionssymbols, von dem umgekehrt der Name, aber keine Anwendungsvorschrift bekannt war.

Funktionsausdrücke mit booleschem Wert

Boolesche Funktionen spielen in gewissen Steuerstrukturen (`while`, `if`) eine wichtige Rolle, wobei zur korrekten Ablaufsteuerung an dieser Stelle Funktions*aufrufe* abgesetzt werden müssen, die keinen Funktions*ausdruck*, sondern (garantiert) einen der beiden Werte `true` oder `false` zurückliefern. Dies gilt vor allem für relationale Operatoren wie `=` (equal), die in vielen anderen Kontexten auch als Operator*symbole* verwendet werden.

So wird Gleichheit in den verschiedenen CAS sowohl bei der Formulierung eines Gleichungssystems als auch dessen Lösung verwendet (exemplarisch in MUPAD)

```
gls:={2*x+3*y=2, 3*x+2*y=1};

                 {3 x + 2 y = 1, 2 x + 3 y = 2}

solve(gls,{x,y});

                    {{x = -1/5, y = 4/5}}
```

In beiden Kontexten wurde `=` als Operatorsymbol verwendet, aus dem ein syntaktisches Objekt "Gleichung" als Funktionsausdruck konstruiert worden ist.

Die meisten CAS verfahren mit symbolischen Ausdrücken der Form $a = b$ auf ähnliche Weise. Eine boolesche Auswertung wird in einem booleschen Kontext automatisch vorgenommen oder kann in einigen CAS durch spezielle Funktionen (MAPLE: `evalb`, MUPAD: `bool`) erzwungen werden. Allerdings entsprechen die Ergebnisse nicht immer den Erwartungen (MUPAD, ähnlich auch die anderen CAS):

```
1=2;

                          1 = 2
```

aber

```
if 1=2 then yes else no end_if;
```

no

Sehen wir uns die boolesche Auswertung einzelner Ausdrücke näher an:

```
bool(x+y=3);
```

FALSE

```
bool(x+y=x+y);
```

TRUE

Im ersten Fall ist die Antwort für alle Variablenbelegungen $(x, y) = (t, 3 - t)$ falsch, aber danach war hier nicht gefragt. Im zweiten Fall sind linke und rechte Seite syntaktisch gleich.

```
bool(x+x=2*x);
```

TRUE

```
bool(x*(x+1)=x*x+x);
```

FALSE

```
bool(1<sqrt(5));
```

```
Error: Can't evaluate to boolean [_less]
```

```
bool(1<float(sqrt(5)));
```

TRUE

Im ersten Beispiel sind die beiden Seiten der Gleichung *nach Auswertung* syntaktisch gleich, im zweiten Fall besteht semantische, nicht aber syntaktische Gleichheit. Im dritten Beispiel schließlich konnte gar keine boolesche Auswertung vorgenommen werden, da `sqrt(5)` einen Funktionsausdruck konstruiert, von dem MuPAD nicht weiss, wie er mit 1 zu vergleichen ist. Erst der Hinweis darauf, es doch einmal mit Näherungswerten zu versuchen, führt hier zum gewünschten Ergebnis.

Funktionstransformationen

Funktionstransformationen ersetzen gewisse Kombinationen von Funktionssymbolen und evtl. speziellen Funktionsargumenten durch andere, semantisch gleichwertige Kombinationen.

Funktionstransformationen, die man damit oft auch als Simplifikationen wahrnimmt, sind eines der zentralen Designelemente eines CAS, da auf diesem Wege syntaktisch verschiedene, aber semantische gleichwertige Ausdrücke produziert werden können.

Eine solche Transformation kann darin bestehen, dass wie in obigem Beispiel, das Nacheinanderausführen einer Funktion und ihrer Inversen durch die identische Funktion ersetzt wird oder dass, wie in der Funktion `expand`, Produkte distributiv durch Summen, trigonometrische Funktionen mit Mehrfachwinkeln durch solche mit Einfachwinkeln, Potenzen mit Exponentsummen nach den Potenzgesetzen durch die entsprechenden Basisprodukte usw. ersetzt werden.

```
expand((x+1)*(x+2));
```

$$x^2 + 3\,x + 2$$

```
expand(sin(x+y));
```

$$\sin(x)\cos(y) + \cos(x)\sin(y)$$

```
expand(exp(a+sin(b)));
```

$$e^a e^{\sin(b)}$$

Charakteristisch für solche Transformationen ist die Möglichkeit, das Aufeinandertreffen von vorgegebenen Funktionssymbolen in einem Ausdruck festzustellen. Dabei wird ein Grundsatz der klassischen Funktionsauswertung verletzt: Es wird nicht nur der *Wert* der Aufrufargumente benötigt, sondern auch Information über deren *Struktur*.

So muss etwa beim Aufruf `expand(`sum_1`*`sum_2`)` die Funktion erkennen, dass ihr Argument ein Produkt ist, dessen Faktoren extrahieren und eine Funktion `expandproduct(`sum_1`,`sum_2`)` aufrufen, die ihrerseits erkennt, dass sum_1 und sum_2 Summen sind, die Summanden extrahiert, alle paarweisen Produkte zwischen den einzelnen Summanden der beiden Summen bildet und diese danach aufsummiert. Ähnlich müsste `expand(sin(`*sum*`))` das Funktionssymbol `sin` des Aufrufarguments erkennen und danach eine Funktion `expandsin` (in MAPLE heisst sie `‘expand/sin‘`) aufrufen.

> Charakteristisch für Funktionstransformationen ist also der Umstand, dass nicht nur der (semantische) Wert, sondern auch die (syntaktische) Struktur der Aufrufparameter an der Bestimmung des Rückgabewerts beteiligt ist, womit komplexere Teilstrukturen des Ausdrucks zu analysieren sind.

Funktionstransformationen sind ihrer Natur nach *rekursiv,* da nach dem Anwenden einer Transformation ein Ausdruck entstehen kann, auf den weitere Transformationen angewendet werden können. So führt MAPLE bei der Berechnung von

```
cos(exp(ln(arcsin(x))));
```

$$\sqrt{1-x^2}$$

etwa erst die Transformation $\exp(\ln(u)) = u$ und dann $\cos(\arcsin(x)) = \sqrt{1-x^2}$ aus.

Genau wie das System der Funktionsnamen muss auch das System der Funktionstransformationen zur Laufzeit erweiterbar sein, wenn es dem Nutzer gestattet werden soll, selbstdefinierte Funktionssymbole auf dieselbe Weise einzusetzen wie systemdefinierte. Dazu muss er etwa in die Lage versetzt werden, auch selbstdefinierte Paare Funktion/Inverse oder deren Ableitungen in den Transformationsmechanismus des Systems auf konsistente Weise einzubauen.

CAS verfolgen zur Realisierung solcher Funktionstransformationen zwei verschiedene Konzepte, ein *funktionales* (MAPLE, MUPAD) bzw. ein *regelbasiertes* (AXIOM, REDUCE). MACSYMA und MATHEMATICA stellen beide Mechanismen zur Verfügung.

Das funktionale Transformationskonzept

Beim funktionalen Konzept versucht man, alle Funktionsinvokationen als Funktionsaufrufe zu gestalten und die Besonderheiten von Funktionstransformationen durch eine genaue Unterscheidung zwischen (rekursiven) Funktionsaufrufen und Funktionssymbolen in den Griff zu bekommen. Da in die Abarbeitung der entsprechenden Funktion die *Struktur* der Argumente mit eingeht, werden dazu Funktionen benötigt, die diese Struktur wenigstens teilweise analysieren. MAPLE verfügt für diesen Zweck über die Funktion `type(A,T)`, die prüft, ob ein Ausdruck A den “Typ” T hat. Wie bereits an anderer Stelle erwähnt handelt es sich dabei allerdings nicht um ein strenges Typkonzept, sondern um eine Typanalyse, die nur die entsprechende syntaktische Analyse, noch dazu nur der obersten Ebene, der Struktur des entsprechenden Ausdrucks verwendet. Dies erkennen wir etwa an folgendem Beispiel:

```
exp(ln(x));
```

$$x$$

```
h:=exp(ln(x)+ln(y));
```

$$e^{\ln(x)+\ln(y)}$$

```
simplify(h);
```

$$x\,y$$

Die Struktur des Arguments verbirgt im zweiten Beispiel die Anwendbarkeit der Transformationsregel. Erst nach eingehender Analyse, die mit `simplify` angestoßen wird, gelingt die Umformung[5].

Betrachten wir als Beispiel, wie MAPLE die Exponentialfunktion definiert:

```
interface(verboseproc=2):print(`exp`);

 proc(x)
local i,t,q;
options `Copyright 1993 by Waterloo Maple Software`;
    if nargs <> 1 then ERROR(`expecting 1 argument, got `.nargs)
    elif type(x, 'complex(float)') then evalf('exp'(x))
    elif type(x, 'rational') then exp(x) := 'exp'(x)
    elif type(x, 'function') and op(0, x) = ln then exp(x) := op(1, x)
    elif type(x, 'function') and type(op(0, x), 'function') and
    op([0, 0], x) = `@` and op([0, 1], x) = ln then
        exp(x) := op([0, 2], x)(op(x))
    elif type(x, 'function') and type(op(0, x), 'function') and
    op([0, 0], x) = `@@` and op([0, 1], x) = ln then
        exp(x) := (ln@@(op([0, 2], x) - 1))(op(x))
    elif type(x, `*`) and type(- I*x/Pi, 'rational') then
        i := - I*x/Pi;
        if 1 < i or i <= -1 then
            t := trunc(i);
            t := t + irem(t, 2);
            exp(x) := exp(I*(i - t)*Pi)
        elif type(6*i, 'integer') or type(4*i, 'integer') then
            exp(x) := cos(- I*x) + I*sin(- I*x)
        else exp(x) := 'exp'(x)
        fi
    elif type(x, `*`) and nops(x) = 2 and type(op(1, x), 'rational') and
    type(op(2, x), 'function') and op([2, 0], x) = ln then
        if type(op(1, x), fraction) and irem(op([1, 2], x), 2, 'q') = 0
        then exp(x) := sqrt(op([2, 1], x))^(op([1, 1], x)/q)
        else exp(x) := op([2, 1], x)^op(1, x)
        fi
    else exp(x) := 'exp'(x)
    fi
end
```

[5]Dies gelingt auch mit `expand`, was in diesem Fall Potenzgesetze anwendet, um Exponenten*summen* zu ersetzen. MuPAD 1.4.2. war nur auf diese Weise in der Lage, die genannte Vereinfachung vorzunehmen.

Die meisten Zeilen beginnen mit `type(x, ...)`, analysieren also zuerst die Struktur des aufrufenden Arguments. Sehen wir uns die einzelnen Zeilen nacheinander an.

```
elif type(x, 'complex(float)') then evalf('exp'(x))
```

Handelt es sich um eine Float-Zahl, wird `evalf`, das numerische Interface, mit dem Argument `'exp'(x)` aufgerufen. Die Quotes stehen für die fehlende Auswertung, d.h. an dieser Stelle wird das Funktionssymbol samt Argument, also der *Funktionsausdruck* $\exp(x)$ an `evalf` übergeben. `evalf` seinerseits ist eine Funktionstransformation, die am Kopf `exp` des Aufrufparameters erkennt, dass sie die Arbeit an eine numerische Auswertungsfunktion `'evalf/exp'` (durch die Quotes entsteht ein Funktionsbezeichner!) für die Exponentialfunktion mit komplexwertigem Argument weiterzureichen hat.

Dasselbe gilt in der nächsten Zeile

```
elif type(x, 'rational') then exp(x) := 'exp'(x)
```

wenn das Argument x nämlich zu einer (ganzen oder) rationalen *Zahl* auswertet. In diesem Fall kann man den Ausdruck nicht weiter vereinfachen und es wird der entsprechende *Funktionsausdruck* zurückgeben:

```
exp(1/2);
                              exp(1/2)

exp(27/3);
                               exp(9)
```

Am letzten Beispiel sehen wir noch einmal, dass das Argument vor dem Aufruf von `exp` wirklich ausgewertet wurde. Im dritten else-Zweig

```
elif type(x, 'function') and op(0, x) = ln then exp(x) := op(1, x)
```

wird schließlich die von uns betrachtete Transformation vorgenommen: Ist x ein Funktionsausdruck, der mit `ln` beginnt, so wird das (erste und einzige) Argument dieses Funktionsausdrucks zurückgegeben. Die folgenden Zeilen dienen der Analyse von Ausdrücken, die den Funktionsiterationsoperator verwenden. Wir wollen sie übergehen. Interessant ist noch die Zeilen

```
elif type(x, `*`) and type(- I*x/Pi, 'rational') then
    i := - I*x/Pi;
    if 1 < i or i <= -1 then
        t := trunc(i);
        t := t + irem(t, 2);
        exp(x) := exp(I*(i - t)*Pi)
    elif type(6*i, 'integer') or type(4*i, 'integer') then
        exp(x) := cos(- I*x) + I*sin(- I*x)
    else exp(x) := 'exp'(x)
    fi
```

in der geprüft wird, ob es sich vielleicht um einen Ausdruck $e^{I\pi\, i}$ handelt, den man bekanntlich in die Form $cos(\pi\, i) + I\, sin(\pi\, i)$ umformen kann. Wenn nicht $-1 < i \leq 1$ gilt, wird `exp` mit einem entsprechend adjustierten Argument rekursiv aufgerufen, ansonsten geprüft, ob i ein Vielfaches von $1/6$ oder $1/4$ ist. Für die entsprechenden Winkelgrößen sind die Funktionswerte $\sin(x)$ und $\cos(x)$ bekannt und die entsprechende Transformation wird durchgeführt, ansonsten wird der Funktionsausdruck in unveränderter Form zurückgegeben.

Funktionstransformationen sind in einem solchen Konzept manchmal nur über Umwege zu realisieren, wenn Funktions*aufrufe*, also Code algorithmischer Natur, abzuarbeiten sind, da hierbei die Aufrufargumente ausgewertet werden, wobei deren ursprüngliche Struktur verändert werden kann. Möchte man etwa das Polynom $f = x^3 + x^2 - x + 1 \in \mathbf{Z}[x]$ nicht über den ganzen Zahlen, sondern modulo 2, also im Ring $\mathbf{Z}/2\mathbf{Z}[x]$, faktorisieren, so führen in MAPLE weder

```
factor(f) mod 2;
```

$$x^3 + x^2 - x + 1$$

noch

```
factor(f mod 2);
```

$$(x+1)\left(x^2+1\right),$$

zum Ziel. Das zweite Ergebnis kommt der Wahrheit zwar schon näher (Ausmultiplizieren ergibt $x^3 + x^2 + x + 1 \equiv f\ (mod\ 2)$), aber ist wegen $(x^2 + 1) \equiv (x + 1)^2\ (mod\ 2)$ noch immer falsch.

In beiden Fällen wird bei der Auswertung der Funktionsargumente die für eine Funktionstransformation notwendige Kombination der Symbole `factor` und `mod` zerstört, denn `factor` ist ein Funktions*aufruf*, der als Ergebnis ein Produkt, die Faktorzerlegung des Arguments über den ganzen Zahlen, zurückliefert. Im ersten Fall (der intern als `mod(factor(f),2)` umgesetzt wird) wird vor dem Aufruf von `mod` das Polynom (über den ganzen Zahlen) faktorisiert. Das Ergebnis enthält die Information `factor` nicht mehr. Im zweiten Fall dagegen wird das Polynom f erst modulo 2 reduziert, enthält danach die Information `mod` nicht mehr und wird deshalb wiederum ganzzahlig faktorisiert.

In klassischen Programmiersprachen werden derartige Ambiguitäten heutzutage über Polymorphie aufgelöst, was ein detailliertes Typkonzept voraussetzt. Da ein solches in CAS der zweiten Generation nicht zur Verfügung steht, müsste auf verschiedene Funktionsnamen, etwa `factor` und `factormod`, zurückgegriffen werden. Um dies wenigstens in dem Teil, der dem Nutzer sichtbar ist, zu vermeiden, führt MAPLE ein Funktions*symbol* `Factor` ein, das sein Argument unverändert zurückgibt, so dass `mod` im Aufruf

```
Factor(f) mod 2;
```

$$(x+1)^3$$

der intern sofort als `mod(Factor(f),p)` umgesetzt wird, an der speziellen Form seines ersten Arguments erkennen kann, dass modular zu faktorisieren ist und die spezielle modulare Faktorisierungsroutine `` `mod/factor`(f,p) `` aufrufen kann[6].

Ähnlich geht die MAPLE-Funktion `expand` vor. `expand(A)` analysiert die Gestalt des Ausdrucks $A = f(x_1, \ldots, x_n)$ und ruft eine entsprechende Funktion

`` `expand/f` ``$(x_1, \ldots, x_n)$

auf, die die eigentliche Transformation vornimmt. Hier wird bereits die Fähigkeit eines CAS zur Stringmanipulation verwendet, indem zur Laufzeit (!), während des Aufrufs von `expand`, aus einem in den Argumenten vorkommenden Funktionssymbol `f` ein neuer Funktionsname `expand/f` und, sofern eine entsprechende Funktion definiert ist, auch der zugehörige Funktionsaufruf generiert wird. Dies ist zugleich der Mechanismus, der es einem Nutzer erlaubt, die Fähigkeiten des Systems zu "Expandieren" für selbstdefinierte Funktionen zu erweitern.

Soll etwa l eine additive Funktion darstellen, für die `expand` die Transformation $l(x + y) = l(x) + l(y)$ ausführt, so muss man dazu eine Funktion

```
`expand/l`:= proc(y) local x;
   x:=expand(y);
   if type(x,`+`) then convert(map(l,[op(x)]),`+`)
   else l(x) fi
end;
```

[6] Die Interna sind im Systemkern verborgen.

definieren.

Einen anderen Zugang verfolgt MuPAD: Hier wird jedem Funktionssymbol, auf dem `expand` nichttrivial agieren soll, als Eigenschaft ein Funktionsattribut mit dem Namen "expand" zugeordnet, das eine entsprechende Funktion trägt. Obigen Effekt für l kann man folgendermaßen erreichen:

```
l:= funcattr(l, "expand",
  proc(x) begin x:= expand(x);
       if type(x) = "_plus" then _plus(map(op(x),l))
       else l(x)
       end_if
  end_proc);
```

Die Nachteile dieses Konzepts fallen sofort ins Auge:

1. Man hat mit jedem Funktionsaufruf eine Menge verschiedener Typinformationen zu ermitteln, womit jeder Funktionsaufruf relativ aufwendige Operationen anstösst. Deshalb ist es oft sinnvoll, bereits berechnete Funktionswerte zu speichern, wenn man weiß, dass sie sich nicht verändern.

2. Die Typanalyse ist bei vielen Funktionen von ähnlicher Bauart, so dass unnötig Code dupliziert wird.

3. Die so definierten Transformationsregeln sind relativ "starr". Möchte man z.B. das Verhalten der `exp`-Funktion dahingehend ändern, dass sie bei rein imaginärem Argument *immer* die trigonometrische Darstellung verwendet, so müsste man den gesamten oben gegebenen Code kopieren, an den entsprechenden Stellen ändern und dann die (natürlich außerdem vor Überschreiben geschützte) `exp`-Funktion entsprechend neu definieren.

Deshalb hat eine Funktionsdefinition eine komplexere Struktur als in einer klassischen Programmiersprache. In Maple etwa kann eine Funktion eine *Funktionswert-Tabelle* anlegen, in die alle bereits berechneten Funktionswerte eingetragen werden. Diese wird inspiziert, bevor die Auswertung entsprechend der Funktionsdefinition initiiert wird. In MuPAD kann eine Funktionsdefinition außerdem noch über eine Tabelle von Funktionsattributen verfügen.

Das regelbasierte Transformationskonzept

Beim regelbasierten Zugang werden die Funktionstransformationen über einen allgemeinen Simplifikationsmechanismus als direkte Transformationen des Ableitungsbaums ausgeführt. Dazu wird ein Satz von *Transformationsregeln* der Gestalt `lhs => rhs` auf den aktuellen Ausdruck A anwendet, indem grob gesprochen im Ableitungsbaum von A Teilbäume der Gestalt lhs gesucht und durch Teilbäume der Gestalt rhs ersetzt werden. Das wird so lange wiederholt, bis keine Ersetzungen mehr möglich sind. Im Gegensatz zum funktionalen Zugang sind hier also der Simplifikator und die jeweils anzuwendenden Regelsätze voneinander getrennt, was es auf einfache Weise ermöglicht, Regelsätze zu ergänzen und für spezielle Zwecke zu modifizieren und anzupassen.

Wir werden uns in einem späteren Kapitel genauer mit der Funktionsweise eines solchen Regelsystems vertraut machen. An dieser Stelle wollen wir uns nur einmal anschauen, welche Regeln Reduce zum Simplifizieren verschiedener Funktionen kennt. Die jeweiligen Regeln sind unter dem Funktionssymbol als Liste gespeichert und können mit der Funktion `showrules` ausgegeben werden:

```
showrules exp;

{exp(~x) => e^x}

showrules log;
```

```
{log(1) => 0,
 log(e) => 1,
 log(e^~x) => x,
 df(log(~x),~x) => 1/x,
 df(log(~x/~y),~z) => df(log(x),z) - df(log(y),z)}
```

Zum Verständnis der ersten Regel muss hinzugefügt werden, dass REDUCE die Exponentialfunktion damit als spezielle Potenzfunktion auffasst und Potenzfunktionen wie die anderen Arithmetikoperatoren aus Effizienzgründen außerhalb des Regelsystems "fest verdrahtet" sind. Wie bei Funktionen ist auch bei Regeln zwischen Deklaration und Anwendung zu unterscheiden. Dabei ist insbesondere auch die Verwendung von formalen Parametern denkbar, die als Platzhalter für beliebige Teilausdrücke, d.h. für entsprechende Teilbäume in den Regeln auftreten. So gilt etwa $log(exp(A)) = A$, egal wie der Teilausdruck A beschaffen ist. Solche formalen Parameter werden in REDUCE durch eine vorangestellte Tilde gekennzeichnet und sind bei einer Anwendung an *allen* Stellen durch denselben Ausdruck zu ersetzen. Regeln dieser Art nennt man $\boxed{\text{allgemeine Regeln}}$ im Gegensatz zu Regeln, wo der zu ersetzende Teilbaum *genau* mit *lhs* übereinzustimmen hat, welche man $\boxed{\text{spezielle Regeln}}$ nennt. So sind die beiden ersten Regeln für `log` spezielle, die beiden anderen allgemeine. Letztere beiden Regeln geben die Transformationsvorschriften an, wenn die Funktionssymbole `log` und `exp` bzw. `df` (für die Ableitung) und `log` irgendwo im Ableitungsbaum aufeinanderfolgen.

Ein solches Regelsystem kann durchaus einen größeren Umfang erreichen:

```
showrules sin;

{sin(pi) => 0,
 sin(pi/2) => 1,
 sin(pi/3) => sqrt(3)/2,
 sin(pi/4) => sqrt(2)/2,
 sin(pi/6) => 1/2,
 sin((5*pi)/12) => sqrt(2)/4*(sqrt(3) + 1),
 sin(pi/12) => sqrt(2)/4*(sqrt(3) - 1),
 sin((~(~ x)*i)/~(~ y)) => i*sinh(x/y) when impart(y)=0,
 sin(atan(~u)) => u/sqrt(1 + u**2),
 sin(2*atan(~u)) => 2*u/(1 + u**2),
 sin(~n*atan(~u)) => sin((n - 2)*atan(u))*(1 - u**2)/(1 + u**2)
        + cos((n - 2)* atan(u))*2*u/(1 + u**2)
                when fixp(n) and n>2,
 sin(acos(~u)) => sqrt(1 - u**2),
 sin(2*acos(~u)) => 2*u*sqrt(1 - u**2),
 sin(2*asin(~u)) => 2*u*sqrt(1 - u**2),
 sin(~n*acos(~u)) => sin((n - 2)*acos(u))*(2*u**2 - 1)
        + cos((n - 2)*acos(u))*2* u*sqrt(1 - u**2)
                when fixp(n) and n>2,
 sin(~n*asin(~u)) => sin((n - 2)*asin(u))*(1 - 2*u**2)
        + cos((n - 2)*asin(u))*2* u*sqrt(1 - u**2)
                when fixp(n) and n>2,
 sin((~x + ~(~ k)*pi)/~d) => sign(k/d)*cos(x/d)
                when x freeof pi and abs(k/d)=1/2,
 sin((~(~ w) + ~(~ k)*pi)/~(~ d)) =>
        (if evenp(fix(k/d)) then 1 else  - 1)
        *sin(( w + remainder(k,d)*pi)/d)
                when w freeof pi and ratnump(k/d) and abs(k/d)>=1,
 sin((~(~ k)*pi)/~(~ d)) => sin((1 - k/d)*pi)
```

```
                when ratnump(k/d) and k/d>1/2,
 sin(asin(~x)) => x,
 df(sin(~x),~x) => cos(x)}$
```

Manche der angegebenen Regeln sind noch konditional untersetzt, d.h. nur dann anwendbar, wenn die in der Definition verwendeten formalen Parameter mit aktuellen Parametern belegt sind, die noch Zusatzvoraussetzungen erfüllen. Diese Effekte sind von regelorientierten Programmiersprachen wie etwa Prolog aber gut bekannt.

Diese Regeln werden automatisch angewendet, wenn REDUCE das Funktionssymbol `sin` in einem Ausdruck antrifft. So werden etwa (Regel 9 – 11) Ausdrücke der Form $\sin(n \cdot \arctan(x))$ aufgelöst:

```
sin(5*atan(x));
```

$$\frac{x^5 - 10\,x^3 + 5\,x}{\sqrt{x^2+1}\,(x^4 + 2\,x^2 + 1)}$$

Das selbe Ergebnis kann man mit MAPLE erreichen, da die Definition von `sin` die Information enthält, dass

$$\sin(\arctan(x)) = \frac{x}{\sqrt{1+x^2}}$$

gilt, wie wir an folgendem Quelltextfragment erkennen:

```
print(sin);
proc(x)
    local n, t;
    ...
        elif type(x, 'function') and nops(x) = 1 then
            n := op(0, x);
            t := op(1, x);
    ...
            elif n = 'arctan' then t/sqrt(1 + t^2)
    ...
    end
```

Allerdings muss dazu über `expand` erst eine Funktionstransformation angestoßen werden, die die Funktionssymbole `sin` und `arctan` unmittelbar zusammenbringt, indem die Mehrfachwinkel-Ausdrücke aufgelöst werden.

```
sin(5*arctan(x));
```

$$\sin(5\ \arctan(x))$$

```
expand(%);
```

$$16\,\frac{x}{\left(1+x^2\right)^{5/2}} - 12\,\frac{x}{\left(1+x^2\right)^{3/2}} + \frac{x}{\sqrt{1+x^2}}$$

```
normal(%);
```

$$\frac{x\left(5 - 10\,x^2 + x^4\right)}{\left(1+x^2\right)^{5/2}}$$

In beiden Konzepten sind aus Effizienzgründen oft Teile der Transformationen im Systemkern fest eingebrannt. Dies gilt insbesondere, auch aus anderen Gründen, für die Polynomarithmetik.

3.6 Steuerstrukturen im symbolischen Rechnen

Die größte Ähnlichkeit mit klassischen Programmiersprachen weist ein CAS im Bereich der Steuerstrukturen auf. Es werden in der einen oder anderen Form alle für eine imperative Programmiersprache üblichen Steuerstrukturen (Anweisungsfolgen, Verzweigungen, Schleifen) zur Verfügung gestellt, die sich selbst in der Syntax an gängigen Vorbildern wie PASCAL oder C orientieren. Auch Unterprogrammtechniken stehen zur Verfügung, wie wir im Abschnitt "Funktionen" bereits gesehen hatten.

Da (geschachtelte) Listen eine zentrale Stellung im Datentypdesign von CAS einnehmen, wird für diese auch ein ausreichendes Repertoire an Funktionalität zur Verfügung gestellt. Dabei ist zu beachten, dass Maple und MuPAD die etwa in einem Funktionsaufruf auftretende, kommaseparierte *Argumentliste* (expression sequence) als grundlegenden Datentyp verwenden, aus dem durch entsprechende Funktionen Listen, Mengen und andere Datenstrukturen erstellt werden können, die anderen CAS dagegen direkt mit auch dem Nutzer zugänglichen Listen als grundlegendem Datentyp operieren.

Um mit solchen Listen umzugehen wird erst einmal ein Zugriffsoperator auf einzelne Listenelemente, evtl. auch über mehrere Ebenen hinweg, benötigt, der gewöhnlich `part` (als Teil einer Liste) oder `op` (für Operand in einer Argumentliste) heißt. Die meisten Systeme stellen auch eine iterierte Version zur Verfügung, mit der man tiefergelegene Teilausdrücke in einer geschachtelten Liste selektieren kann.

Mit diesen Operatoren wäre eine Listentraversion nun mit der klassischen `for`-Anweisungen möglich, etwa als

```
> for i from 1 to nops(l) do ... something with op(l,i) ...
```

wobei `nops(l)` die Länge der Liste l zurückgibt und mit `op(l,i)` auf die einzelnen Listenelemente zugegriffen wird. Aus naheliegenden Effizienzgründen stellen die meisten CAS jedoch eine spezielle Listentraversion der Form

```
> for x in l do ... something with x ...
```

zur Verfügung.

Daneben spielt die Listenmanipulation eine wichtige Rolle, insbesondere deren Generierung, selektive Generierung und uniforme Manipulation. Zur Erzeugung von Listen gibt es in allen CAS einen *Sequenzierungsoperator*, der eine Liste aus einzelnen Elementen nach einer Bildungsvorschrift generiert. In Maple und MuPAD wird dafür ein eigener Operator verwendet. Reduce hat die Syntax von `for` so erweitert, dass ein Wert zurückgegeben wird.

Maple:

```
u:=[seq(i^2,i=1..5)];
```

$$u := [1, 4, 9, 16, 25]$$

MuPAD:

```
u:=[i^2 $ i=1..5];
```

$$u := [1, 4, 9, 16, 25]$$

Reduce:

```
u:=for i:=1:5 collect i^2;
```

$$u := \{1, 4, 9, 16, 25\}$$

Bei der $\boxed{\text{selektiven Listengenerierung}}$ möchte man aus einer bereits vorhandenen Liste alle Elemente mit einer gewissen Eigenschaft auswählen. Maple, MuPAD und Mathematica haben dafür einen eigenen Select-Operator, der als Argumente eine boolesche Funktion und eine Liste nimmt und die gewünschte Teilliste zurückgibt. Reduce nutzt für diesen Zweck die erweiterte `for`-Syntax, mit der man auch iteriert erzeugte Listen zusammenhängen kann, sowie `if`-Anweisungen, die einen Wert zurückgeben.

Betrachten wir etwa, wie man die Primzahlen bis 50 in einer Liste aufsammeln kann:

Maple (aehnlich MuPAD):

```
select(isprime,[$1..50]);
```

$$[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47]$$

Mathematica:

```
Select[Range[1,50],PrimeQ]
```

$$\{2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47\}$$

Macsyma:

```
sublist(1..50,primep);
```

$$[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47]$$

Reduce:

```
for i:=1:50 join if primep(i) then {i} else {};
```

$$\{2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47\}$$

$\boxed{\text{Uniforme Listenmanipulationen}}$ treten immer dann auf, wenn man auf alle Elemente einer Liste ein und dieselbe Funktion anwenden möchte. Viele der Systeme gehen bei Funktionen, die auf Listen angewendet werden, jedoch nur für deren einzelne Elemente einen Sinn ergeben, davon aus, dass die Funktion auf die einzelnen Elemente anzuwenden ist. So liefert etwa MuPAD

```
u:=[sin(i) $ i=1..3];
```

$$[\sin(1), \sin(2), \sin(3)]$$

```
float(u);
```

$$[0.8414709848, 0.9092974268, 0.141120008]$$

wobei offensichtlich die Funktionstransformation `float` $\circ$ `list` $\longrightarrow$ `list` $\circ$ `float` vorgenommen worden ist. Dort, wo dies nicht automatisch geschieht, kann die Funktion `map` eingesetzt werden, die als Argumente eine Funktion f und eine Liste l nimmt und die Funktion auf alle Listenelemente anwendet.

```
u:=[1,2,3];
```

$$[1,2,3]$$

```
sin(u);
```

$$\sin([1,2,3])$$

aber

```
map(u,sin);
```

$$[\sin(1),\sin(2),\sin(3)]$$

In einem Anhang zu diesem Kapitel sind die wichtigsten Operationsmöglichkeiten auf Listen in den einzelnen CAS zusammengestellt.

Wir hatten in den Beispielen bereits gesehen, dass in REDUCE

$$\boxed{\text{Steuerstrukturen einen Wert zurückgeben.}}$$

Dieses Verhalten zeigen die meisten der Systeme (ausgenommen MAPLE, MUPAD und teilweise MATHEMATICA). Es zeigt sich, dass die spezielle Art einer Steuerstruktur es sogar erlaubt, diese selbst als Funktionsaufruf zu implementieren, etwa wie in MATHEMATICA:

```
> While[test,body]
```

führt die Anweisungsfolge `body` so lange aus, bis `test` den Wert *true* zurückgibt. Dies eröffnet weitere Möglichkeiten der Unifizierung von Funktions- und Steuerstrukturen.

Substitutionslisten

Mehrere der Konstrukte, die wir in diesem Kapitel kennengelernt haben, spielen in Substitutionslisten zusammen, die die Mehrzahl der CAS als Ausgabeform des `solve`-Operators verwendet.

Betrachten wir etwa die Ausgabe, die MAPLE beim Lösen des Gleichungssystems

$$\{x^2+y=2,\ y^2+x=2\}$$

produziert:

```
polys:={x^2+y-2, y^2+x-2};
s:=[solve(polys,{x,y})];
```

$$\begin{gathered}[\{y=-2,x=-2\},\ \{y=1,x=1\},\\ \{y=\mathit{RootOf}(\_Z^2-\_Z-1),x=1-\mathit{RootOf}(\_Z^2-\_Z-1)\}]\end{gathered}$$

MAPLE verwendet aus Gründen, die wir später noch kennenlernen werden, hier selbst Quadratwurzeln nicht von sich aus. Wir wenden deshalb noch auf jeden einzelnen Eintrag der Liste `s` die Funktion `allvalues` an, die einen ROOTOF-Ausdruck, hinter dem sich mehrere Nullstellen verbergen, in die entsprechenden Wurzelausdrücke oder, wenn dies nicht möglich ist, in numerische Näherungslösungen aufspaltet.

```
s:=map(allvalues,s);
```

$$\begin{array}{l}[\{y=-2,x=-2\},\ \{y=1,x=1\},\\ \{y=1/2\sqrt{5}+1/2,x=1/2-1/2\sqrt{5}\},\\ \{y=1/2-1/2\sqrt{5},x=1/2\sqrt{5}+1/2\}]\end{array}$$

Als Ergebnis erhalten wir eine Liste `s` von 4 Lösungen. Deren mathematisch ansprechende Form (Verwendung des Gleichheitszeichens) ist auch aus programmiertechnischer Sicht günstig. Wir können jeden einzelnen Eintrag der Liste als lokale Variablensubstitution in komplexeren Ausdrücken verwenden. Wollen wir etwa durch die Probe die Richtigkeit der Rechnungen prüfen, so können wir nacheinander jeden Listeneintrag aus `s` in `polys` substituieren und nachfolgend vereinfachen:

```
for v in s do print(simplify(subs(v,polys))) od;
```

oder noch einfacher

```
map(v->simplify(subs(v,polys)),s);
```

$$[\{0\},\ \{0\},\ \{0\},\ \{0\}]$$

Wir haben hierbei wesentlich davon Gebrauch gemacht, dass bis auf Mathematica die einzelnen CAS nicht zwischen der mathematischen Relation $A = B$ (`A equal B`) und dem Substitutionsoperator $x = A$ (`x replaceby A`) unterscheiden. Eine solche Liste `s` bezeichnet man deshalb als *Substitutionsliste.*

Macsyma, Maple, Mathematica, MuPAD und Reduce geben ihre Lösungen für Gleichungssysteme in mehreren Variablen als solche Substitutionslisten zurück, ebenso Axiom, wenn das Gleichungssystem exakt gelöst werden kann[7]. Macsyma, Mathematica und Reduce verwenden diese Darstellung auch für Gleichungen in einer Variablen, Maple und MuPAD dagegen in diesem Fall nur, wenn Gleichungen und Variablen als einelementige *Mengen* angegeben werden.

[7] Axiom gibt stets eine Liste von *Gleichungen* zurück, die ob des Zusammenfallens von Gleichheits- und Substitutionssymbols für explizite Lösungen als Substitutionsliste verwendet werden kann.

Anhang: Operationen auf Listen in den einzelnen CAS

Im Folgenden werden die wichtigsten Listenoperationen für die einzelnen CAS, die wir betrachten, an exemplarischen Beispielen demonstriert. Die Beispiele sind im Einzelnen

Zugriff:	Zugriff auf das Element i der Liste l
Erzeugung:	Erzeugung einer Liste $\{f(i),\ i=a,\ldots,b\}$
Selektion:	Liste der Primzahlen bis 50
Mapping:	Wende die Funktion f auf alle Elemente der Liste l an.

Axiom

Zugriff	`l.i`
Erzeugung	`[f(i) for i in a..b]`
Selektion	`[i for i from 1 to 50 | prime?(i)]`
Mapping	`map(f,l)`

Macsyma

Zugriff	`part(l,i)`
Erzeugung	`makelist(f(i),i,a,b)` oder `map(f,a..b)`
Selektion	`sublist(1..50,primep)`
Mapping	`map(f,l)`

Maple

Zugriff	`op(i,l)`
Erzeugung	`[seq(f(i),i=a..b)]`
Selektion	`select(isprime,[seq(i, i=1..50)])`
Mapping	`map(f,l)`

Mathematica

Zugriff	`Part[l,i]`
Erzeugung	`Table[f(i),i,{a,b}]` oder `Map[f,Range[a,b]]`
Selektion	`Select[Range[1,50],PrimeQ]`
Mapping	`Map[f,l]`

MuPAD

Zugriff	`op(l,i)`
Erzeugung	`[f(i)$ i=a..b]`
Selektion	`select([$1..50],isprime)`
Mapping	`map(l,f)`

Reduce

Zugriff	`part(l,i)`
Erzeugung	`for i:=a:b collect f(i)`
Selektion	`for i:=1:50 join if primep(i) then {i} else {}`
Mapping	`map(f,l)`

Kapitel 4

Das Simplifizieren von Ausdrücken

Eine wichtige Eigenschaft von CAS ist die Möglichkeit, *zielgerichtet* Ausdrücke in eine semantisch gleichwertige, aber syntaktisch verschiedene Form zu transformieren. Derartige Transformationen wollen wir im folgenden als $\boxed{\text{Simplifikation}}$ bezeichnen.

Neben einigen allgemeingültigen Vereinfachungen wie etwa

> das Zusammenfassen gleichartiger Summanden in einem polynomialen Ausdruck
>
> oder Vereinfachungen der Art
>
> $$\begin{array}{rcl} \sin(\arcsin(x)) & \to & x \\ \sin(\arctan(x)) & \to & \frac{x}{\sqrt{x^2+1}} \\ abs(abs(x)) & \to & abs(x) \end{array}$$

können dabei an verschiedenen Stellen einer Rechnung Transformationen mit unterschiedlichen Intentionen und sogar einander widersprechenden Zielvorgaben erforderlich sein. Zur Berechnung des Integrals

$$\int \log\left(\frac{x+1}{x-1}\right)\, d\,x = (x+1)\log(x+1) - (x-1)\log(x-1)$$

und zur Lösung der Gleichung

$$\log(x+1) - \log(x-1) = 1 \quad \Leftrightarrow \quad x = \frac{e+1}{e-1}$$

sind etwa die Logarithmengesetze in jeweils unterschiedlicher Richtung anzuwenden. Ähnlich kann man polynomiale Ausdrücke expandieren oder aber in faktorisierter Form darstellen, Basen in Potenzfunktionen zusammenfassen oder aber trennen, Additionstheoreme anwenden, um trigonometrische Ausdrücke eher als Summen oder eher als Produkte darzustellen, die Gleichung $\sin(x)^2 + cos(x)^2 = 1$ verwenden, um eher `sin` durch `cos` oder eher `cos` durch `sin` zu ersetzen usw.

Das Simplifikationssystem eines CAS muss deshalb genügend flexibel ausgelegt sein, um auch einen derartigen Wechsel zwischen verschiedenen Simplifikationsstrategien zu ermöglichen.

4.1 Simplifikationssysteme

Um einen solchen Anspruch zu realisieren, werden im Design der CAS zwei verschiedene Strategien verfolgt. Unter der ersten Strategie, $\boxed{\text{Simplifikation nach Aufforderung,}}$ nimmt das CAS eine Reihe allgemeingültiger Simplifikationen, die allen Simplifikationsstrategien gemeinsam sind, automatisch vor. Weitergehende Simplifikationen bedürfen expliziter Aktivität des Nutzers, dem dafür verschiedene Funktionen wie `expand`, `collect`, `factor` oder `normal` zur Verfügung stehen, die verschiedene, häufig erforderliche, aber fest vorgegebene Simplifikationsstrategien (Ausmultiplizieren, Zusammenfassen von Termen nach gewissen Prinzipien, Anwendung von Additionstheoremen für Winkelfunktionen, Anwendung von Potenz- und Logarithmengesetzen usw.) realisieren. Daneben existiert meist eine (oder mehrere) komplexere Funktion `simplify`, die das Ergebnis verschiedener Transformationsstrategien miteinander vergleicht und an Hand des Ergebnisses entscheidet, welches denn nun das "einfachste" ist.

Diese Strategie wird von den Systemen Derive, Maple, Mathematica und MuPAD verwendet. Ihr Vorteil ist die leichte Änderbarkeit der Simplifikationsrichtung im Laufe des interaktiven Dialogs. Der Nachteil dieses Herangehens besteht in der relativen Starrheit des Simplifikationssystems, womit ein Abweichen von den fest vorgegebenen Simplifikationsstrategien nur unter erheblichem Aufwand möglich ist.

Reduce sowie in gewissem Sinne auch Axiom verfolgen die zweite Strategie, die der

$$\boxed{\text{vollständigen automatischen Simplifikation.}}$$

Die Zielrichtung der Simplifikation wird dabei durch einen *gültigen Kontext* bestimmt. Ein eingegebener Ausdruck wird automatisch nach allen im entsprechenden Kontext gerade aktiven Simplifikationsregeln umgeformt. Simplifikation erfolgt also über einen *allgemeinen Simplifikationsmechanismus*, der mit unterschiedlichen Regelsystemen als den jeweiligen "Daten" zu unterschiedlichen Ergebnissen führt. Die Änderung der Simplifikationsrichtung erfolgt durch Veränderung des gültigen Kontexts, der durch den Wert globaler Variablen, sogenannter *Schalter*, bestimmt ist, die man frei untereinander kombinieren kann und die das Zu- bzw. Abschalten bestimmter Simplifikationseffekte bewirken. Auf diese Weise sind Simplifikationsstrategien wesentlich flexibler formulierbar. Allerdings besteht der Nachteil dieses Zugangs darin, dass man in der Vielzahl der verschiedenen Schalter schnell den Überblick über den jeweils aktuell gültigen Kontext verlieren kann.

Ein größeres Problem bei diesem Zugang ist der Umgang mit einander widersprechenden Simplifikationsstrategien, da man diese nicht gleichzeitig wirken lassen darf, wenn man Endlosschleifen vermeiden möchte. Was im ersten Zugang durch das Design der verschiedenen systemspezifischen Simplifikationsfunktionen gewährleistet wurde, kann hier bei einer *beliebigen* Kombinierbarkeit der verschiedenen Schalter *prinzipiell* nicht gewährleistet werden. So kann man die Beziehung $log(x*y) = log(x) + log(y)$ als Transformationsregel

$$log(x*y) \mapsto log(x) + log(y)$$

aber auch als

$$log(x) + log(y) \mapsto log(x*y)$$

anwenden. Verwendet man einen Schalter, um zwischen beiden Strategien *umzuschalten*, so wird das System einen Ausdruck der Form $\log(u \cdot v) + \log(x) + \log(y)$ immer umformen. Da auch solche Ausdrücke manchmal eine Daseinsberechtigung haben, stellt Reduce zwei Schalter `combinelogs` und `expandlogs` zur Verfügung, mit denen man das entsprechende Simplifikationsverhalten ein- oder ausschalten kann. Würde man beide gleichzeitig aktivieren, so würde eine unendliche Simplifikationsschleife eintreten. Dies kann man vermeiden, indem das Einschalten des einen Schalters den anderen automatisch ausschaltet[1].

Wir sehen an diesem Beispiel zugleich, dass es in einem solchen Ansatz schwierig ist, gezielte *teilweise* Simplifikationen vorzunehmen.

Macsyma verfolgt eine Mischung aus beiden Strategien, indem sowohl Schalter zur Beeinflussung des Simplifikationsverhaltens als auch verschiedene Simplifikationsfunktionen zur Verfügung stehen.

Die Vorteile konsistenter Simplifikationssysteme und deren flexible Einsetzbarkeit werden, ähnlich den lokalen Wertzuweisungen an Variablen, durch *lokale Simplifikationssysteme* miteinander kombiniert. Über einfache Mechanismen zur Erstellung und Anwendung lokaler Simplifikationssysteme verfügen derzeit nur die Systeme Axiom, Mathematica und Reduce. Reduce nutzt

[1] In der gegenwärtigen Version 3.6 von Reduce können allerdings beide Schalter nebeneinander existieren. Sind beide aktiviert, so wird zwischen beiden Darstellungen bei jedem Simplifikationszyklus gewechselt.

dabei auf elegante Weise denselben zentralen Simplifikationsmechanismus wie auch für die interne Simplifikation, indem dieser mit *lokalen Regelsystemen* gefüttert wird, die dieselbe Struktur haben wie die dem System bekannten Regeln für Funktionssymbole, wie wir sie mit `showrules` extrahiert hatten.

4.2 Simplifikation und mathematische Exaktheit

Wir hatten bereits gesehen, dass es in beiden Zugängen zur Simplifikationsproblematik einen

$$\boxed{\text{Kern von allgemeingültigen Simplifikationen}}$$

gibt, die allen Simplifikationsstrategien gemeinsam sind und deshalb stets automatisch ausgeführt werden[2].

Dazu gehört zunächst einmal die Strategie, spezielle Werte von Funktionsausdrücken, sofern diese durch "einfachere" Symbole exakt ausgedrückt werden können, durch diese zu ersetzen.

Beispiele (Maple):

```
sqrt(36);
```

$$6$$

```
sin(Pi/4);
```

$$1/2\sqrt{2}$$

```
tan(Pi/6);
```

$$1/3\sqrt{3}$$

```
arcsin(1);
```

$$1/2\,\pi$$

Dies trifft auch für kompliziertere Funktionsausdrücke zu, die auf "elementarere" Funktionen zurückgeführt werden, in denen mehr oder weniger gut studierte spezielle mathematische Funktionen auftreten.

```
GAMMA(1/2);
```

$$\sqrt{\pi}$$

```
int(exp(-x^2),x=0..infinity);
```

$$1/2\sqrt{\pi}$$

```
int(exp(-x^2),x=0..y);
```

$$1/2\,erf(y)\sqrt{\pi}$$

```
sum(1/i^2,i=1..infinity);
```

[2] Derive nimmt initial allerdings überhaupt keine Simplifikationen vor.

$$1/6\,\pi^2$$

```
sum(1/i^n,i=1..infinity);
```

$$\zeta(n)$$

In den Beispielen wurden die Gamma-Funktion $\Gamma(x)$, die Gaußsche Fehlerfunktion $erf(x)$ sowie die Riemannsche Zeta-Funktion $\zeta(n)$ verwendet.

Weiterhin wird auch eine Reihe komplizierterer Umformungen von einigen der Systeme automatisch ausgeführt wie z.B. (MAPLE):

```
sqrt(24);
```

$$2\sqrt{6}$$

```
sqrt(2*sqrt(3)+4);
```

$$\sqrt{3}+1$$

```
sqrt(11+6*sqrt(2))+sqrt(11-6*sqrt(2));
```

$$6$$

Auch werden eindeutige Simplifikationen von Funktionsausdrücken ausgeführt wie etwa

```
abs(abs(x));
```

$$|x|$$

```
tan(arctan(x));
```

$$x$$

```
tan(arcsin(x));
```

$$\frac{x}{\sqrt{1-x^2}}$$

```
abs(-Pi*x);
```

$$\pi\,|x|$$

```
cos(-x);
```

$$\cos(x)$$

```
exp(3*ln(x));
```

$$x^3$$

Auf den ersten Blick mag es deshalb verwundern, dass folgende Ausdrücke nicht vereinfacht werden:

```
sqrt(x^2);
```

$$\sqrt{x^2}$$

```
ln(exp(x));
```

$$\ln(e^x)$$

```
arctan(tan(x));
```

$$\arctan(\tan(x))$$

In jedem der drei Fälle würde der durchschnittliche Nutzer als Ergebnis wohl x erwarten. Für die letzte Beziehung ist das allerdings vollkommen falsch, denn *arctan* ist nur im Intervall $[-\pi/2, \pi/2]$ die Umkehrfunktion von *tan*. Die korrekte Antwort lautet für $x \in \mathbf{R}$ also

$$\arctan(\tan(x)) = x - \left[\frac{x}{\pi} + \frac{1}{2}\right] \cdot \pi,$$

wobei $[a]$ für den ganzen Teil der Zahl $a \in \mathbf{R}$ steht.

Dass auch $\sqrt{x^2} = x$ mathematisch nicht exakt ist, dürfte bei einigem Nachdenken ebenfalls einsichtig sein und als Ergebnis der Simplifikation $|x|$ erwartet werden. Diese Antwort wird auch von Reduce gegeben. Maple allerdings gibt nach expliziter Aufforderung

```
simplify(sqrt(x^2));
```

$$csgn(x)x$$

zurück, obwohl wir in den Beispielen gesehen hatten, dass es auch die Betragsfunktion kennt. Der Grund liegt darin, dass das naheliegende Ergebnis $|x|$ nur für *reelle* Argumente korrekt ist, nicht dagegen für komplexe. Für komplexe Argumente ist die Wurzelfunktion mehrwertig, so dass $\sqrt{x^2} = \pm x$ eine korrekte Antwort wäre. Da man in diesem Fall oft vereinbart, dass der Wert der Wurzel der Hauptwert ist, also derjenige, der in der oberen komplexen Halbebene liegt, wird hier die *komplexe Vorzeichenfunktion* `csgn` verwendet. In diesem Kontext ist auch die Vereinfachung des Ergebnisses zu $|x|$, dem Betrag der komplexen Zahl x, fehlerhaft.

Für noch allgemeinere mathematische Strukturen, in denen Multiplikationen und deren Umkehrung definiert werden können, wie etwa Gruppen (quadratische Matrizen oder ähnliches), ist allerdings selbst diese Simplifikation nicht korrekt. In einem vollkommen rigorosen Verständnis dürfte also $\sqrt{x^2}$ ohne weitere Annahmen über den Wertebereich, aus dem das Argument genommen wird, überhaupt nicht simplifiziert werden.

Die dritte Beziehung $\ln(\exp(x)) = x$ ist wegen der Monotonie der beteiligten Funktionen dagegen für alle reellen Werte von x richtig. Für komplexe Argumente kommt dagegen, ähnlich wie für die Wurzelfunktion, die Mehrdeutigkeit der Logarithmusfunktion ins Spiel.

Weitere interessante Simplifikationsfälle sind die Ausdrücke

```
sqrt(1/x)-1/sqrt(x);

sqrt(x*y)-sqrt(x)*sqrt(y);
```

die für solche Argumente, für die sie "sinnvoll" definiert sind (also hier etwa für positive reelle x), zu Null vereinfacht werden können. Für negative reelle Argumente haben wir in diesem Fall allerdings nicht nur die Mehrdeutigkeit der Wurzelfunktion im Bereich der komplexen Zahlen zu berücksichtigen, sondern kollidieren mit anderen, wesentlich zentraleren Annahmen, wie etwa der automatischen Ersetzung von $\sqrt{-1}$ durch die imaginäre Einheit I. Setzen wir in obigen Ausdrücken

$x = y = -1$ und führen diese Ersetzung aus, so erhalten wir im ersten Fall $I - 1/I = 2\,I$ und im zweiten Fall $\sqrt{1 - I^2} = 2$. Solche Inkonsistenzen tief in komplexen Berechnungen versteckt können zu vollkommen falschen Resultaten führen, ohne dass der Grund dafür offensichtlich wird.

Natürlich sind Simplifikationssysteme mit zu rigiden Annahmen für die meisten Anwendungszwecke untauglich, wenn sie derart simple Umformungen "aus haarspalterischen Gründen" nicht oder nur nach gutem Zureden ausführen. Jedes der CAS muss deshalb für sich entscheiden, auf welchen Grundannahmen seine Simplifikationen sinnvollerweise basieren, um ein ausgewogenes Verhältnis zwischen mathematischer Exaktheit einerseits und Praktikabilität andererseits herzustellen.

In der folgenden Tabelle sind die Simplifikationsergebnisse der verschiedenen CAS (in der Grundeinstellung) auf einer Reihe von Beispielen zusammengestellt (u bedeutet unsimplifiziert):

Ausdruck	Axiom	Derive	Macsyma	Maple	MMa	MuPAD	Reduce
$\lvert-\pi\cdot x\rvert$	$\lvert\pi\cdot x\rvert$	$\pi\cdot\lvert x\rvert$	$\pi\cdot\lvert x\rvert$	$\pi\cdot\lvert x\rvert$	$\pi\cdot\lvert x\rvert$	$\pi\cdot\lvert x\rvert$	$\pi\cdot\lvert x\rvert$
$\arctan(\tan(x))$	x	u	u	u	u	u	u
$\arctan(\tan(\frac{25}{4}\pi))$	$\pi/4$	$\pi/4$	$\pi/4$	$\pi/4$	$\pi/4$	$\pi/4$	$\pi/4$
$\sqrt{x^2}$	u	$\lvert x\rvert$	$\lvert x\rvert$	u	u	u	$\lvert x\rvert$ [3]
$\sqrt{x\,y} - \sqrt{x}\,\sqrt{y}$	u	u	u	u	u	u	u
$\sqrt{\frac{1}{z} - \frac{1}{\sqrt{z}}}$	0	u	u	u	u	u	u
$\ln(\exp(x))$	x	u	x	u	u	u	x
$\ln(\exp(10 * I))$	u	-2i(2π-5)	i(10-4π)	u	10i-4πi	-4πi+10i	10i

Tabelle 1: Simplifikationsverhalten der verschiedenen Systeme an ausgewählten Beispielen

Macsyma, Maple und nun auch teilweise MuPAD besitzen mit dem `assume`-Mechanismus die Möglichkeit, einzelnen Symbolen *Eigenschaften* zuzuordnen, die beim Simplifizieren in die Entscheidungsfindung mit einbezogen werden. Solche Eigenschaften können Annahmen über die Natur einer Variablen (`assume(x,integer)`, `assume(x,real)`), die Zugehörigkeit zu einem reellen Intervall (`assume(x>0)`), die spezielle Natur einer Matrix (`assume(m,quadratic)`) oder andere Eigenschaften sein. Symbole, für die Eigenschaften definiert sind, werden in Maple mit einer Tilde versehen. Einzelne Funktionen sind in der Lage, aus Eigenschaften ihrer Argumente Eigenschaften der Funktionswerte abzuleiten und weiterzugeben. Ähnlich können in Derive Annahmen (positiv, nichtnegativ, reell, komplex, aus einem reellen Intervall) über die Natur einzelner Variablen getroffen werden.

```
> assume(y,real);
> about(y);
Originally y, renamed y~:
  is assumed to be: real

> assume(x>0);
> about(x);
Originally x, renamed x~:
  is assumed to be: RealRange(Open(0),infinity)

> abs(-Pi*x);
```

$$\pi\,x\tilde{}$$

```
> sqrt(x^2);
```

$$x\tilde{}$$

[3]wieder x in Reduce 3.7

```
> sqrt(x*y)-sqrt(x)*sqrt(y);
```

$$\sqrt{x\tilde{}\,y\tilde{}} - \sqrt{x\tilde{}}\,\sqrt{y\tilde{}}$$

```
> simplify(%);
```

$$0$$

```
> sqrt(1/x)-1/sqrt(x);
```

$$0$$

```
> ln(exp(y));
```

$$y\tilde{}$$

In MACSYMA und REDUCE lässt sich außerdem die Strenge der mathematischen Umformungen durch verschiedene Schalter verändern. So kann man etwa in REDUCE über den (allerdings nicht dokumentierten) Schalter `reduced` die klassischen Vereinfachungen von Wurzelsymbolen, die für komplexe Argumente zu fehlerhaften Ergebnissen führen können, zulassen. In MACSYMA können die entsprechenden Schaltern wie `logexpand, radexpand` oder `triginverses` sogar drei Werte annehmen: `false` (keine Simplifikation), `true` (bedachtsame Simplifikation) oder `all` (ständige Simplifikation).

Die mathematische Strenge der Rechnungen, die mit einem CAS allgemeiner Ausrichtung möglich sind, ist in den letzten Jahren stärker ins Blickfeld der Entwickler geraten. Die Konzepte der verschiedenen Systeme sind unter diesem Blickwinkel mehrfach geändert worden, was man an den differierenden Ergebnissen verschiedener Vergleiche, die zu unterschiedlichen Zeiten angefertigt wurden ([19, 18, 23, 25]), erkennen kann. Allerdings werden selbst innerhalb eines Systems an unterschiedlichen Stellen manchmal unterschiedliche Maßstäbe der mathematischen Strenge angelegt, insbesondere bei der Implementierung komplexerer Verfahren. Ein abschließendes Urteil ist deshalb nicht möglich.

Für eine umfassendere Betrachtung dieser Problematik, die auch tiefergehende mathematische Fähigkeiten der verschiedenen Systeme in einen Vergleich einbezieht, sei dazu auf [18] verwiesen.

4.3 Das allgemeine Simplifikationsproblem

Wir hatten gesehen, dass sich das allgemeine Simplifikationsproblem grob als

$$\boxed{\text{das Erkennen der semantischen Gleichwertigkeit syntaktisch verschiedener Ausdrücke}}$$

charakterisieren lässt. Wir wollen nun versuchen, dies begrifflich genauer zu fassen.

Wir betrachten dazu die Menge $\mathcal{E}$ linguistischer Objekte, etwa regulär zusammengesetzter Ausdrücke, welche den (syntaktischen) Eingaben entsprechen, die ein CAS "versteht". Die semantische Gleichwertigkeit solcher Ausdrücke kann dann als eine (nicht notwendig effektiv berechenbare) Äquivalenzrelation $\sim$ auf $\mathcal{E}$ verstanden werden.

Als *Simplifikator* wollen wir eine effektiv berechenbare Funktion $S : \mathcal{E} \longrightarrow \mathcal{E}$ bezeichnen, die die beiden Bedingungen

$$(I) \quad S(S(t)) = S(t) \quad \text{(Idempotenz)} \qquad \text{und} \qquad (E) \quad S(t) \sim t \quad \text{(Äquivalenz)}$$

für alle $t \in \mathcal{E}$ erfüllt.

Wir hatten gesehen, dass man die Simplifikationsvorgänge in einem konkreten CAS als fortgesetzte Funktionstransformationen verstehen kann, wobei ein Arsenal von gewissen Transformationsregeln rekursiv immer wieder auf den entstehenden Zwischenausdruck angewendet wird, bis keine Ersetzungen mehr möglich sind. Diese Ersetzungen erfolgen nicht auf den eingegebenen Ausdrücken, also Elementen aus $\mathcal{E}$, sondern ihren Darstellungen als geschachtelte Listen.

Solche (geschachtelten) Listen kann man als *Binärbäume* auffassen, die rekursiv durch eine Operation `cons` aus atomaren Objekten (Funktionssymbolen, Variablen und Konstanten) und der leeren Liste `()` aufgebaut werden. `M:=cons(A,L)` erzeugt dabei einen Binärbaum M, dessen linker Teilbaum A ein atomares Objekt oder eine nicht leere Liste ist, während der rechte Teilbaum L eine beliebige (evtl. auch leere) Liste $L = (A_2\ A_3\ \dots\ A_n)$ sein muss. Das neue Objekt ist dann die Liste $M = (A\ A_2\ A_3\ \dots\ A_n)$.

Obwohl in praxi in diesen Listen identische Teilausdrücke an verschiedenen Stellen präsent sein können, also die wirkliche Datenstruktur die eines gerichteten kreisfreien Graphen ist, wollen wir uns der Einfachheit halber im Folgenden auf Binärbäume beschränken, indem wir solche identischen Teilausdrücke ggf. durch Duplikate ersetzen. Atomare Objekte und Binärbäume, die sich auf diesem Wege konstruieren lassen, wollen wir im Weiteren unter dem Begriff des *Parse-Baums* zusammenfassen. Atomare Objekte sind genau die Parse-Bäume der Tiefe 0. Ein Binärbaum der Tiefe > 0 ist genau dann ein Parse-Baum, wenn seine linken Blätter atomare Objekte und seine rechten Blätter leere Listen sind.

Die (oBdA umkehrbar eindeutige) Zuordnung zwischen der Menge $\mathcal{E}$ der zulässigen Zeichenketten und der Menge $\mathcal{A}$ der Parse-Bäume erfolgt durch eine (effektiv berechenbare) Funktion $p : \mathcal{E} \longrightarrow \mathcal{A}$, die *Parse-Funktion*.

Aus der Definition lassen sich sofort zwei Eigenschaften herleiten:

(1) Jeder (vollständige) Teilbaum eines Parse-Baums ist selbst wieder ein Parse-Baum.

(2) Ersetzt man in einem Parse-Baum den linken Sohn eines Knotens durch einen anderen Parse-Baum (insbesondere durch eine Variable), so erhält man wieder einen Parse-Baum.

In diesem Kontext lässt sich ein Simplifikator durch ein (endliches) Regelsystem $\mathcal{R}$ beschreiben, dessen Regeln die Gestalt

$$L(u) \longrightarrow R(u)$$

haben mit $L(u), R(u) \in \mathcal{A}$, wobei u eine Liste von formalen Parametern ist, die als linke Blätter in L bzw. R auftreten, und für alle Variablenbelegungen $u = (u_1, \dots, u_m) \vdash U = (U_1, \dots, U_m)$ mit Parse-Bäumen $U_1, \dots, U_m \in \mathcal{A}$

$$p^{-1}\, L(U) \sim p^{-1}\, R(U)$$

in $\mathcal{E}$ gilt. Betrachtet man die Anwendung der einzelnen Regeln als elementaren Simplifikationsschritt, so

ist der zugehörige Simplifikator $S : \mathcal{E} \longrightarrow \mathcal{E}$ der mit p bzw. p^{-1} gekoppelte transitive Abschluss der Ersetzungsrelationen aus $\mathcal{R}$.

S ist somit *effektiv*, wenn die Implementierung von $\mathcal{R}$ die folgenden beiden Bedingungen erfüllt:

1. Es lässt sich effektiv entscheiden, ob es zu gegebenem Parse-Baum B und Regel $(L(u) \longrightarrow R(u))$ eine solche Variablenbelegung $u \vdash U$ gibt, dass $L(U)$ ein Teilbaum von B ist (und die Ersetzung $B^{(1)} := B|_{L(U) \longrightarrow R(U)}$ kann effektiv vorgenommen werden).

2. Nach endlich vielen Schritten $B, B^{(1)}, B^{(2)}, \dots, B^{(N)}$ ist keine Ersetzung mehr möglich.

Während die erste Bedingung, die *Mustererkennung*, sich offensichtlich durch entsprechendes Absuchen des Baumes B unabhängig vom gegebenen Regelsystem erfüllen lässt, hängt die zweite Bedingung, die *Termination*, wesentlich vom Regelsystem $\mathcal{R}$ ab.

Am einfachsten lässt sich die Termination sichern, wenn es eine (teilweise) Ordnungsrelation $\leq$ auf $\mathcal{E}$ gibt, bzgl. derer die rechten Seiten der Regeln "kleiner" als die linken Seiten sind. In diesem Sinne ist dann auch $S(t)$ "einfacher" als t, d.h. es gilt

$$(S) \quad S(t) \leq t \qquad \text{für alle } t \in \mathcal{E}.$$

Die Termination ist gewährleistet, wenn zusätzlich gilt:

(1) $\leq$ ist wohlfundiert.

(2) Für jede Regel $(L(u) \to R(u)) \in \mathcal{R}$ und jede Belegung $u \vdash U$ gilt $p^{-1}\,L(U) > p^{-1}\,R(U)$.

Die zweite Eigenschaft sichert, dass in einem elementaren Simplifikationsschritt die Vereinfachung zu einem "kleineren" Ausdruck führt, die erste, dass eine solche Simplifikationskette nur endlich viele Schritte haben kann. Jede Ordnung, die (2) erfüllt und für die ein Kompliziertheitsmaß $l : \mathcal{E} \longrightarrow \mathbf{N}$ mit

$$e_1 > e_2 \;\Rightarrow\; l(e_1) > l(e_2) \quad \text{für } e_1, e_2 \in \mathcal{E}$$

(wie z.B. Anzahl der verwendeten Zeichen, Zahl der Klammern, der Additionen usw.) existiert, ist wohlfundiert und führt damit zu einem terminierenden Simplifikator. Oftmals ist jedoch die Angabe einer solchen Ordnung auf $\mathcal{E}$ nicht so einfach zu bewerkstelligen und es müssen andere Argumente für die Termination des entsprechenden Simplifikators herangezogen werden.

Ein Simplifikationsprozess kann wesentlich von den gewählten Simplifikationsschritten abhängen, wenn für einen (Zwischen-)Ausdruck mehrere Simplifikationsmöglichkeiten und damit Simplifikationspfade existieren. Dies kann nicht nur Einfluß auf die Rechenzeit, sondern auch auf das Ergebnis selbst haben.

Als *kanonisch* bezeichnet man einen Simplifikator, der zusätzlich der Bedingung

$$(C) \quad s \sim t \Rightarrow S(s) = S(t) \qquad \text{für alle } s, t \in \mathcal{E}$$

genügt. Für einen solchen Simplifikator führen wegen (E) alle möglichen Simplifikationspfade zum selben Ergebnis. $S(t)$ bezeichnet man deshalb auch als **die** *kanonische Form* des Ausdrucks $t \in \mathcal{E}$. Ein solcher Simplifikator hat eine in Bezug auf unsere Ausgangsfrage starke Eigenschaft:

Ein kanonischer Simplifikator erlaubt es, eine Klasse semantisch äquivalenter Ausdrücke an Hand des (syntaktischen) Aussehens der entsprechenden kanonischen Form eindeutig zu identifizieren.

Ein solcher kanonischer Simplifikator kann deshalb insbesondere dazu verwendet werden, die semantische Äquivalenz von zwei gegebenen Ausdrücken zu prüfen, d.h. das *Identifikationsproblem* zu lösen: Zwei Ausdrücke sind genau dann äquivalent, wenn ihre kanonischen Formen (Zeichen für Zeichen) übereinstimmen.

Solche kanonischen Simplifikatoren existieren nur für spezielle Klassen von Ausdrücken $\mathcal{E}$. Deshalb ist auch die folgende Abschwächung dieses Begriffs von Interesse. Nehmen wir an, dass $\mathcal{E}/\sim$, wie in den meisten Anwendungen in der Computeralgebra, die Struktur einer additiven Gruppe trägt, d.h. ein spezielles Symbol $0 \in \mathcal{E}$ für das Nullelement sowie eine Funktion $M : \mathcal{E} \times \mathcal{E} \longrightarrow \mathcal{E}$ existiert, die die Differenz zweier Ausdrücke (effektiv syntaktisch) aufzuschreiben vermag. Letzeres bedeutet insbesondere, dass

$$s \sim t \Leftrightarrow M(s,t) \sim 0$$

gilt. Als *Normalform* bezeichnen wir dann einen Simplifikator S, für den zusätzlich

$$(N) \quad t \sim 0 \Rightarrow S(t) = 0 \qquad \text{für alle } t \in \mathcal{E}$$

gilt, d.h. jeder Nullausdruck $t \in \mathcal{E}$ wird durch S auch als solcher erkannt.

Diese Forderung ist schwächer als die der Existenz einer kanonischen Form: Es kann sein, dass für zwei Ausdrücke $s, t \in \mathcal{E}$, die keine Nullausdrücke sind, zwar $s \sim t$ gilt, diese jedoch zu verschiedenen Normalformen simplifizieren.

Gleichwohl können wir mit einem solchen Normalform-Operator S ebenfalls das Identifikationsproblem lösen, denn es gilt offensichtlich

$$s \sim t \Leftrightarrow S(M(s,t)) = 0.$$

4.4 Simplifikation polynomialer und rationaler Ausdrücke

Wir hatten bei der Betrachtung der Fähigkeiten eines CAS bereits gesehen, dass sie besonders gut in der Lage sind, polynomiale und rationale Ausdrücke zu vereinfachen. Der Grund dafür ist die Tatsache, dass in der Menge dieser Ausdrücke kanonische bzw. normale Formen existieren.

Betrachten wir dazu den Polynomring $S := R[x_1, \ldots, x_n]$ in den Variablen $X = (x_1, \ldots, x_n)$ über dem Grundring R. Wir hatten gesehen, dass solche Polynome $f \in S$ in expandierter Form als Summe von Monomen $f = \sum_a c_a X^a$ dargestellt werden, wobei $a = (a_1, \ldots, a_n) \in \mathbf{N}^n$ ein Multiindex ist und X^a kurz für $x_1^{a_1} \cdots x_n^{a_n}$ steht. In MAPLE, MUPAD und MATHEMATICA kann man eine solche Darstellung durch die Funktionen `expand` oder `normal` erzeugen, in REDUCE und MACSYMA wird sie standardmäßig für die Darstellung von Polynomen verwendet.

Diese Darstellung ist eindeutig, d.h. eine kanonische Form für Polynome $f \in S$, wenn für die Koeffizienten, also die Elemente aus R, eine solche kanonische Form existiert und die Reihenfolge der Summanden festgelegt ist. Zur Festlegung der Reihenfolge definiert man gewöhnlich eine Ordnung auf $T(X) = \{X^a \,:\, a \in \mathbf{N}^n\}$, dem *Monoid der Terme* in $(x_1, \ldots, x_n)$.

Als *distributive Darstellung* eines Polynoms $f \in S$ bzgl. einer solchen Ordnung bezeichnet man eine Darstellung $f = \sum_a c_a X^a$, in der die Summanden paarweise verschiedene Terme enthalten, diese in fallender Reihenfolge angeordnet sind und die einzelnen Koeffizienten in ihre kanonische Form gebracht wurden. In dieser Darstellung ist die Addition von Polynomen besonders effizient ausführbar. Ist die gewählte Ordnung darüberhinaus *monoton*, d.h. gilt

$$s < t \Rightarrow s \cdot u < t \cdot u \quad \text{für alle } s,t,u \in T(X),$$

so kann man auch die Multiplikation recht effektiv ausführen, da dann beim gliedweisen Multiplizieren einer geordneten Summe mit einem Monom die Summanden geordnet bleiben. Wohlfundierte Ordnungen mit dieser Zusatzeigenschaft bezeichnet man als *Termordnungen*.

Zusammenfassend können wir folgenden Satz formulieren:

Satz 3 *Existiert auf R eine kanonische Form, dann ist die distributive Darstellung von Polynomen $f \in S$ mit Koeffizienten in kanonischer Form eine kanonische Form auf S.*

Der Beweis ist offensichtlich. Damit kann in Polynomringen über gängigen Grundbereichen wie den ganzen oder rationalen Zahlen, für die kanonische Formen existieren, effektiv gerechnet werden.

Als *rekursive Darstellung* bezeichnet man die Darstellung von Polynomen aus S als Polynome in x_n mit Koeffizienten aus $R[x_1, \ldots, x_{n-1}]$, wobei die Summanden nach fallenden Potenzen von x_n angeordnet und die Koeffizienten rekursiv nach demselben Prinzip dargestellt sind. Die rekursive Natur des bewiesenen Satzes führt unmittelbar zu folgendem

Korollar 1 *Existiert auf R eine kanonische Form, dann ist auch die rekursive Darstellung von Polynomen $f \in S$ mit Koeffizienten in kanonischer Form eine kanonische Form auf S.*

Die rekursive Darstellung des in distributiver kanonischer Form gegebenen Polynoms

$$f := 2\,x^3\,y^2 + 3\,x^2\,y^2 + 5\,x^2\,y - x\,y^2 - 3\,x\,y + x - y - 2$$

als Element von $R[y][x]$ ist

$$(2y^2)\,x^3 + (3y^2 + 5y)\,x^2 + (-y^2 - 3y + 1)\,x + (-y - 2)$$

Der Leser wird sich fragen, ob Ähnliches auch für die *faktorisierte Darstellung* von Polynomen zutrifft. Schließlich gibt es einzelne polynomiale Ausdrücke, wie etwa

$$u := \left(x - xy + x^2 + (x - y + 1)\left(x^2 + 3\right)\right)\left(-9y - 3y^2 + y^3 + x^4\left(2y + y^2 + 1\right) - 5\right)$$

deren distributive Darstellung

```
> normal(u);
```

$$\begin{aligned}&-20x - 12y - 31xy - 10x^2 - 5x^3 + 18y^2 + 3x^4 + 12y^3 + 4x^5 - 3y^4 + 2x^6 + x^7 - 3xy^2\\ &\quad -13x^2y + 7xy^3 - 9x^3y - xy^4 + 3x^4y + 7x^5y + 3x^6y + 2x^7y + 3x^2y^2 + 5x^2y^3\\ &\quad -3x^3y^2 - x^2y^4 + x^3y^3 - 3x^4y^2 - 3x^4y^3 + 2x^5y^2 - x^5y^3 - x^6y^3 + x^7y^2 - 15\end{aligned}$$

deutlich umfangreicher als eine vollständige Faktorzerlegung ist:

```
> factor(u);
```

$$\left(x + x^2 + 3\right)(x - y + 1)(y + 1)^2\left(y + x^4 - 5\right)$$

Zunächst ist festzuhalten, dass das Rechnen mit Polynomen in faktorisierter Form mit wesentlich mehr Rechenaufwand verbunden ist als wenn diese in distributiver oder rekursiver Form vorliegen, da bei *jeder* Addition zweier Polynome eine (wenigstens teilweise) Faktorisierung notwendig wird. Das kann eine sehr zeitaufwendige Prozedur sein. Da sich außerdem Primfaktoren nur eindeutig bis auf eine Einheit des jeweiligen Integritätsbereichs bestimmen lassen, wird zur Herstellung einer kanonischen Form selbst bei Gültigkeit des Satzes von der eindeutigen Primfaktorzerlegung eine zusätzliche Normierungen notwendig.

Die *faktorisierte Darstellung* selbst von Polynomen mit ganzzahligen Koeffizienten erfüllt deshalb in den meisten CAS nur die (hier trivialen) Anforderungen an eine Normalform.

Rationale Funktionen:

Wir hatten bereits mehrfach gesehen, dass CAS hervorragend in der Lage sind, auch rationale Funktionen zu vereinfachen. Allerdings ist es in einigen Beispielen besser, die spezielle Simplifikationsstrategie `normal` zu verwenden als den allgemeinen Ansatz `simplify`, wie folgendes Beispiel (MuPAD) demonstriert:

```
u:= a^3/((a-b)*(a-c)) + b^3/((b-c)*(b-a)) + c^3/((c-a)*(c-b));
```

$$\frac{a^3}{(a-b)(a-c)} + \frac{b^3}{(b-a)(b-c)} + \frac{c^3}{(c-a)(c-b)}$$

```
simplify(u);
```

$$\frac{a^3}{-ab - ac + bc + a^2} - \frac{b^3}{ab - ac + bc - b^2} + \frac{c^3}{ab - ac - bc + c^2}$$

```
normal(u);
```

$$a + b + c$$

`normal` verfährt nach folgendem einfachen Schema: Es bestimmt einen gemeinsamen Hauptnenner und überführt dann das entstehende Zählerpolynom in seine distributive Form. Auf diese Weise entsteht zwar keine kanonische Form wie im Fall der Polynome, da ja Zähler und Nenner nicht unbedingt teilerfremd sein müssen, allerdings eine Normalform, denn auf diese Weise werden Null-Ausdrücke sicher erkannt. Es gilt damit folgender

Satz 4 *Existiert auf R eine Normalform, dann liefert die beschriebene Normalisierungsstrategie eine Normalform auf dem Ring $R(x)$ der rationalen Funktionen über R.*

Eine solche Darstellung bezeichnet man deshalb auch als *rationale Normalform*. Von ihr gelangt man zu einer kanonischen Form, indem man den gcd von Zähler und Nenner ausdividiert und dann die beiden Teile des Bruches noch geeignet normiert, vgl. etwa [4]. Da die Berechnung des gcd aber im Vergleich zu den anderen arithmetischen Operationen aufwendiger sein kann, verzichtet z.B. Reduce auf diese Berechnung, wenn nicht der Schalter `gcd` gesetzt ist:

```
(x^5-1)/(x^3-1);
```

$$\frac{x^5-1}{x^3-1}$$

```
on gcd; ws;
```

$$\frac{x^4+x^3+x^2+x+1}{x^2+x+1}$$

Ähnlich verfährt die Mathematica-Funktion `Together`, während die Normalformberechnung `normal` in Maple und MuPAD das Austeilen des gcd einschließt. Allerdings wird auch hier auf eine Normierung verzichtet, so dass das Ergebnis eine Normalform, aber keine kanonische Form ist.

Auf Grund der guten Eigenschaften, die die beschriebenen Simplifikationsverfahren polynomialer und rationaler Ausdrücke besitzen, werden sie auch auf Ausdrücke angewendet, die nur teilweise eine solche Struktur besitzen.

Betrachten wir etwa die Vereinfachung, die Reduce automatisch an einem trigonometrischen Ausdruck vornimmt:

```
(sin(x)+cos(x))^5;
```

$$\cos(x)^5 + 5\cos(x)^4\sin(x) + 10\cos(x)^3\sin(x)^2 + 10\cos(x)^2\sin(x)^3 \\ + 5\cos(x)\sin(x)^4 + \sin(x)^5$$

Ein ähnliches Ergebnis liefert die Funktion `expand` bei "konservativeren" Systemen wie z.B. MuPAD:

```
expand((sin(x)+cos(x))^5);
```

$$\cos(x)^5 + \sin(x)^5 + 5\cos(x)\sin(x)^4 + 5\cos(x)^4\sin(x) \\ +10\cos(x)^2\sin(x)^3 + 10\cos(x)^3\sin(x)^2$$

In beiden Fällen entstehen polynomiale Ausdrücke, aber nicht in (freien) Variablen, sondern in allgemeineren Ausdrücken, wie in diesem Fall `sin(x)` und `cos(x)`, die für die vorzunehmende Normalform-Expansion als algebraisch unabhängig angenommen werden. Die zwischen ihnen bestehende algebraische Relation $\sin(x)^2+\cos(x)^2 = 1$ kann durch andere Simplifikationsmechanismen zur Wirkung gebracht werden. Solche allgemeineren Ausdrücke werden als *Kerne* bezeichnet.

Ein Kern kann dabei ein Symbol oder ein Ausdruck mit einem nicht-arithmetischen Funktionssymbol als Kopf sein.

Polynomiale und rationale Ausdrücke in solchen Kernen spielen eine wichtige Rolle im Simplifikationsdesign aller CAS. Sowohl die Herstellung rationaler Normalformen und anschließende Anwendung weitergehender Vereinfachungen wie in obigem Beispiel als auch die gezielte Umformung anderer funktionaler Abhängigkeiten in rationale wie etwa beim Expandieren trigonometrischer Ausdrücke werden angewendet.

MACSYMA und REDUCE unterteilen ihre Simplifikationssysteme sogar in zwei Teile; einen fest eingebrannten Teil, der die als "standard quotients" (REDUCE) bezeichneten rationalen Ausdrücke in Kernen verwaltet und einen regelbasierten, der die Beziehungen zwischen nicht-polynomialen Funktionssymbolen erfasst. In MACSYMA liegt eine spezielle Schicht zwischen dieser als *Canonical Rational Expression* (CRE) bezeichneten Darstellung und der defaultmäßig verwendeten, so dass die interne Repräsentation meist vor dem Nutzer verborgen bleibt. Liegt ein Ausdruck als CRE vor, so erkennt man das an einer speziellen Markierung `/R/` in der Ausgabe. Für unseren Ausdruck erhalten wir dann (MACSYMA)

```
u:(sin(x)+cos(x))^5;
```

$$(\sin x + \cos x)^5$$

```
rat(u);
```

$$\sin^5 x + 5\,\cos x\,\sin^4 x + 10\,\cos^2 x\,\sin^3 x + 10\,\cos^3 x\,\sin^2 x + 5\,\cos^4 x\,\sin x + \cos^5 x$$

Mit dem Befehl `listratvars` können wir uns sogar die "Variablen" der entsprechenden CRE ausgeben lassen.

```
listratvars(u);
                        [cos(x), sin(x)]
```

Wir sehen, dass der betrachtete Ausdruck in der Tat als Polynom in diesen Kernen als verallgemeinerten Unbestimmten dargestellt wird. Noch deutlicher wird dies an folgendem komplexerem Ausdruck

```
u:(exp(x)*cos(x)+cos(x)*sin(x)^4+2*cos(x)^3*sin(x)^2+cos(x)^5)/
(x^2-x^2*exp(-2*x))-(exp(-x)*cos(x)+cos(x)*sin(x)^2+cos(x)^3)/
(x^2*exp(x)-x^2*exp(-x));
```

$$\frac{\cos x\,\sin^4 x + 2\,\cos^3 x\,\sin^2 x + \cos^5 x + e^x\,\cos x}{x^2 - x^2\,e^{-2\,x}} - \frac{\cos x\,\sin^2 x + \cos^3 x + e^{-x}\,\cos x}{x^2\,e^x - x^2\,e^{-x}}$$

```
rat(u);
```

$$\frac{\begin{array}{c}(e^x)^2\,\cos x\,\sin^4 x + \left(2\,(e^x)^2\,\cos^3 x - e^x\,\cos x\right)\,\sin^2 x \\ + (e^x)^2\,\cos^5 x - e^x\,\cos^3 x + \left((e^x)^3 - 1\right)\,\cos x\end{array}}{x^2\,(e^x)^2 - x^2}$$

```
listratvars(u);
```

$$[x, e^x, \cos x, \sin x]$$

Die Bildung einer rationalen Normalform wird von allen CAS der zweiten Generation unterstützt bzw. intensiv verwendet.

System	Rationale Normalform bilden
Macsyma	`rat(u)` wandelt in interne CRE um
Maple	`normal(u)`
Mathematica	`Together[u]`
MuPAD	`normal(u)`
Reduce	standardmäßig verwendet

Tabelle 2: Zur Berechnung von rationalen Normalformen in verschiedenen CAS

4.5 Simplifizieren trigonometrischer Ausdrücke und Regelsysteme

Die verschiedenen Additionstheoreme zeigen, dass zwischen den trigonometrischen Funktionen eine große Zahl von Abhängigkeiten bestehen. Die trigonometrischen Ausdrücke bilden damit eine Klasse, in der besonders viele verschiedene Darstellungen möglich und für die verschiedensten Zwecke auch nützlich sind.

Wir wollen diesen Effekt zunächst an einigen Beispielen studieren, in denen vom entsprechenden CAS selbst solche Umformungen eingesetzt werden. Dazu betrachten wir verschiedene Ausdrücke, die trigonometrische Funktionen enthalten, integrieren diese, bilden die Ableitung der so erhaltenen Stammfunktionen und untersuchen, ob die Systeme in der Lage sind zu erkennen, dass die so produzierten Ausdrücke mit den ursprünglichen Integranden zusammenfallen. Neben unserer eigentlichen Problematik erlaubt die Art der Antwort der einzelnen Systeme zugleich einen gewissen Einblick, wie diese die entsprechenden Integrationsaufgaben lösen.

Als Beipiele betrachten wir die Funktionen

$$\begin{aligned}
f_1 &:= \sin(3x)\,\sin(5x),\\
f_2 &:= \cos\left(\tfrac{x}{2}\right)\,\cos\left(\tfrac{x}{3}\right),\\
f_3 &:= \sin\left(2x-\tfrac{\pi}{6}\right)\,\cos\left(3x+\tfrac{\pi}{4}\right),\\
f_4 &:= \frac{1}{\cos^4(x)},\\
f_5 &:= \frac{1}{\sin^2(x)\,(1-\cos(x))},\\
f_6 &:= \frac{1}{\sin(2x)\,(3\tan(x)+5)}
\end{aligned}$$

Eine typische Antwort von MAPLE lautet etwa

$$g_1 := \int f_1\,dx = \frac{\sin(2x)}{4} - \frac{\sin(8x)}{16}$$

$$g_1' := \frac{\cos(2x)}{2} - \frac{\cos(8x)}{2}\ ,$$

woran wir das Vorgehen gut erkennen: Zur Berechnung des Integrals wurde das Produkt von Winkelfunktionen durch Anwenden der entsprechenden Additionstheoreme in eine Summe einzelner Winkelfunktionen mit Vielfachen von x als Argument umgewandelt. Eine solche Summe kann man leicht integrieren. Wollen wir aber jetzt prüfen, dass $f_1 = g_1'$ gilt, so sind diese Winkelfunktionen mit verschiedenen Argumenten zu vergleichen, wozu es notwendig ist, die Vielfachen von x als Argument wieder aufzulösen, also die entsprechenden Regeln in der anderen Richtung anzuwenden.

Welcher Art von Simplifikationsregeln werden in beiden Fällen verwendet ? Für die erste Aufgabe sind Produkte in Summen zu verwandeln. Das erreicht man durch (mehrfaches) Anwenden der Regeln **Produkt-Summe**

$$\begin{aligned}
\sin(x)\sin(y) &\Rightarrow \tfrac{1}{2}(\cos(x-y)-\cos(x+y))\\
\cos(x)\cos(y) &\Rightarrow \tfrac{1}{2}(\cos(x-y)+\cos(x+y))\\
\sin(x)\cos(y) &\Rightarrow \tfrac{1}{2}(\sin(x-y)+\sin(x+y))
\end{aligned}$$

Diese Regeln sind invers zu den Regeln **Summe-Produkt**, die man beim Expandieren von Winkelfunktionen, deren Argumente Summen oder Differenzen sind, anwendet:

$$\begin{aligned}
\sin(x+y) &\Rightarrow \sin(x)\cos(y)+\cos(x)\sin(y)\\
\cos(x+y) &\Rightarrow \cos(x)\cos(y)-\sin(x)\sin(y)\\
\sin(-x) &\Rightarrow -\sin(x)\\
\cos(-x) &\Rightarrow \cos(x)
\end{aligned}$$

Bei der Formulierung des letzten Regelsatzes haben wir bereits berücksichtigt, dass REDUCE (wie im übrigen auch die anderen CAS) eine Differenz $x - y$ als Summe $x + (-y)$ darstellt, wir also keine Simplifikationsregeln für eine *binäre* Operation MINUS, wohl aber für die *unäre* Operation MINUS angeben müssen.

Zur Anwendung der Produkt-Summe-Regeln ist, wie oben beschrieben, der Parse-Baum des zu vereinfachenden Ausdrucks darauf zu untersuchen, ob in ihm Muster der Form

```
        (* (sin x) (cos y))
oder    (* (cos x) (cos y))
oder    (* (sin x) (sin y))
(oder   (* (cos y) (sin x)))
```

vorkommen, wobei x und y formale Parameter sind. Findet man eine solche Stelle, so sind x und y mit den entsprechenden Teilausdrücke zu unifizieren und der Parse-Baum nach der angegebenen Regel zu transformieren. Diese Transformationen sind so oft auszuführen, so lange sich noch die zu ersetzenden Muster im Parse-Baum finden lassen.

Wir haben damit in einem Regelsystem wie bei Funktionen zwischen formalen Parametern während der Regeldeklaration und Belegung dieser Variablen während der Regelanwendung zu unterscheiden. Im Gegensatz zu Funktionsaufrufen ist bei der Zuordnung von Ausdrücken zu den einzelnen formalen Parametern einer Regel einige Arbeit zu leisten: Es muss das in der Regel angegebene Muster *erkannt* werden. Diesen Prozess nennt man *Unifikation*. Weiterhin müssen wir wie generell im symbolischen Rechnen zwischen Variablen und Symbolen unterscheiden. Betrachten wir noch einmal das Regelsystem für `log`, das von REDUCE verwendet wird, um diesen Unterschied zu verdeutlichen.

```
{log(1) => 0,
log(e) => 1,
log(e^~x) => x,
df(log(~x),~x) => 1/x}
```

Hier steht `e` für das Symbol e, die Basis des natürlichen Logarithmus, `~x` dagegen für einen formalen Parameter, wobei in der letzten Regel sogar an zwei Stellen des Musters derselbe Teilausdruck stehen muss.

Diese Feinheiten von Regelsystemen sind in der Form besonders gut in REDUCE und MATHEMATICA ausgeprägt. Wir wollen deshalb für diesen Abschnitt das System REDUCE zu Grunde legen.

Formulieren wir zunächst die obigen Regeln in REDUCE-Notation. Die Tilde vor einer Variablen bedeutet, dass sie als formaler Parameter verwendet wird, d.h. durch einen beliebigen Ausdruck (= Teilbaum) beim Anwenden der Regel ersetzt werden kann.

```
trigsum0:={ % Produkt-Summen-Regeln
        cos(~x)*cos(~y) => 1/2 * ( cos(x+y) + cos(x-y)),
        sin(~x)*sin(~y) => 1/2 * (-cos(x+y) + cos(x-y)),
        sin(~x)*cos(~y) => 1/2 * ( sin(x+y) + sin(x-y))};
trigexpand0:={ % Summen-Produkt-Regeln
        sin(~x+~y) => sin x * cos y + cos x * sin y,
        cos(~x+~y) => cos x * cos y - sin x * sin y};
```

Die Regeln $\sin(-x) = -\sin(x)$ und $\cos(-x) = \cos(x)$ werden nicht benötigt, da dieser Teil der Simplifikation (gerade/ungerade Funktionen) als spezielle *Eigenschaft* der jeweiligen Funktion vermerkt ist[4] und genau wie die Vereinfachung rationaler Funktionen gesondert behandelt wird.

Wendet man diese Regeln (mehrfach) auf eine Summe von Produkten von Winkelfunktionen `sin` und `cos` an, so sollte man am Ende eine Summe von Winkelfunktionen erhalten, die wegen der Linearität des Integraloperators für das Integrieren besonders geeignet ist. In der Tat wird beim

[4] wie man mit `flagp('sin,'odd)` und `flagp('cos,'even)` erkennt

einmaligen Anwenden der Regel auf ein Produkt dieses in zwei Summanden zerlegt, die selbst wieder Produkte sind, deren Faktorenzahl sich jedoch um 1 verringert hat. Leider klappt das nicht so, wie erwartet, wie etwa das Beispiel

```
sin(x)*sin(2x)*cos(3x) where trigsum0;
```

$$\frac{-\cos(6\,x)+\cos(4\,x)-2\sin(x)^2}{4}$$

zeigt. Hier ist der Ausdruck $\sin(x)^2$ nicht weiter vereinfacht worden, weil er nicht die Gestalt `(* (sin A) (sin B))`, sondern `(expt (sin A) 2)` hat. Wir müssen also noch Regeln hinzufügen, die es erlauben, auch $\sin(x)^n$ und analog $\cos(x)^n$ zu vereinfachen.

Solche Regeln können wir aus der Beziehung $\sin^2(x) = \frac{1-\cos(2x)}{2}$ (analog für $\cos(x)^2$) ableiten, indem wir einen solchen Faktor von $\sin(x)^n$ abspalten und auf die verbleibende Potenz $\sin(x)^{n-2}$ dieselbe Regel rekursiv anwenden. Ein derartiges rekursives Vorgehen ist typisch für Regelsysteme und erlaubt es, Simplifikationen zu definieren, deren Rekursionstiefe von einem der formalen Parameter (wie hier n) abhängig ist. Allerdings müssen wir dazu *konditionierte Regeln* formulieren, denn obige Ersetzung darf nur für ganzzahlige $n > 1$ angewendet werden, um den Abbruch der Rekursion zu sichern. Auch das ist in REDUCE möglich. Eine entsprechende Erweiterung der Regeln `trigsum` sähe dann so aus:

```
trigsum1:={
        sin(~x)^(~n) => (1-cos(2x))/2*sin(x)^(n-2) when fixp n and (n>1),
        cos(~x)^(~n) => (1+cos(2x))/2*cos(x)^(n-2) when fixp n and (n>1)};
```

In Reduce können wir diese Regeln jedoch weiter vereinfachen, wenn wir den

$\boxed{\text{Unterschied zwischen algebraischen und exakten Ersetzungsregeln}}$

beachten. Betrachten wir dazu die distributive Normalform von $(a+1)^5$, also den Ausdruck

$$A = a^5 + 5\,a^4 + 10\,a^3 + 10\,a^2 + 5\,a + 1,$$

und ersetzen in ihn a^2 durch x. Mathematica liefert als Ergebnis

```
A /. (a^2->x)
```

$$1 + 5\,a + 10\,a^3 + 5\,a^4 + a^5 + 10\,x$$

während Reduce

```
(a+1)^5 where (a^2 => x);
```

$$a\,x^2 + 10\,a\,x + 5\,a + 5\,x^2 + 10\,x + 1$$

zurückgibt.

Wir sehen, dass Mathematica nur solche Muster ersetzt hat, die *exakt* auf den Ausdruck a^2 passen, während Reduce erkennt, dass auch in einem Ausdruck $a^k, k > 2$ der Faktor a^2 enthalten ist, also die eine Regel durch eine Serie von Regeln $(a^{k+2} \Rightarrow x \cdot a^k), k \geq 0$ ersetzt hat.

Unser gesamtes Regelsystem `trigsum` für Reduce lautet damit:

```
trigsum:={ % Produkt-Summen-Regeln
        sin(~x)*sin(~y) => 1/2*(cos(x-y)-cos(x+y)),
        sin(~x)*cos(~y) => 1/2*(sin(x-y)-sin(x+y)),
        cos(~x)*cos(~y) => 1/2*(cos(x-y)+cos(x+y)),
        cos(~x)^2 => (1+cos(2x))/2,
        sin(~x)^2 => (1-cos(2x))/2};
```

Die Anwendung dieses Regelsystems **Produkt-Summe** führt auf eine kanonische Darstellung polynomialer trigonometrischer Ausdrücke, ist also für Simplifikationszwecke hervorragend geeignet. Genauer gilt folgender

Satz 5 *Sei R ein Unterkörper von* **C**. *Dann kann jeder polynomiale Ausdruck $P(\sin(x), \cos(x))$ mit Koeffizienten aus R mit obigem Regelsystem* `trigsum` *in einen Ausdruck der Form*

$$\sum_{k>0} (a_k \sin(kx) + b_k \cos(kx)) + c$$

mit $a_k, b_k, c \in R$ verwandelt werden.

Diese Darstellung ist eindeutig. Genauer: Hat R eine kanonische oder Normalform, so kann diese Darstellung zu einer kanonischen bzw. Normalform für die Klasse der betrachteten Ausdrücke verfeinert werden.

Beweis : Da das Regelsystem alle Produkte und Potenzen von Winkelsystemen ersetzt und sich in jedem elementaren Simplifikationsschritt die Anzahl der Multiplikationen verringert, ist nur die Eindeutigkeit der Darstellung zu zeigen. Die Koeffizienten a_k, b_k, c in einer Darstellung

$$f(x) = \sum_{k>0} (a_k \sin(kx) + b_k \cos(kx)) + c$$

kann man aber wie in der Theorie der Fourierreihen zurückgewinnen. Berechnen wir etwa für ein festes ganzzahliges $n > 0$ das Integral

$$\begin{aligned}
&2 \textstyle\int_0^{2\pi} f(x) \sin(nx)\, dx \\
&\quad = \textstyle\sum_{k>0} a_k \int_0^{2\pi} 2\sin(kx)\sin(nx)\, dx \\
&\qquad\qquad + \textstyle\sum_{k>0} b_k \int_0^{2\pi} 2\cos(kx)\sin(nx)\, dx + 2c \int_0^{2\pi} \sin(nx)\, dx \\
&\quad = \textstyle\sum_{k>0} a_k \int_0^{2\pi} (\cos((k-n)x) - \cos((k+n)x))\, dx \\
&\qquad\qquad + \textstyle\sum_{k>0} b_k \int_0^{2\pi} (\sin((n-k)x) + \sin((n+k)x))\, dx \\
&\quad = 2\pi\, a_n,
\end{aligned}$$

so sehen wir, dass $f(x)$ jeden Koeffizienten a_n eindeutig bestimmt. Wir haben dabei von unseren Formeln Produkt-Summe und der Tatsache $\int_0^{2\pi} \sin(mx)\, dx = \int_0^{2\pi} \cos(mx)\, dx = 0$ für $m \neq 0$ Gebrauch gemacht. Analog ergeben sich die Koeffizienten b_n und c eindeutig aus der Funktion $f(x)$. □

Neben der Umwandlung von Potenzen der Winkelfunktionen in Summen mit Mehrfachwinkel-Argumenten ist an anderen Stellen, etwa beim Lösen goniometrischer Gleichungen, auch die umgekehrte Transformation von Interesse, da sie die Zahl der verschiedenen Funktionsausdrücke, die als Kerne in diesen Ausdruck eingehen, verringert. Auf diese Weise liefert die anschließende Berechnung der rationalen Normalform oft ein besseres Ergebnis.

Mathematisch kann man die entsprechenden Formeln aus der *Moivreschen Formel* für komplexe Zahlen und den Potenzgesetzen herleiten: Aus

$$\cos(nx) + i \cdot \sin(nx) = e^{inx} = (e^{ix})^n = (\cos(x) + i \cdot \sin(x))^n$$

und den binomischen Formeln ergibt sich durch Vergleich der Real- und Imaginärteile unmittelbar

$$\cos(nx) = \sum_{2k \le n} (-1)^k \binom{n}{2k} \sin(x)^{2k} \cos(x)^{n-2k}$$

und

$$\sin(nx) = \sum_{2k < n} (-1)^k \binom{n}{2k+1} \sin(x)^{2k+1} \cos(x)^{n-2k-1}$$

Durch eine Mischung von funktionaler und regelbasierter Programmierung lassen sich auch solche komplizierten Regeln implementieren. In REDUCE etwa könnte man für die zweite Formel eine Funktion `Msin` als

```
procedure Msin(n,x);
  begin scalar k,l;
  while (2*k<n) do
  << l:=l+(-1)^k*binomial(n,2k+1)*sin(x)^(2k+1)*cos(x)^(n-2k-1);
     k:=k+1;
  >>;
  return l;
end;
```

zur Berechnung der rechten Seite aus den formalen Parametern n und x vereinbaren, die in der Regel

```
sin(~n*~x) => Msin(n,x) when fixp n and (n>1)
```

zum Einsatz kommt und aus dem unifizierten Muster dessen Ersatz berechnet.

Wir können allerdings auch stärker "informatisch" vorgehen und dem Ansatz

$$\sin(kx) = \sin((k-1)x + x) = \sin((k-1)x)\cos(x) + \cos((k-1)x)\sin(x)$$

folgen, den wir wieder rekursiv zur Anwendung bringen. Unser gesamtes Regelsystem hat dann die Gestalt:

```
trigexpand:={ % Produkt-Summen-Regeln
        sin(~x+~y) => sin x * cos y + cos x * sin y,
        cos(~x+~y) => cos x * cos y - sin x * sin y,
        sin(~k*~x) => sin((k-1)*x)*cos x+cos((k-1)*x)*sin x
                    when fixp k and k>1,
        cos(~k*~x) => cos((k-1)*x)*cos x-sin((k-1)*x)*sin x
                    when fixp k and k>1};
```

Diese Umformungsregeln erlauben es, komplizierte trigonometrische Ausdrücke, in deren Argumenten Summen und Mehrfachwinkel auftreten, durch trigonometrische Ausdrücke mit nur noch wenigen verschiedenen und einfachen Argumenten zu ersetzen. Hängen die Argumente nur ganzzahlig von einer Variablen x ab, so können wir das System wiederum zu einer kanonische Normalform erweitern. Genauer gesagt gilt der folgende Satz:

Satz 6 *Sei R ein Unterring von $\mathbf{C}$. Dann kann man jeden polynomialen Ausdruck in $\sin(k\,x)$ und $\cos(k\,x)$, $k \in \mathbf{Z}$, mit Koeffizienten aus R durch obiges Regelsystem* `trigexpand` *und die zusätzliche Regel $\{\sin(x)^2 \Rightarrow 1-\cos(x)^2\}$ in einen Ausdruck der Form*

$$P(\cos(x)) + \sin(x)\,Q(\cos(x))$$

verwandeln, wobei $P(z)$ und $Q(z)$ Polynome mit Koeffizienten aus R sind.

Diese Darstellung ist eindeutig. Genauer: Hat R eine kanonische oder Normalform, so kann diese Darstellung zu einer kanonischen bzw. Normalform für die Klasse der betrachteten Ausdrücke verfeinert werden.

BEWEIS : Es ist wiederum nur die Eindeutigkeit zu zeigen.

$$P_1(\cos(x)) + \sin(x)\,Q_1(\cos(x)) = P_2(\cos(x)) + \sin(x)\,Q_2(\cos(x))$$

gilt aber genau dann, wenn $(P_1-P_2)(\cos(x)) + \sin(x)\,(Q_1-Q_2)(\cos(x))$ identisch verschwindet. Wir haben also nur zu zeigen, dass aus der Tatsache, dass $P(\cos(x)) + \sin(x)\,Q(\cos(x))$ identisch verschwindet, bereits folgt, dass $P(z)$ und $Q(z)$ beides Nullpolynome sind. Aus $P(\cos(x)) +$

$\sin(x)\,Q(\cos(x)) = 0$ folgt aber nach der Substitution $x = -x$ auch $P(\cos(x)) - \sin(x)\,Q(\cos(x)) = 0$ und schließlich $P(\cos(x)) = Q(\cos(x)) = 0$. Da ein nichttriviales Polynom aber nur endlich viele Nullstellen besitzt, folgt die Behauptung. □

Mit diesen beiden Strategien lassen sich die ersten drei Integrationsaufgaben, mit denen wir diesen Abschnitt begonnen hatten, zufriedenstellend lösen. Einzige Schwierigkeit ist die Vereinfachung von $f_2 - g_2'$, da

$$g_2 := \int f_2\,dx = \frac{3}{5}\sin\left(\frac{5x}{6}\right) + 3\sin\left(\frac{x}{6}\right)$$

Winkel enthält, von denen nicht auf den ersten Blick sichtbar ist, dass es sich um ganzzahlige Vielfache eines gemeinsamen Winkels handelt. Die explizite Substitution $x = 6y$, die das System darauf hinweist, hilft hier aber weiter.

Die klare Struktur der beiden Simplifikationssysteme verwendet implizit die Tatsache, dass Reduce standardmäßig polynomiale Ausdrücke in ihre distributive Normalform transformiert. Dies muss dem Regelsystem hinzugefügt werden, wenn eine solche Simplifikation nicht automatisch erfolgt. Wir wollen diesen Aspekt an Mathematica demonstrieren. Vorab sei auf einige Besonderheiten der Syntax dieses Systems hingewiesen, die zum Verständnis der Formeln unerlässlich sind. Für eine detailliertere Einführung verweisen wir auf einschlägige Systemliteratur.

Mathematica unterscheidet im Gegensatz etwa zu Reduce streng zwischen Groß- und Kleinschreibung. Die meisten Funktionsnamen beginnen jede “Silbe” mit einem Großbuchstaben. Ihre Argumentliste wird im Gegensatz zu anderen Systemen in eckige Klammern eingeschlossen. Runde Klammern dienen ausschließlich der Gruppierung von Teilausdrücken. Einstellige Funktionen wie etwa `Simplify, Together` oder `Expand` können auch in der Postfix-Notation `<Expr>//Expand` statt `Expand[<Expr>]` geschrieben werden, was der Auswertungsreihenfolge (erst die Argumente, dann der Funktionsaufruf) stärker entspricht als die übliche Präfixnotation. Für viele zweistellige Funktionen existieren Infix-Notationen in Operatorform, wie etwa für die Substitutionsoperatoren `/.` (einmalige Substitution) und `//.` (rekursive Substitution; beide unterscheiden sich genau wie Evaluationstiefe 1 und maximale Evaluationstiefe). Formale Parameter werden durch einen Unterstrich, etwa als `x_`, gekennzeichnet, dem im Gegensatz zu Reduce weitere Information über den Typ oder eine Default-Initialisierung folgen können. Deshalb dürfen, ebenfalls im Gegensatz zu Reduce, Unterstriche nicht in Variablennamen auftreten.

Obige Regelsysteme hätten damit in Mathematica folgende Gestalt:

```
trigexpand={ (* Expandiert Winkelfunktionen *)
        Cos[x_+y_] -> Cos[x]*Cos[y] - Sin[x]*Sin[y],
        Sin[x_+y_] -> Sin[x]*Cos[y] + Cos[x]*Sin[y],
        Cos[n_Integer*x_]/; (n>1)  ->
                Cos[(n-1)*x]*Cos[x]-Sin[(n-1)*x]*Sin[x],
        Sin[n_Integer*x_]/; (n>1)  ->
                Sin[(n-1)*x]*Cos[x]+Cos[(n-1)*x]*Sin[x]};

trigsum={ (* Verwandelt Produkte in Summen von Mehrfachwinkeln *)
        Cos[x_]*Cos[y_] -> 1/2*(Cos[x+y]+Cos[x-y]),
        Sin[x_]*Sin[y_] -> 1/2*(-Cos[x+y]+Cos[x-y]),
        Sin[x_]*Cos[y_] -> 1/2*(Sin[x+y]+Sin[x-y]),
        Sin[x_]^(n_Integer)/; (n>1) -> (1-Cos[2*x])/2*Sin[x]^(n-2) ,
        Cos[x_]^(n_Integer)/; (n>1) -> (1+Cos[2*x])/2*Cos[x]^(n-2) };
```

Wenden wir das zweite System auf $\sin(x)^7$ an, so erhalten wir nicht die aus unseren Rechnungen mit Reduce erwartete Normalform, sondern einen teilfaktorisierten Ausdruck:

```
Sin[x]^7//.trigsum
```

$$\frac{(1-\cos(2\,x))^3\,\sin(x)}{8}$$

Dieser Ausdruck muss erst expandiert werden, damit die Regeln erneut angewendet werden können.

```
%//Expand;
%//.trigsum
```

$$\frac{\sin(x)}{8}+\frac{3\ (1+\cos(4\,x))\ \sin(x)}{16}-\frac{3\ (-\sin(x)+\sin(3\,x))}{16}-\frac{(1+\cos(4\,x))\ (-\sin(x)+\sin(3\,x))}{32}$$

usw., ehe nach vier solchen Schritten schlielich das Endergebnis

$$\frac{35\ \sin(x)}{64}-\frac{21\ \sin(3\,x)}{64}+\frac{7\ \sin(5\,x)}{64}-\frac{\sin(7\,x)}{64}$$

feststeht.

Wir benötigen also nach jedem Simplifikationsschritt noch die Überführung in die rationale Normalform, also die Funktionalität von `Together`. Allerdings kann man nicht einfach

```
x_ -> Together[x]
```

schreiben, denn diese Regel würde ja immer passen und damit das Regelsystem nicht terminieren. Eine genauere Analyse zeigt, dass wir zusätzlich nur das Distributivitätsgesetz für Produkte von Summen benötigen. Das kann man durch eine einzige weitere Regel erreichen:

```
trigsum={ (* Verwandelt Produkte in Summen von Mehrfachwinkeln *)
        Cos[x_]*Cos[y_] -> 1/2*(Cos[x+y]+Cos[x-y]),
        Sin[x_]*Sin[y_] -> 1/2*(-Cos[x+y]+Cos[x-y]),
        Sin[x_]*Cos[y_] -> 1/2*(Sin[x+y]+Sin[x-y]),
        Sin[x_]^(n_Integer)/; (n>1) -> (1-Cos[2*x])/2*Sin[x]^(n-2) ,
        Cos[x_]^(n_Integer)/; (n>1) -> (1+Cos[2*x])/2*Cos[x]^(n-2) ,
        (a_+b_)*c_ -> a*c+b*c};
```

Nun zeigt die Simplifikation das erwünschte Verhalten:

```
Sin[x]^7//.trigsum
```

$$\frac{35\ \sin(x)}{64}-\frac{21\ \sin(3\,x)}{64}+\frac{7\ \sin(5\,x)}{64}-\frac{\sin(7\,x)}{64}$$

Da die beiden Regelsysteme `trigsum` und `trigexpand` zueinander entgegengesetzt sind, kann man nicht beide gleichzeitig in einem Simplifikationssystem zulassen, ohne Zirkelschlüsse zu verursachen. Dem Nutzer stehen in den verschiedenen Systemen dafür verschiedene fest eingebaute Simplifikationsstrategien zur Verfügung. Welche Freiheiten erlauben in dieser Richtung die einzelnen Systeme?

Derive: Über Knöpfe kann zwischen verschiedenen eingebauten Strategien gewechselt werden. Neben Expandieren und Faktorisieren polynomialer Ausdrücke (= verschiedene polynomiale Normalformen) kann zwischen Expandieren und Sammeln von Exponenten, von logarithmischen Ausdrücken und Winkelfunktionen gewählt werden. Letzteres entspricht den beiden hier vorgestellten Strategien, wobei noch gewählt werden kann, ob in Richtung $\sin(x)$ oder $\cos(x)$ simplifiziert werden soll.

Die vier "M-Systeme" Macsyma, Maple, Mathematica und MuPAD stellen eine ganze Reihe von Simplifikationsprozeduren mit verschiedenen Parametern zur Verfügung, die unterschiedliche Ziele ansteuern. In Maple sind dies insbesondere die Befehle `expand, combine, simplify` und `convert`.

Auf trigonometrische Ausdrücke angewendet bewirken `combine` (Produkte in Winkelsummen) und `expand` (Winkelsummen in Produkte) die obigen Umformungen, während man mit `convert` bzw. `rewrite` (u.a.) trigonometrische Funktionen und deren Umkehrfunktionen in Ausdrücke mit *exp* und *log* von komplexen Argumenten umformen kann. Dies entspricht dem REDUCE-Regelsatz

```
trigexp:={
        tan(~x) => sin(x)/cos(x),
        sin(~x) => (exp(i*x)-exp(-i*x))/(2*i),
        cos(~x) => (exp(i*x)+exp(-i*x))/2};
```

In der folgenden Tabelle sind die Namen der entsprechenden Funktionen in den einzelnen Systemen einander gegenübergestellt, die dieselben Transformationen wie oben beschrieben bewirken.

Wirkung	Macsyma	Maple	Mathematica	MuPAD
Argumentsummen auflösen	trigexpand	expand	TrigExpand	expand
Produkte zu Mehrfachwinkeln	trigreduce	combine	TrigReduce	combine
Trig $\mapsto$ Exp	exponentialize	convert(u,exp)	TrigToExp	rewrite(u,exp)
Exp $\mapsto$ Trig	demoivre	convert(u,trig)	ExpToTrig	?

Tabelle 3: Ausgewählte Transformationsfunktionen für Ausdrücke, die trigonometrische Funktionen enthalten.

Maple erlaubt es auch, innerhalb des Simplify-Befehls eigene Regelsysteme anzugeben, kennt dabei aber keine formalen Parameter. Statt dessen kann man für einzelne neue Funktionen den simplify-Befehl erweitern, was aber ohne interne Systemkenntnisse recht schwierig ist. Ähnlich kann man in MuPAD die Funktionsaufrufe `combine` und `expand` durch die Definition entsprechender Attribute auf andere Funktionssymbole ausdehnen.

Macsyma und Mathematica erlauben die Definition eigener Regelsysteme, mit denen man die intern vorhandenen Transformationsmöglichkeiten erweitern kann.

Dies gilt auch für Axiom, welches im Paket `TranscendentalManipulations` neben Umformungen von einer Winkelfunktion zu einer anderen die beiden Funktionen `expand` (Winkelsummen in Produkte) und `simplify` (alles in `sin` und `cos`) zur Verfügung stellt. Ein ähnlich umfangreicher Satz von Transformationsfunktionen wie in den anderen CAS fehlt allerdings.

Auch Reduce kennt nur wenige eingebaute Regeln, was sich insbesondere für die Arbeit mit trigonometrischen Funktionen als nachteilig erweist[5]. Wir haben aber in diesem Abschnitt gesehen, dass es nicht schwer ist, solche Regelsysteme selbst zu entwerfen. Jedoch muss der Nutzer dazu wenigstens grobe Vorstellungen über die interne Datenrepräsentation besitzen. Besonders günstig ist, dass man eigene Simplifikationsregeln nicht nur als global gültige, sondern seit der Version 3.5 auch als lokal gültige Regeln anwenden kann. Insbesondere letzteres erlaubt es, gezielt Umformungen mit überschaubaren Seiteneffekten auf Ausdrücken vorzunehmen.

Doch kehren wir zur Berechnung der 6 Beispielintegrale zurück. Hier sind die von Derive berechneten Ergebnisse im einzelnen aufgelistet:

[5] In neueren Versionen gibt es allerdings das Paket `trigsimp`, das Simplifikationsroutinen für trigonometrische Funktionen zur Verfügung stellt.

$$g_1 := \frac{\sin(2x)}{4} - \frac{\sin(8x)}{16}$$

$$g_2 := \frac{3}{5}\sin\left(\frac{5x}{6}\right) + 3\sin\left(\frac{x}{6}\right)$$

$$g_3 := -\frac{\sqrt{6}+\sqrt{2}}{40}\cos(5x) + \frac{\sqrt{6}-\sqrt{2}}{40}\sin(5x) + \frac{\sqrt{6}-\sqrt{2}}{8}\cos(x) - \frac{\sqrt{6}+\sqrt{2}}{8}\sin(x)$$

$$g_4 := \frac{2}{3}\tan(x) + \frac{\sin(x)}{3\cos^3(x)}$$

$$g_5 := -\frac{2\cos^2(x) - 2\cos(x) - 1}{3\sin(x)\,(\cos(x) - 1)}$$

$$g_6 := \frac{1}{10}\ln\left(\frac{\sin(x)}{5\cos(x) + 3\sin(x)}\right)$$

REDUCE geht bei der Berechnung der ersten drei Integrale anders vor: Aus der Kenntnis der Form der Antwort wird diese mit unbestimmten Koeffizienten angesetzt und die konkreten Koeffizientenwerte werden ausgerechnet. Dies erkennen wir deutlich, wenn wir Integrale mit allgemeinen Koeffizienten eingeben

```
int(sin(a*x)*sin(b*x),x);
```

$$\frac{\mathit{sin}(a\,x)\,\mathit{cos}(b\,x)\,b\;-\;\mathit{sin}(b\,x)\,\mathit{cos}(a\,x)\,a}{a^2\;-\;b^2}$$

oder

```
int(sin(a*x)*sin(b*x)*sin(c*x),x);
```

$$\begin{aligned}&\big(\sin(ax)\sin(bx)\cos(cx)a^2c + \sin(ax)\sin(bx)\cos(cx)b^2c - \sin(ax)\sin(bx)\cos(cx)c^3\\ &+\sin(ax)\sin(cx)\cos(bx)a^2b - \sin(ax)\sin(cx)\cos(bx)b^3 + \sin(ax)\sin(cx)\cos(bx)bc^2\\ &-\sin(bx)\sin(cx)\cos(ax)a^3 + \sin(bx)\sin(cx)\cos(ax)ab^2 + \sin(bx)\sin(cx)\cos(ax)ac^2\\ &+2\cos(ax)\cos(bx)\cos(cx)abc\big)/(a^4 - 2a^2b^2 - 2a^2c^2 + b^4 - 2b^2c^2 + c^4)\end{aligned}$$

Damit vermeidet das System in den ersten drei Fällen, Winkelfunktionen mit neuen Argumenten einzuführen, was die nachfolgende Simplifikation natürlich wesentlich vereinfacht. Die Berechtigung zu solchem Vorgehen folgt aus

Satz 7 *Sei $\{m_1, m_2, \ldots, m_k\}$ eine endliche (Multi-)Menge reeller Zahlen. Wir betrachten die Menge der Ausdrücke*

$$T := \{t_1(m_1x) * \cdots * t_k(m_kx)\ :\ t_i \in \{\sin, \cos\}\}$$

und $L = L(T)$ die Menge der reellen Linearkombinationen von Produkten aus T.

Dann gibt es zu jedem $f(x) \in L$ einen Ausdruck $g(x) \in L$ und ein $c \in \mathbf{R}$ mit

$$\int f(x)\,dx = g(x) + c\,x,$$

d.h. das Integral von $f(x)$ lässt sich bis auf einen linearen Korrekturterm als polynomialer Ausdruck in im wesentlichen denselben Kernen darstellen wie $f(x)$.

BEWEIS : Wendet man auf $f(x) = t_1(m_1x)\cdots t_k(m_kx)$ unsere Regeln `trigsum` an, so erhält man eine Linearkombination von Ausdrücken $t(\sum_{i=1}^k \varepsilon_i m_i x)$ mit $t \in \{\sin, \cos\}$ und $\varepsilon_i = \pm 1$. Integration solcher Ausdrücke bleibt in der Klasse außer für $\sum \varepsilon_i m_i = 0$ und $t = \cos$. Expansion der

Ausdrücke $t(\sum \varepsilon_i m_i x)$, aus denen das Integral zusammengesetzt ist, längs der "Sollbruchstellen" mit `trigexpand` führt wieder in die Klasse L zurück. $\square$

REDUCE liefert damit die folgenden Antworten:

$$g_1 := \frac{3\sin(5x)\cos(3x) - 5\sin(3x)\cos(5x)}{16}$$

$$g_2 := \frac{6\left(-2\sin(\frac{x}{3})\cos(\frac{x}{2}) + 3\sin(\frac{x}{2})\cos(\frac{x}{3})\right)}{5}$$

$$g_3 := \frac{3\sin(\frac{12x-\pi}{6})\sin(\frac{12x+\pi}{4}) + 2\cos(\frac{12x-\pi}{6})\cos(\frac{12x+\pi}{4})}{5}$$

$$g_4 := \frac{\sin(x)\left(2\sin(x)^2 - 3\right)}{3\cos(x)\left(\sin(x)^2 - 1\right)}$$

$$g_5 := \frac{2\sin(x)^2 + 2\cos(x) - 1}{3\sin(x)\left(\cos(x) - 1\right)}$$

$$g_6 := \frac{-\log\left(3\tan(x) + 5\right) + \log\left(\tan(x)\right)}{10}$$

Die Berechnung der letzten drei Integrale erfolgte mit Hilfe eines allgemeinen Verfahrens, das darauf beruht, die Kerne $\sin(x)$, $\cos(x)$ durch $\tan(\frac{x}{2})$ auszudrücken. Dies ist mit folgendem Regelsatz möglich, nachdem alle anderen Winkelfunktionen allein durch `sin` und `cos` ausgedrückt sind:

```
trigtan:={
        sin(~x) => (2*tan(x/2))/(1+tan(x/2)^2),
        cos(~x) => (1-tan(x/2)^2)/(1+tan(x/2)^2)};
```

Die dazu inverse Regel

```
invtrigtan:={ tan(~x) => sin(2x)/(1+cos(2x) };
```

erlaubt es, Halbwinkel im Ergebnis wieder so weit als möglich zu eliminieren. Grundlage dieses Vorgehens ist die Fähigkeit der Systeme, rationale Funktionen in der Integrationsvariablen zu integrieren sowie der

Satz 8 *Sei* $R(z) = \frac{P(z)}{Q(z)}$ *eine rationale Funktion. Dann kann*

$$\int R(\tan(\frac{x}{2}))\,dx$$

durch die Substitution $y = \tan(\frac{x}{2})$ *auf die Berechnung des Integrals einer rationalen Funktion zurückgeführt werden.*

Beweis: Es gilt $\tan(x)' = 1 + tan(x)^2$ und folglich $dx = \frac{2\,dy}{1+y^2}$, womit wir

$$\int R(\tan(\frac{x}{2}))\,dx = \int \frac{R(y)}{1+y^2}\,dy$$

erhalten.

4.6 Das allgemeine Simplifikationsproblem

Betrachten wir zum Abschluss dieses Kapitels, wie sich die verschiedenen CAS bei der Simplifikation komplexerer Ausdrücke verhalten.

1. Beispiel:

```
u1:=(exp(x)*cos(x)+cos(x)*sin(x)^4+2*cos(x)^3*sin(x)^2+cos(x)^5)/
(x^2-x^2*exp(-2*x))-(exp(-x)*cos(x)+cos(x)*sin(x)^2+cos(x)^3)/
(x^2*exp(x)-x^2*exp(-x));
```

Dieser nicht zu komplizierte Ausdruck enthält neben trigonometrischen auch Exponentialfunktionen. Hier sind offensichtlich die Regeln anzuwenden, die weitestgehend eine der Winkelfunktionen durch die andere ausdrücken. Als Ergebnis erhält man den Ausdruck

$$\frac{\cos(x)\,(e^x+1)}{x^2}$$

Eine genauere Analyse zeigt, dass hierfür einzig die Regel $\cos(x)^2 \Rightarrow 1-\sin(x)^2$ anzuwenden ist.

MAPLE hatte noch in der Version V.3 bei diesem Ausdruck Schwierigkeiten, den kleinsten gemeinsamen Nenner zu erkennen:

```
simplify((exp(x)-exp(-x))/(1-exp(-2*x)));
```

lieferte

$$-\frac{e^x-e^{-x}}{-1+e^{-2\,x}} \qquad \text{statt} \qquad e^x$$

Das ist in neueren Versionen behoben.

REDUCE benötigt, nach unseren bisherigen Ausführungen nicht unerwartet, als Hilfestellung die oben angegebene Regel[6], während DERIVE, MATHEMATICA und MUPAD ebenfalls bereits mit `Simplify` zu dem erwarteten Ergebnis kommen. MACSYMA kann den Ausdruck mit einer speziellen Simplifikationsroutine für trigonometrische Funktionen ebenfalls zufriedenstellend vereinfachen.

2. Beispiel:

```
u2:=16*cos(x)^3*cosh(x/2)*sinh(x)-6*cos(x)*sinh(x/2)-
6*cos(x)*sinh(3/2*x)-cos(3*x)*(exp(3/2*x)+exp(x/2))*(1-exp(-2*x));
```

Hier sind die Simplifikationsregeln für trigonometrische und hyperbolische Funktionen anzuwenden. Das kann man etwa durch Umformung in Exponentialausdrücke erreichen. Dieses Zusatzwissen erlaubt es REDUCE, mit den Regelsystemen

```
hyp2exp:={
tanh(~x) => sinh(x)/cosh(x),
coth(~x) => cosh(x)/sinh(x),
sinh(~x) => (e^x-e^(-x))/2,
cosh(~x) => (e^x+e^(-x))/2};

trig2exp:={
tan(~x) => sin(x)/cos(x),
cot(~x) => cos(x)/sin(x),
sin(~x) => (e^(I*x)-e^(-I*x))/2/I,
cos(~x) => (e^(I*x)+e^(-I*x))/2};
```

die Identität $u=0$ zu erkennen.

Dasselbe Ergebnis erhielt man in MAPLE V.3 über

```
u2a:=convert(u2,exp);
u2b:=expand(u2a);
```

Um den Ausdruck auch in neueren Versionen zu Null zu vereinfachen, ist noch eine zusätzliche Anwendung der Potenzgesetze notwendig:

```
combine(u2b,power);
```

[6]Auch die Funktion `trigsimp(u);` aus dem Paket `trigsimp` liefert das obige Resultat

Ähnlich kann man in MACSYMA und MUPAD vorgehen, während in MATHEMATICA wiederum ein Aufruf der Funktion `Simplify` genügt. DERIVE genügt ein Aufruf von Simplify, wenn man vorher `Trigonometry:=Expand` gesetzt hat.

System	Beispiel `u1`	Beispiel `u2`
Derive	(S)implify	(S)implify mit `Trigonometry:=Expand`
Macsyma	`rat(trigreduce(u1))`	`expand(exponentialize(u2))`
Maple	`simplify(u1)`	`combine(expand(convert(u2,exp)),power)`
Mathematica	`u1//Simplify`	`u2//Simplify`
MuPAD	`simplify(u1)`	`combine(expand(rewrite(u2,exp)),exp)`
Reduce	`u1 where trigsin`	`u2 where hyp2exp,trig2exp`

Tabelle 4: Simplifikation von u_1 und u_2 auf einen Blick

3. Beispiel (aus [6, S. 81])

```
u3:=log(tan(x/2)+sec(x/2))-arcsinh(sin(x)/(1+cos(x)));
```

Diesen Ausdruck vermag keines der Systeme zu Null zu vereinfachen, obwohl auch hier das Vorgehen für einen Mathematiker mit geübtem Blick sehr transparent ist: Wegen $arcsinh(a) = \log(a+\sqrt{a^2+1})$ ist *arcsinh* durch einen logarithmischen Ausdruck zu ersetzen und dann die beiden Logarithmen zusammenzufassen. Dies wird in DERIVE beim Vereinfachen automatisch vorgenommen, ist in MACSYMA, MAPLE und MUPAD durch eingebaute Funktionen (`convert(u3,ln)`) und in den anderen Systemen durch Angabe einer einfachen Regel realisierbar, etwa in MATHEMATICA als

```
Arch2Log = { ArcSinh[x_] -> Log[x+Sqrt[1+x^2]] }
```

Vereinfacht man die Differenz A dieser beiden Logarithmen mit `simplify`, so wartet nur MATHEMATICA mit einem einigermaßen passablen Ergebnis auf:

$$\log\left(\sec(\frac{x}{2})+\tan(\frac{x}{2})\right)-\log\left(\sqrt{\sec(\frac{x}{2})^2}+\tan(\frac{x}{2})\right)$$

Die Simplifikation endet an der Stelle, wo $\sqrt{x^2}$ nicht zu x vereinfacht wird, obwohl dies hier aus dem Kontext heraus erlaubt wäre (wenn man voraussetzt, dass es sich um reelle Funktionen handelt): $\log(\sec(\frac{x}{2})+\tan(\frac{x}{2}))$ hat als Definitionsbereich $D=\{x\,:\,\cos(\frac{x}{2})>0\}$. Wir können MATHEMATICA mit der Regel

```
SimpSqrt = { Sqrt[x_^2] -> x }
```

veranlassen, diese Vereinfachung trotzdem vorzunehmen und erhalten danach schließlich 0.

Um in den anderen Systemen ebenfalls die Nulläquivalenz des betrachteten Ausdrucks A zu erkennen, ist es sinnvoll, die beiden Logarithmen zusammenzufassen. In DERIVE erfolgt dies automatisch und ist mit `logcontract` (MACSYMA), `combine(A,ln)` (MAPLE und MUPAD) oder dem Schalter `combinelogs` (REDUCE) auch in den anderen Systemen auf einfache Weise möglich. MAPLE V.4 weigert sich allerdings, die Zusammenfassung $\ln(u)-\ln(v)\mapsto\ln(u/v)$ vorzunehmen, wenn nicht $u,v>0$ bekannt ist, da die Werte des Logarithmus sonst ja komplex werden und das Ergebnis verfälscht sein könnte. Da regelbasiertes Programmieren von MAPLE nicht unterstützt wird, müssen wir uns an dieser Stelle damit behelfen, die geforderte Funktionstransformation "per Hand" vorzunehmen, indem wir A auseiandernehmen und anders wieder zusammenbauen[7]:

```
A1 := combine(A,ln);
A2 := ln(op([1,1],A1)/op([2,2,1],A1));
```

[7] Ab Version V.5 kann diese Funktionstransformation über den Befehl `combine(A,ln,anything,symbolic)` erreicht werden.

Nachdem diese Hürde genommen ist entsteht als Argument der Logarithmusfunktion ein komplizierter trigonometrischer Ausdruck

$$B := \frac{\tan(\frac{x}{2}) + \sec(\frac{x}{2})}{\frac{\sin(x)}{1+\cos(x)} + \sqrt{\frac{(\sin(x))^2}{(1+\cos(x))^2} + 1}},$$

der mit den bisher betrachteten Regeln weiter vereinfacht werden kann. MATHEMATICA liefert bereits mit `Simplify[B]` ein verwertbares Ergebnis

$$\frac{1 + \sin(\frac{x}{2})}{\cos(\frac{x}{2})\sqrt{\sec(\frac{x}{2})^2} + \sin(\frac{x}{2})},$$

das mit der Regel `simpsqrt` zu Eins vereinfacht.

Die anderen Systeme haben Schwierigkeiten, die Winkelargumente x und $x/2$ zueinander in Beziehung zu setzen. Ersetzen wir in B deshalb noch $x \mapsto 2y$:

```
C:=subs(x=2*y,B);
```

MAPLE liefert nun mit

```
C1:=simplify(expand(C));
```

$$\frac{\sin(y) + 1}{\sin(y) + \mathrm{csgn}(\cos(\overline{y}))}$$

ein erstes Ergebnis, dem man nach Einschränkung des Definitionsbereichs von y

```
assume(y,real);
C1;
```

$$\frac{\sin(y) + 1}{\sin(y) + \mathrm{sgn}(\cos(y))}$$

die Nulläquivalenz wiederum "fast" ansieht (siehe die obige Bemerkung zum Definitionsbereich D des betrachteten Ausdrucks). Dasselbe Ergebnis erhält man mit DERIVE, wenn `Trigonometry := Expand` einstellt wird.

MACSYMA liefert mit

```
trigsimp(trigexpand(C));
```

$$\frac{\cos y\ \sin y + \cos y}{|\cos y| + \cos y\ \sin y};$$

ein Ergebnis ähnlicher Qualität. Man beachte, dass dieser einfache Ausdruck nur entsteht, wenn zuerst die Mehrfachwinkel aufgelöst werden, ehe die allgemeine Simplifikationsroutine gestartet wird.

Mit REDUCE kann man dieselben Umformungen durch geeignete Regelsysteme erreichen, die in der Datei `trig` enthalten sind. Zunächst erzeugen wir den Ausdruck B. In Version 3.6 funktioniert die Befehlsfolge

```
on combinelogs$
A:=(u3 where archtolog)$
B:=part(A,1)$
```

In Version 3.7 dagegen werden die Logarithmen selbst für den Fall, dass `on combinelogs` gesetzt ist, nicht zusammengefasst, so dass wir wie oben die entsprechende Funktionstransformation "per Hand" vornehmen müssen:

```
B:=part(A,1,1)/part(A,2,1,1)$
C:=sub(x=2*y,B)$
C where trig;
```

In Version 3.6 erhalten wir als (vorsichtigeres) Ergebnis korrekterweise noch

$$\frac{\cos(y)\sin(y)+\cos(y)}{\mathrm{abs}(\cos(y))+\cos(y)\sin(y)},$$

während Version 3.7. (in der Version vom 15 April 1999) wieder die Vereinfachung $\sqrt{x^2}=x$ vornimmt und so den Wert 1 zurückliefert.

Bei MuPAD führt dieselbe Strategie zu keinem verwertbaren Ergebnis.

Wir sehen hieran bereits, dass man sich offensichtlich stets genügend komplizierte Simplifikationsprobleme ausdenken kann, die mit dem vorhandenen Instrumentarium nicht aufgelöst werden können. Es gilt jedoch noch mehr:

Satz 9 *Aus den Funktionssymbolen* sin, abs *und* $*, +$ *sowie der Konstanten* π *und ganzen Zahlen kann man Ausdrücke zusammenstellen, für die das Simplifikationsproblem algorithmisch unlösbar ist.*

Das allgemeine Simplifikationsproblem gehört damit zur Klasse der algorithmisch unlösbaren Probleme.

Der Beweis dieses Satzes (der hier nicht geführt wird) wird zurückgeführt auf die negative Antwort zum 10. Hilbertschen Problem, die Matiyasewitsch und Roberts Ende der 60er Jahre gefunden haben.

Kapitel 5

Algebraische Zahlen

Wir hatten im Kapitel 4 gesehen, dass sich Polynome mit Koeffizienten aus einem Grundbereich R, auf dem eine kanonische Form existiert, selbst stets in eine kanonische Form bringen lassen und so das Rechnen in diesen Strukturen besonders einfach ist. Dies gilt insbesondere für Polynome über den ganzen oder rationalen Zahlen, da für beide Bereiche offensichtliche kanonische Formen (etwa die Darstellung ganzer Zahlen mit Vorzeichen im Dezimalsystem oder die rationaler Zahlen als unkürzbare Brüche mit positivem Nenner) existieren.

Treten als Koeffizienten Wurzelausdrücke wie $\sqrt{2}$ oder $\sqrt{2+\sqrt{3}}$ auf, so kann man die Frage nach einer kanonischen oder wenigstens normalen Form schon nicht mehr so einfach beantworten. Eine solche Darstellung müsste ja wenigstens die bereits früher betrachteten Identitäten

$$\sqrt{2\sqrt{3}+4} = 1+\sqrt{3}$$

oder

$$\sqrt{11+6\sqrt{2}}+\sqrt{11-6\sqrt{2}} = 6$$

erkennen, wenn die entsprechenden Wurzelausdrücke als Koeffizienten auftreten. Während dies in Maple und Derive automatisch erfolgt, sind die anderen Systeme nur mit viel gutem Zureden bereit, diese Simplifikation vorzunehmen. Selbst das Normalformproblem, also jeweils die Vereinfachung der Differenz beider Seiten der Identitäten zu 0, wird weder in MuPAD noch in Mathematica oder Macsyma allein durch den Aufruf von `simplify` gelöst. MuPAD (`radsimp`), Macsyma (`denest_sqrt`) und Mathematica 3.0 (`RootReduce`) stellen spezielle Simplifikationsroutinen für geschachtelte Wurzelausdrücke zur Verfügung, was bei Mathematica im stärkeren `FullSimplify` integriert ist. Mit Reduce und Axiom habe ich keinen direkten Weg gefunden, diese Aufgabe zu lösen.[1].

Axiom	unbekannt
Derive	automatisch
Macsyma	`denest_sqrt`
Maple	automatisch
Mathematica	`RootReduce`
MuPAD	`radsimp`
Reduce	nutzerdefiniertes Regelsystem

Tabelle 1: Vereinfachung geschachtelter Wurzelausdrücke

Gleichwohl hatten wir gesehen, dass solche Wurzelausdrücke im symbolischen Rechnen eine wichtige Rolle spielen, weil durch sie "interessante" reelle Zahlen dargestellt werden, die z.B. zur exakten Beschreibung von Nullstellen benötigt werden. Wir wollen deshalb zunächst untersuchen,

[1] Dieses Wissen kann man allerdings wiederum wenigstens Reduce mit einem einfachen Regelsystem beibringen.

wie die einzelnen Systeme mit univariaten Polynomen, in deren Koeffizienten solche Wurzelausdrücke auftreten, zurechtkommen. Wir werden dazu im folgenden mit Hilfe der verschiedenen CAS zunächst exemplarisch, ausgehend vom Polynom $f(x) = x^3+x+1$ dessen drei Nullstellen x_1, x_2, x_3 bestimmen, danach aus diesen von dem entsprechenden System selbst erzeugten Wurzelausdrücken das Produkt $(x-x_1)(x-x_2)(x-x_3)$ formen und schließlich versuchen, dieses Produkt durch Ausmultiplizieren und Vereinfachen wieder in das Ausgangspolynom $f(x)$ zu verwandeln. Schließlich werden wir versuchen, kompliziertere polynomiale Ausdrücke in x_1, x_2, x_3 zu vereinfachen.

Diese Rechnungen, ausgeführt in verschiedenen großen CAS allgemeiner Ausrichtung, sollen zugleich ein wenig von der Art und Weise vermitteln, wie man in jedem der Systeme die entsprechenden Rechnungen veranlassen kann.

5.1 Rechnen mit Nullstellen dritten Grades

5.1.1 Nullstellen dritten Grads mit Derive

Derive stellt im Gegensatz zu den anderen großen CAS im Rahmen seines Interpreters nur einen sehr eingeschränkten Befehlssatz bereit, den der Nutzer im Normalfall auch nicht verwenden soll. Statt dessen erfolgt der Dialog über eine menügesteuerte Oberfläche, die verschiedene Aktionen auf dem Arbeitsblatt ermöglicht. Dazu ist es insbesondere auf einfache Weise möglich, über Kursortasten auf beliebige Teilausdrücke bisher erzeugter Ausgaben (entsprechend dem jeweiligen Parse-Baum) zuzugreifen.

Wir wollen deshalb die Elemente dieser menügesteuerten Oberfläche zur Ausführung unserer Rechnungen verwenden. Die Namen der entsprechenden Knöpfe sind im folgenden als Funktionsaufruf dargestellt. Derive (Version 3.04) liefert die folgenden Antworten:

```
1: x^3+x+1

2: factor complex #1: (0.4 s)
```

$$\begin{aligned}&\left(x + \left(\tfrac{1}{18}\sqrt{93}+\tfrac{1}{2}\right)^{1/3} - \left(\tfrac{1}{18}\sqrt{93}-\tfrac{1}{2}\right)^{1/3}\right)\\ &\quad\cdot\Big(x + \left(\tfrac{1}{144}\sqrt{93}-\tfrac{1}{16}\right)^{1/3} - \left(\tfrac{1}{144}\sqrt{93}+\tfrac{1}{16}\right)^{1/3}\\ &\qquad -i\left(\left(\tfrac{3}{128}\sqrt{93}+\tfrac{29}{128}\right)^{1/6}+\left(\tfrac{29}{128}-\tfrac{3}{128}\sqrt{93}\right)^{1/6}\right)\Big)\\ &\quad\cdot\Big(x + \left(\tfrac{1}{144}\sqrt{93}-\tfrac{1}{16}\right)^{1/3} - \left(\tfrac{1}{144}\sqrt{93}+\tfrac{1}{16}\right)^{1/3}\\ &\qquad +i\left(\left(\tfrac{3}{128}\sqrt{93}+\tfrac{29}{128}\right)^{1/6}+\left(\tfrac{29}{128}-\tfrac{3}{128}\sqrt{93}\right)^{1/6}\right)\Big)\end{aligned}$$

```
3: expand #2 (1.3 s)
```

$$\begin{aligned}&x^3 + x\left(\tfrac{1}{1152}\sqrt{93}+\tfrac{29}{3456}\right)^{1/3} + x\left(\tfrac{29}{3456}-\tfrac{1}{1152}\sqrt{93}\right)^{1/3} + x\left(\tfrac{3}{128}\sqrt{93}+\tfrac{29}{128}\right)^{1/3}\\ &\quad + x\left(\tfrac{29}{128}-\tfrac{3}{128}\sqrt{93}\right)^{1/3} - x\left(\tfrac{1}{18}\sqrt{93}+\tfrac{29}{54}\right)^{1/3} - x\left(\tfrac{29}{54}-\tfrac{1}{18}\sqrt{93}\right)^{1/3}\\ &\quad + x + \left(\tfrac{7}{7776}\sqrt{93}+\tfrac{5}{576}\right)^{1/3} + \left(\tfrac{1}{31104}\sqrt{93}-\tfrac{1}{3456}\right)^{1/3} + \left(\tfrac{7}{288}\sqrt{93}+\tfrac{15}{64}\right)^{1/3}\\ &\quad + \left(\tfrac{1}{1152}\sqrt{93}-\tfrac{1}{128}\right)^{1/3} + \left(\tfrac{1}{486}\sqrt{93}+\tfrac{1}{54}\right)^{1/3} - \left(\tfrac{1}{31104}\sqrt{93}+\tfrac{1}{3456}\right)^{1/3}\\ &\quad - \left(\tfrac{7}{7776}\sqrt{93}-\tfrac{5}{576}\right)^{1/3} - \left(\tfrac{1}{1152}\sqrt{93}+\tfrac{1}{128}\right)^{1/3} - \left(\tfrac{7}{288}\sqrt{93}-\tfrac{15}{64}\right)^{1/3}\\ &\quad - \left(\tfrac{1}{486}\sqrt{93}-\tfrac{1}{54}\right)^{1/3}\end{aligned}$$

Analysieren wir dieses Polynom in x mit Koeffizienten aus einem offensichtlich komplizierten Teilring der komplexen Zahlen mit unserem bisherigen Wissen über den strukturellen Aufbau von Ausdrücken näher.

Die dritten Wurzeln $(U)^{\frac{1}{3}}$ sind intern wohl als Funktionsausdrücke der Form `expt(U,1/3)` dargestellt, so dass es sich bei diesem univariaten Polynom in Wirklichkeit um ein *Polynom in mehreren Kernen* handelt, zwischen denen aber im Gegensatz zu den trigonometrischen Ausdrücken, die wir im letzten Kapitel betrachtet hatten, stärkere algebraische Relationen bestehen. Außerdem ist der Radikand eines solchen Wurzelausdrucks selbst wieder von komplizierterer polynomialer Gestalt.

> Das Wurzelsymbol wirkt damit, wie generell das Funktionssymbol eines Kerns, wie eine Trennwand zwischen dem (äußeren) rationalen Ausdruck, in den der Wurzelausdruck als Kern eingeht, und dem (inneren) rationalen Ausdrucken, der als Radikand, d.h. als Argument des Funktionsausdrucks, auftritt.

Die algebraischen Abhängigkeiten zwischen verschiedenen Kernen mit komplizierter innerer Struktur herzuleiten ist schwierig. Selbst wenn es sich wie hier um Kerne handelt, die alle aus demselben Funktionssymbol `expt` hervorgegangen sind, kann man entsprechende Funktionstransformationen nur vornehmen, wenn die Argumente in einer für die jeweilige Transformation geeigneten Form vorliegen, nicht aber (nur) in einer semantisch äquivalenten.

Hier ist die Situation jedoch deutlich einfacher, da in obigem Ausdruck (bis auf rationale Vielfache des Radikanden) im wesentlichen nur zwei verschiedene dritte Wurzeln auftreten. Es sollte also eigentlich nicht so schwierig sein, diesen Ausdruck zu simplifizieren. Jedoch Fehlanzeige! Offensichtlich werden von Derive intern die Kerne der Form `expt([expr],1/3)` *sämtlich* als verschieden angesehen, was bei einer Interpretation als komplexe Zahlen, ohne die Geschichte ihrer Entstehung zu kennen, auch korrekt wäre, da die entsprechende komplexe Funktion `expt` mehrdeutig ist.

```
4: simplify #3 (16.1 s)
```

$$
\begin{aligned}
x^3 + x\Big(&\big(\tfrac{29}{3456} + \tfrac{1}{1152}\sqrt{93}\big)^{1/3} + \big(\tfrac{29}{3456} - \tfrac{1}{1152}\sqrt{93}\big)^{1/3} + \big(\tfrac{3}{128}\sqrt{93} + \tfrac{29}{128}\big)^{1/3} \\
&+ \big(\tfrac{29}{128} - \tfrac{3}{128}\sqrt{93}\big)^{1/3} - \big(\tfrac{1}{18}\sqrt{93} + \tfrac{29}{54}\big)^{1/3} - \big(\tfrac{29}{54} - \tfrac{1}{18}\sqrt{93}\big)^{1/3} + 1\Big) \\
&- \big(\tfrac{1}{31104}\sqrt{93} + \tfrac{1}{3456}\big)^{1/3} + \big(\tfrac{1}{31104}\sqrt{93} - \tfrac{1}{3456}\big)^{1/3} - \big(\tfrac{1}{1152}\sqrt{93} + \tfrac{1}{128}\big)^{1/3} \\
&+ \big(\tfrac{1}{1152}\sqrt{93} - \tfrac{1}{128}\big)^{1/3} - \big(\tfrac{7}{7776}\sqrt{93} - \tfrac{5}{576}\big)^{1/3} + \big(\tfrac{1}{486}\sqrt{93} + \tfrac{1}{54}\big)^{1/3} \\
&- \big(\tfrac{1}{486}\sqrt{93} - \tfrac{1}{54}\big)^{1/3} - \big(\tfrac{7}{288}\sqrt{93} - \tfrac{15}{64}\big)^{1/3} + \big(\tfrac{14}{243}\sqrt{93} + \tfrac{5}{9}\big)^{1/3}
\end{aligned}
$$

Eine numerische Näherung der Summen der hier versammelten Wurzelausdrücke

```
5: approximate #4 (0.1 s)
```

$$x^3 + x + 1$$

zeigt gleichwohl, dass unsere Rechnungen bis hierher richtig waren, also einzig das Simplifizieren der entsprechenden Ausdrücke die Fähigkeiten des CAS übersteigt, obwohl Derive beim Vereinfachen geschachtelter Quadratwurzeln noch so gut abschnitt. Das Hauptproblem beim Simplifizieren dieser Ausdrücke besteht, wie bereits angemerkt, darin, dass sich Derive offensichtlich nicht merkt, dass die im wesentlichen gleichen Kerne der Gestalt $(29 \pm 3\sqrt{93})^{\frac{1}{3}}$ von derselben Quelle herrühren. Damit wird die in diesem Fall sogar rationale Abhängigkeit, die zwischen ihnen besteht, nicht verwendet.

5.1.2 Nullstellen dritten Grades mit Maple und MuPAD

Im weiteren wollen wir die sprachlichen Mittel, die die einzelnen Interpreter anbieten, zur Formulierung von Anfragen intensiver nutzen. So werden wir etwa die in allen Systemen vorhandene

Funktion `solve` verwenden, mit der man die Nullstellen von Polynomen und allgemeineren Gleichungen und Gleichungssystemen bestimmen kann. Allerdings ist das Ausgabeformat dieser Funktion von System zu System verschieden und differiert selbst zwischen unterschiedlichen Typen von Gleichungen und Gleichungssystemen.

MAPLE liefert etwa für univariate Polynome, die wir in diesem Abschnitt untersuchen, eine *Ausdruckssequenz* zurück, die die verschiedenen Nullstellen zusammenfasst. Wir wollen sie durch Einklammern `[...]` gleich in eine Liste verwandeln:

```
s:=[solve(x^3+x+1,x)];
```

$$[-1/6\,\%1+2\,\%2,\ 1/12\,\%1-\tfrac{1}{\%1}+1/2\,I\sqrt{3}\,(-1/6\,\%1-2\,\%2)\,,$$
$$1/12\,\%1-\tfrac{1}{\%1}-1/2\,I\sqrt{3}\,(-1/6\,\%1-2\,\%2)]$$

$$\%1 := \sqrt[3]{108+12\sqrt{93}}$$
$$\%2 := \frac{1}{\sqrt[3]{108+12\sqrt{93}}}$$

Wir sehen, dass MAPLE das Problem der voneinander abhängigen Kerne auf eine clevere Art und Weise löst: Das Endergebnis wird unter Verwendung von neuen Variablen %1 und %2 formuliert, mit denen ein gemeinsamer Teilausdruck, der in der Formel mehrfach vorkommt, identifiziert wird. Auf diese Information wird bei späteren Simplifikationen zurückgegriffen, was für unsere Aufgabenstellung bereits ausreichend ist:

```
p:=product(x-op(i,s),i=1..3);
```

$$(x+1/6\,\%1-2\,\%2)\left(x-1/12\,\%1+\%2-1/2\,I\sqrt{3}\,(-1/6\,\%1-2\,\%2)\right)$$
$$\left(x-1/12\,\%1+\%2+1/2\,I\sqrt{3}\,(-1/6\,\%1-2\,\%2)\right)$$

```
p1:=expand(p);
```

$$1/2+x+x^3+\frac{1}{18}\sqrt{93}-\frac{8}{108+12\sqrt{93}}$$

Expand ist ein polynomialer Normalformoperator, d.h. er multipliziert Produkte aus und fasst gleichartige Terme zusammen, wobei x und %1 als Kerne betrachtet werden. Die separierten Teilausdrücke können vom Nutzer angesprochen werden, so dass wir diese Umformungen nachvollziehen können:

```
%1;
```

$$\sqrt[3]{108+12\sqrt{93}}$$

```
s1:=subs(%1=u,%2=1/u,s);
```

$$s1 := [-1/6\,u+2\,u^{-1}, 1/12\,u-u^{-1}+1/2\,I\sqrt{3}\left(-1/6\,u-2\,u^{-1}\right),$$
$$1/12\,u-u^{-1}-1/2\,I\sqrt{3}\left(-1/6\,u-2\,u^{-1}\right)]$$

```
expand(product(x-op(i,s1),i=1..3));
```

$$x+x^3+\frac{1}{216}\,u^3-8\,u^{-3}$$

Durch Rationalmachen, d.h. durch Multiplizieren mit dem zu u konjugierten Wurzelausdruck, kann man das Absolutglied des Polynoms p weiter vereinfachen. Diese Technik ist jedoch ein spezieller mathematischer Trick, der in der polynomialen Normalformbildung rationaler Ausdrücke nicht enthalten ist. Er wird folglich bei Anwendung von *expand* auch nicht ausgeführt. Die folgende Anweisung führt schließlich zum gewünschten Ergebnis.

```
simplify(p);
```

$$x^3 + x + 1$$

Experimentieren wir weiter mit den Ausdrücken aus der Sequenz s. Es stellt sich heraus, dass Potenzsummen der drei Wurzeln stets ganzzahlig sind. Wir wollen wiederum prüfen, wieweit MAPLE in der Lage ist, dies zu erkennen. Dazu sammeln wir in einer Liste die Ergebnisse der Vereinfachung von $x_1^k + x_2^k + x_3^k$ für $k = 2, \ldots, 9$ auf und versuchen Sie danach weiter auszuwerten.

```
a:=[seq(simplify(sum(s[i]^k,i=1 .. 3)),k=2 .. 9)];
```

$$[-2, -3, 2, 5, 6\,\frac{29+3\,\sqrt{3}\sqrt{31}}{\left(9+\sqrt{3}\sqrt{31}\right)^2}, -7, -36\,\frac{29+3\,\sqrt{3}\sqrt{31}}{\left(9+\sqrt{3}\sqrt{31}\right)^2}, 144\,\frac{135+14\,\sqrt{3}\sqrt{31}}{\left(9+\sqrt{3}\sqrt{31}\right)^3}]$$

```
map(evalf,a);
        [-2., -3., 2., 5., 1.000000000, -7., -6.000000001, 6.000000002]

map(normal,a,expanded);
                              [-2, -3, 2, 5, 1, -7, -6, 6]
```

Wir sehen, dass die Potenzsummen offensichtlich immer ganzzahlig sind, MAPLE jedoch nicht immer in der Lage ist, mit dem Aufruf von `simplify` Nenner rational zu machen. Erst die explizite Aufforderung, die Nenner der einzelnen Ausdrücke zu expandieren (die man glücklicherweise und trickreich formulieren kann), führt zum erwarteten Ergebnis. Nebenbei sei bemerkt, dass hier genau dieselben Umformungen notwendig sind wie zur Vereinfachung geschachtelter Wurzelausdrücke, die auch MAPLE vorhin noch so bravourös von allein bewerkstelligt hat.

Ähnlich ist in MUPAD vorzugehen:

```
s:=solve(x^3+x+1,x);
s1:=_mult(x-op(s,i) $i=1..3);
expand(s1);
```

$$x^3 + 3x\sqrt[3]{\left(\frac{\sqrt{31}\sqrt{108}}{108} + 1/2\right)}\sqrt[3]{\left(\frac{\sqrt{31}\sqrt{108}}{108} - 1/2\right)} + 1$$

```
simplify(%);
```

$$x^3 + 3x\sqrt[3]{\left(\frac{\sqrt{93}}{18} + 1/2\right)}\sqrt[3]{\left(\frac{\sqrt{93}}{18} - 1/2\right)} + 1$$

Die Ergebnisse der ersten beiden Aufrufe sind hier weggelassen; sie haben ähnlich kompliziertes Aussehen wie in MAPLE. Auch hier werden gemeinsame Teilausdrücke separiert, wie man mit der (undokumentierten) Funktion `rationalize` erkennt:

```
rationalize(s);
```

$$\left\{\frac{\mathrm{X73}}{2} - \frac{\mathrm{X74}}{2} + \frac{\mathrm{X72X75}\,(\mathrm{X73}+\mathrm{X74})}{2}, \frac{\mathrm{X73}}{2} - \frac{\mathrm{X74}}{2} - \frac{\mathrm{X72X75}\,(\mathrm{X73}+\mathrm{X74})}{2}, -\mathrm{X73}+\mathrm{X74}\right\},$$

$$\left\{\mathrm{X75} = \sqrt{3}, \mathrm{X72} = i, \mathrm{X73} = \sqrt[3]{\frac{\sqrt{31}\sqrt{108}}{108} + 1/2}, \mathrm{X74} = \sqrt[3]{\frac{\sqrt{31}\sqrt{108}}{108} - 1/2}\right\}$$

Das Ergebnis von `expand` zeigt eine weitere Schwierigkeit, die hier aus der mathematischen Strenge resultiert: Die beiden Wurzelausdrücke, die im übrigen aus einer gemeinsamen Quelle stammen, wie wir weiter unten noch sehen werden, werden nicht zusammengefasst, weil das im Komplexen im Prinzip falsch werden kann. Diese Funktionstransformation, die Verwandlung der "äußeren" in eine "innere" Multiplikation, kann durch `radsimp` bewerkstelligt werden, was aber eine sehr zeitaufwendige Sache ist. Auch für die Berechnung der Potenzsummen der Nullstellen benötigen wir diese Vereinfachung und erhalten (allerdings erst nach 216 s.)

```
[radsimp(_plus(op(s,i)^k $ i=1 .. 3)) $ k=2 .. 9];
```

```
[-2, -3, 2, 5, 1, -7, -6, 6]
```

5.1.3 Nullstellen dritten Grads mit Mathematica, Macsyma und Axiom

Mathematica hat seinen Gleichungslöser in der neuen Version 3.0 gründlich überarbeitet und liefert für unsere Gleichung dritten Grades folgende Antwort:

```
s=Solve[x^3+x+1==0,x]
```

$$\left\{ \{x \to -\left(\frac{2}{3\left(-9+\sqrt{93}\right)}\right)^{\frac{1}{3}} + \frac{\left(\frac{-9+\sqrt{93}}{2}\right)^{\frac{1}{3}}}{3^{\frac{2}{3}}}\}, \{x \to \frac{-\left(1+i\sqrt{3}\right)\left(\frac{-9+\sqrt{93}}{2}\right)^{\frac{1}{3}}}{2\,3^{\frac{2}{3}}} + \frac{1-i\sqrt{3}}{2^{\frac{2}{3}}\left(3\left(-9+\sqrt{93}\right)\right)^{\frac{1}{3}}}\}, \{x \to \frac{-\left(1-i\sqrt{3}\right)\left(\frac{-9+\sqrt{93}}{2}\right)^{\frac{1}{3}}}{2\,3^{\frac{2}{3}}} + \frac{1+i\sqrt{3}}{2^{\frac{2}{3}}\left(3\left(-9+\sqrt{93}\right)\right)^{\frac{1}{3}}}\} \right\}$$

Das Ergebnis des `Solve`-Operators ist keine Liste oder Menge von Lösungen, sondern eine Substitutionsliste, wie wir sie im Abschnitt 3.6. kennen gelernt haben. Solche Substitutionslisten werden auch von Maple und MuPAD für Gleichungssysteme in mehreren Veränderlichen verwendet, wo man die einzelnen Lösungskomponenten verschiedenen Variablen zuordnen muss. Mathematicas Ausgabeformat ist in dieser Hinsicht also, wie auch das von Axiom, Macsyma und Reduce, konsistenter.

Die einzelnen Komponenten der Lösung können wir durch die Aufrufe `x /. s[[i]]` erzeugen, welche jeweils die in der Lösungsliste aufgesammelten Substitutionen ausführen. Diese darf sich im Produkt $\prod_{i=1}^{3}(x-x_i)$ natürlich nur auf die Berechnung von x_i ausdehnen. Durch entsprechende Klammerung erreichen wir, dass im Ausdruck `x-(x /. s[[i]])` das erste x als Symbol erhalten bleibt, während für das zweite x die entsprechende Ersetzung $x \mapsto x_i$ ausgeführt wird.

Hängen wir noch eine Simplifikation an, so erhalten wir bereits das Polynom $f(x)$, von dem wir ausgegangen waren, zurück.

```
Product[x-(x /. s[[i]]),{i,1,3}]//Simplify
```

$$1+x+x^3$$

Auch für die Potenzsummen der Nullstellen, die man mit dem Sequenzierungsoperator `Table` aufsammeln kann, bekommt man auf diesem Wege eine Liste von ganzen Zahlen:

```
Table[Simplify[Sum[(x/.s[[i]])^k,{i,1,3}]],{k,2,9}]
```

$$\{-2, -3, 2, 5, 1, -7, -6, 6\}$$

Ähnlich gehen die Systeme Axiom und Macsyma vor, wobei hier oft eine genauere Kenntnis speziellerer Designmomente notwendig ist. So liefert etwa Axiom mit

```
s:=solve(x^3+x+1)
```

das etwas verwunderliche Ergebnis $[x^3 + x + 1 = 0]$. Erst die explizite Aufforderung nach einer Lösung in Radikalen

```
s:=radicalSolve(x^3+x+1)
```

$$\left[x = \frac{\left(-3\ \sqrt{-3}+3\right)\ \sqrt[3]{\frac{\sqrt{\frac{31}{3}}-3}{6}}^{\,2}+2}{\left(3\ \sqrt{-3}+3\right)\ \sqrt[3]{\frac{\sqrt{\frac{31}{3}}-3}{6}}},\ x = \frac{\left(-3\ \sqrt{-3}-3\right)\ \sqrt[3]{\frac{\sqrt{\frac{31}{3}}-3}{6}}^{\,2}-2}{\left(3\ \sqrt{-3}-3\right)\ \sqrt[3]{\frac{\sqrt{\frac{31}{3}}-3}{6}}},\ x = \frac{3\ \sqrt[3]{\frac{\sqrt{\frac{31}{3}}-3}{6}}^{\,2}-1}{3\ \sqrt[3]{\frac{\sqrt{\frac{31}{3}}-3}{6}}}\right]$$

wartet mit einem Ergebnis auf, das dem der anderen Systeme ähnelt. Die Vereinfachung des Produkts sowie die Berechnung der Potenzsummen geschieht ähnlich wie in Maple über die Konversion von Listen in Produkte bzw. Summen und bedarf keiner zusätzlichen Simplifikationsaufrufe:

```
reduce(*,[x-subst(x,s.i) for i in 1..3])

       3
      x  + x + 1
                                        Type: Expression Integer

[reduce(+,[subst(x**k,s.i) for i in 1..3]) for k in 2..9]

      [- 2,- 3,2,5,1,- 7,- 6,6]
                                        Type: List Expression Integer
```

Macsyma[2] liefert als Ergebnis des `solve`-Operators die Lösungsmenge in Form einer Substitutionsliste in ähnlicher Form wie auch Axiom.

```
s:solve(x^3+x+1,x);
```

$$\left\{x = \sqrt[3]{\frac{\sqrt{31}}{6\,\sqrt{3}}-\frac{1}{2}}\ \left(-\frac{\sqrt{3}\,i}{2}-\frac{1}{2}\right) - \frac{\frac{\sqrt{3}\,i}{2}-1/2}{3\ \sqrt[3]{\frac{\sqrt{31}}{6\,\sqrt{3}}-1/2}},\ x = \sqrt[3]{\frac{\sqrt{31}}{6\,\sqrt{3}}-\frac{1}{2}}\ \left(\frac{\sqrt{3}\,i}{2}-\frac{1}{2}\right) - \frac{-\frac{\sqrt{3}\,i}{2}-1/2}{3\ \sqrt[3]{\frac{\sqrt{31}}{6\,\sqrt{3}}-1/2}},\ x = \sqrt[3]{\frac{\sqrt{31}}{6\,\sqrt{3}}-\frac{1}{2}} - \frac{1}{3\ \sqrt[3]{\frac{\sqrt{31}}{6\,\sqrt{3}}-1/2}}\right\}$$

[2] In Macsyma ist der Doppelpunkt der Zuweisungsoperator

```
s1:product(x-subst(part(s,i),x),i,1,3);
```

$$\left(x-\sqrt[3]{\frac{\sqrt{31}}{6\,\sqrt{3}}-\frac{1}{2}}+\frac{1}{3\,\sqrt[3]{\frac{\sqrt{31}}{6\,\sqrt{3}}-1/2}}\right)$$
$$\left(x+\frac{\frac{\sqrt{3}\,i}{2}-1/2}{3\,\sqrt[3]{\frac{\sqrt{31}}{6\,\sqrt{3}}-1/2}}-\sqrt[3]{\frac{\sqrt{31}}{6\,\sqrt{3}}-\frac{1}{2}}\,\left(-\frac{\sqrt{3}\,i}{2}-\frac{1}{2}\right)\right)$$
$$\left(x-\sqrt[3]{\frac{\sqrt{31}}{6\,\sqrt{3}}-\frac{1}{2}}\,\left(\frac{\sqrt{3}\,i}{2}-\frac{1}{2}\right)+\frac{-\frac{\sqrt{3}\,i}{2}-1/2}{3\,\sqrt[3]{\frac{\sqrt{31}}{6\,\sqrt{3}}-1/2}}\right)$$

Eine Vereinfachung muss wie in MAPLE und MUPAD explizit angestoßen werden, wozu es die Funktion `ev` (evaluate) gibt, die mit verschiedenen Argumenten zu verschiedenem Simplifikationsverhalten veranlasst werden kann. Hier hilft der Zusatz `radcan` weiter, der eine Vereinfachung von Radikalen anstößt:

```
ev(s1,radcan);
```

$$x^3+x+1$$

Auf dieselbe Weise lassen sich die Potenzsummen berechnen:

```
s2:ev(makelist(sum(subst(part(s,i),x)^k,i,1,3),k,2,9),radcan);
```

$$[-2,-3,2,5,1,-7,-6,6]$$

An diesem Beispiel wird der funktionale Charakter der im symbolischen Rechnen verwendeten Programmiersprachen sehr deutlich. Natürlich hätte man, und sei es zur besseren Lesbarkeit, auch die einzelnen Schritte zur Konstruktion der Liste der zu evaluierenden Ausdrücke nacheinander ausführen und die jeweiligen Zwischenergebnisse in Variablen speichern können. Listen sind jedoch eine sehr bequeme Datenstruktur, mit der sich auch größere (homogene) Datenmengen wie in diesem Fall die verschiedenen Potenzsummen von einer Funktion an die nächste übergeben lassen.

5.2 Die allgemeine Lösung einer Gleichungen dritten Grades

Wir haben bei der Analyse der bisherigen Ergebnisse nur darauf geachtet, ob die verschiedenen Systeme mit den produzierten Größen auch vernünftig umgehen konnten. Obwohl das Aussehen der Lösung von System zu System sehr differierte, schienen die Ergebnisse semantisch äquivalent zu sein, da die abgeleiteten Resultate, die jeweiligen Potenzsummen, zu denselben ganzen Zahlen vereinfacht werden konnten. Um zu beurteilen, wie angemessen die jeweiligen Antworten ausfielen, müssen wir deshalb zunächst Erwartungen an die Gestalt der jeweiligen Antwort formulieren.

Wir wollen deshalb als Intermezzo jetzt das Verfahren zur exakten Berechnung der Nullstellen von Gleichungen dritten Grades als geschachtelte Wurzelausdrücke kennenlernen und dann vergleichen, inwiefern die verschiedenen Systeme auch inhaltlich zufriedenstellende Antworten geben.

Aus dem Grundkurs Algebra ist bekannt, dass jedes Polynom dritten Grades $f(x)=x^3+a\,x^2+b\,x+c$ mit reellen Koeffizienten drei komplexe Nullstellen besitzt, von denen aus Stetigkeitsgründen wenigstens eine reell ist. Die beiden anderen Nullstellen sind entweder ebenfalls reelle Zahlen oder aber zueinander konjugierte komplexe Zahlen.

Um eine allgemeine Darstellung der Nullstellen durch Wurzelausdrücke in a,b,c zu erhalten, überführt man das Polynom f zunächst durch die Substitution $x\mapsto y-\frac{a}{3}$ in die *reduzierte Form* (alle Rechnungen mit MUPAD)

```
f:=x^3+a*x^2+b*x+c:
collect(subs(f,x=y-a/3),y);
```

$$y^3 + y\left(b - \frac{a^2}{3}\right) + \left(c - \frac{ab}{3} + \frac{2a^3}{27}\right) = y^3 + p\,y + q$$

Diese Form hängt nur noch von den zwei Parametern $p = b - \frac{a^2}{3}$ und $q = c - \frac{ab}{3} + \frac{2a^3}{27}$ ab. Die Lösung der reduzierten Gleichung $g = y^3 + py + q$ suchen wir in der Form $y = u + v$, wobei wir die Aufteilung in zwei Summanden so vornehmen wollen, das eine noch zu spezifizierende Nebenbedingung erfüllt ist.

```
g:=y^3+p*y+q;
expand(subs(g,y=u+v));
```

$$q + pu + pv + u^3 + v^3 + 3uv^2 + 3u^2v$$

Diesen Ausdruck formen wir um zu

$$(u+v)(3uv+p) + (u^3 + v^3 + q)$$

und wählen u und v so, dass

$$3uv + p = u^3 + v^3 + q = 0$$

gilt. Aus der ersten Beziehung erhalten wir $v = \frac{-p}{3u}$, welches, in die zweite Gleichung eingesetzt, nach kurzer Umformung eine quadratische Gleichung für u^3 ergibt:

$$\begin{aligned} u^3 + v^3 + q &= u^3 - \tfrac{p^3}{27u^3} + q = 0 \\ \iff \quad & (u^3)^2 + q \cdot u^3 - \tfrac{p^3}{27} = 0 \end{aligned}$$

Die Diskriminante D dieser Gleichung ist

$$D = \left(\frac{q}{2}\right)^2 + \left(\frac{p}{3}\right)^3 \qquad \left(= -\frac{abc}{6} + \frac{b^3}{27} + \frac{c^2}{4} + \frac{a^3c}{27} - \frac{a^2b^2}{108}\right)$$

und je nach Vorzeichen von D erhalten wir für u^3 zwei reelle Lösungen $u^3 = -\frac{q}{2} \pm \sqrt{D}$ (falls $D > 0$), eine reelle Doppellösung $u^3 = -\frac{q}{2}$ (falls $D = 0$) oder zwei zueinander konjugierte komplexe Lösungen $u^3 = -\frac{q}{2} \pm i \cdot \sqrt{-D}$ (falls $D < 0$).

Betrachten wir zunächst den Fall $\mathbf{D \geq 0}$. Zur Ermittlung der reellen Lösung können wir die (eindeutige) reelle dritte Wurzel ziehen und erhalten $u_1 = \sqrt[3]{-\frac{q}{2} \pm \sqrt{D}}$ und nach geeigneter Erweiterung des Nenners $v_1 = \frac{-p}{3u} = \sqrt[3]{-\frac{q}{2} \mp \sqrt{D}}$. Beide Lösungen liefern also ein und denselben Wert

$$y_1 = \sqrt[3]{-\frac{q}{2} + \sqrt{D}} + \sqrt[3]{-\frac{q}{2} - \sqrt{D}}.$$

Diese Formel nennt man die *Cardanosche Formel* für die Gestalt der reellen Lösung einer reduzierten Gleichung dritten Grades. Die beiden komplexen Lösungen erhält man, wenn man mit den entsprechenden komplexen dritten Wurzeln für u startet. Diese unterscheiden sich von u_1 nur durch eine dritte Einheitswurzel $\omega = -\frac{1}{2} + \frac{i}{2}\sqrt{3}$ oder $\omega^2 = \bar{\omega} = -\frac{1}{2} - \frac{i}{2}\sqrt{3}$. Wegen $u_1v_1 = u_2v_2 = u_3v_3$ erhalten wir

$$\begin{aligned} u_2 &= \omega u_1,\ v_2 = \bar{\omega} v_1 \\ u_3 &= \bar{\omega} u_1,\ v_3 = \omega v_1 \end{aligned}$$

Zusammen ergeben sich die beiden komplexen Lösungen als

$$y_{2,3} = -\frac{u_1 + v_1}{2} \pm \frac{i}{2}\sqrt{3}(u_1 - v_1).$$

Im Falle $D = 0$ gilt $u_1 = v_1$, so dass die beiden komplexen Lösungen zu einer reellen Doppellösung zusammenfallen.

Wesentlich interessanter fällt der Fall $\mathbf{D < 0}$ aus. Man beachte zunächst, dass wegen $D = \left(\frac{q}{2}\right)^2 + \left(\frac{p}{3}\right)^3$ dann $p < 0$ gilt. Wollen wir aus obiger Gleichung für u^3 die Lösungen für u berechnen, so haben wir aus einer komplexen Zahl die dritte Wurzel zu ziehen. Kombinieren wir die Ergebnisse wie im ersten Fall, so stellt sich heraus, dass wir über den Umweg der komplexen Zahlen drei *reelle* Lösungen bekommen. Dieser Fall wird deshalb auch als *casus irreducibilis* bezeichnet, da er vor Einführung der komplexen Zahlen nicht aus den Cardanoschen Formeln hergeleitet werden konnte.

Führen wir die Rechnungen genauer aus. Zunächst überführen wir den Ausdruck für u^3 in die für das Wurzelziehen geeignetere trigonometrische Form.

$$\begin{aligned}
& u^3 = -\tfrac{q}{2} \pm i \cdot \sqrt{-D} = R \cdot (\cos(\phi) \pm i \cdot \sin(\phi)) \\
& \text{mit} \qquad R := \sqrt{\left(\tfrac{q}{2}\right)^2 - D} = \sqrt{-\left(\tfrac{p}{3}\right)^3} \\
& \text{und} \qquad \phi := \arccos\left(\tfrac{-q}{2R}\right)
\end{aligned}$$

Wir erhalten daraus die drei komplexen Lösungen

$$\begin{aligned}
& u_{k+1} = r \cdot (\cos(\tfrac{\phi + 2\,k\pi}{3}) \pm i \cdot \sin(\tfrac{\phi + 2\,k\pi}{3})), \;\; k = 0, 1, 2 \\
& \text{mit} \qquad r := \sqrt[3]{R} = \sqrt{-\tfrac{p}{3}}
\end{aligned}$$

Da $uv = -\frac{p}{3}$ reell ist, stellt sich ähnlich wie im ersten Fall heraus, dass u und v zueinander konjugierte komplexe Zahlen sind, womit sich bei der Berechnung von y die Imaginärteile der beiden Summanden wegheben. Wir erhalten schließlich die *trigonometrische Form* der drei reellen Nullstellen von g

$$y_{k+1} = 2r \cdot \cos(\tfrac{\phi + 2\,k\pi}{3}), \;\; k = 0, 1, 2.$$

Dass es sich bei den drei reellen Zahlen y_1, y_2, y_3 wirklich um Nullstellen von g handelt, kann man allerdings auch ohne den Umweg über komplexe Zahlen sehen: Für $y = 2\sqrt{-\frac{p}{3}}\cos(\alpha)$ gilt wegen der Produkt-Summe-Regel für trigonometrische Funktionen

$$\begin{aligned}
g(y) &= \tfrac{-8p}{3}\sqrt{-\tfrac{p}{3}}\cos^3(\alpha) + 2p\sqrt{-\tfrac{p}{3}}\cos(\alpha) + q \\
&= \tfrac{-8p}{3}\sqrt{-\tfrac{p}{3}}\left(\tfrac{\cos(3\alpha) + 3\cos(\alpha)}{4}\right) + 2p\sqrt{-\tfrac{p}{3}}\cos(\alpha) + q \\
&= \tfrac{-2p}{3}\sqrt{-\tfrac{p}{3}}\cos(3\alpha) + q = 2R\cos(3\alpha) + q.
\end{aligned}$$

Damit verschwindet $g(y_k)$ für $k = 0, 1, 2$, da ja gerade $\phi = \arccos\left(\frac{-q}{2R}\right)$ war.

Fassen wir unsere Ausführungen in dem folgenden Satz zusammen.

Satz 10 *Sei $g := y^3 + py + q$ eine reduziertes Polynom dritten Grades und $D = \left(\frac{q}{2}\right)^2 + \left(\frac{p}{3}\right)^3$ dessen Diskriminante. Dann gilt*

1. *Ist $D > 0$, so hat g eine reelle und zwei komplexe Nullstellen. Diese ergeben sich mit*

$$u_1 = \sqrt[3]{-\frac{q}{2} + \sqrt{D}}, \qquad v_1 = \sqrt[3]{-\frac{q}{2} - \sqrt{D}}$$

   aus der Cardanoschen Formel

$$y_1 = u_1 + v_1$$

   bzw. als

$$y_{2,3} = -\frac{u_1 + v_1}{2} \pm \frac{i}{2}\sqrt{3}\,(u_1 - v_1).$$

2. *Ist $D = 0$, so hat g eine einfache reelle Nullstelle $y_1 = 2\sqrt[3]{-\frac{q}{2}}$ und eine reelle Doppelnullstelle $y_{2,3} = -\sqrt[3]{-\frac{q}{2}}$.*

3. *Ist $D < 0$, so hat g mit $\phi = \arccos\left(\frac{-q}{2R}\right)$ drei reelle Nullstellen*

$$y_{k+1} = 2\sqrt{-\frac{p}{3}} \cdot \cos\left(\frac{\phi + 2\,k\pi}{3}\right),\ k = 0, 1, 2.$$

5.3 * Die allgemeine Lösung einer Gleichungen vierten Grades

Eine Darstellung durch Wurzelausdrücke existiert auch für die Nullstellen von Polynomen vierten Grades. Die nachstehenden Ausführungen folgen [14, Kap. 16].

Mit der Substitution $x \mapsto y - \frac{a}{4}$ kann man das Polynom vierten Grades $f = x^4 + ax^3 + bx^2 + cx + d$ zunächst wieder in die *reduzierte Form* $g = y^4 + py^2 + qy + r$ ohne kubischen Term überführen. Sind nun (u, v, w) drei komplexe Zahlen, die die Bedingungen

$$\begin{aligned} uvw &= -\tfrac{q}{8} \\ u^2 + v^2 + w^2 &= -\tfrac{p}{2} \\ u^2v^2 + u^2w^2 + v^2w^2 &= \tfrac{p^2 - 4r}{16} \end{aligned}$$

erfüllen, so ist $y = u + v + w$ eine Nullstelle von g. In der Tat, ersetzen wir in g die Variable y durch die angegebene Summe und gruppieren die Terme entsprechend, so erhalten wir den Ausdruck

```
g:=y^4+p*y^2+q*y+r:
expand(subs(g,y=u+v+w));
```

$$r + qu + qv + qw + 2puv + 2puw + 2pvw + u^4 + v^4 + w^4 + pu^2 + pv^2 + pw^2 + 4uv^3 + 4u^3v + 4uw^3 + 4u^3w + 4vw^3 + 4v^3w + 12uvw^2 + 12uv^2w + 12u^2vw + 6u^2v^2 + 6u^2w^2 + 6v^2w^2$$

$$= (u + v + w)(8uvw + q) + (uv + uw + vw)(4(u^2 + v^2 + w^2) + 2p) + (u^2 + v^2 + w^2)^2 + 4(u^2v^2 + u^2w^2 + v^2w^2) + p(u^2 + v^2 + w^2) + r,$$

der für alle Tripel (u, v, w), die die angegebenen Bedingungen erfüllen, verschwindet. Für ein solches Tripel sind aber nach dem Vietaschen Wurzelsatz (u^2, v^2, w^2) die drei Nullstellen der kubischen Gleichung

$$z^3 + \frac{p}{2}z^2 + \frac{p^2 - 4r}{16}z - \frac{q^2}{64} = 0$$

Sind umgekehrt z_1, z_2, z_3 Lösungen dieser kubischen Gleichung, so erfüllt jedes Tripel (u, v, w) mit $u = \pm\sqrt{z_1}, v = \pm\sqrt{z_2}, w = \pm\sqrt{z_3}$ nach dem Vietaschen Wurzelsatz die Gleichungen

$$\begin{aligned} uvw &= \pm\tfrac{q}{8} \\ u^2 + v^2 + w^2 &= -\tfrac{p}{2} \\ u^2v^2 + u^2w^2 + v^2w^2 &= \tfrac{p^2 - 4r}{16}, \end{aligned}$$

wobei vier der Vorzeichenkombinationen in der ersten Gleichung das Vorzeichen + und die restlichen das Vorzeichen − ergeben. Ist (u, v, w) ein Tripel, das als Vorzeichen in der ersten Gleichung − ergibt, so auch die Tripel $(u, -v, -w)$, $(-u, v, -w)$ und $(-u, -v, w)$. Damit sind

$$y_1 = u + v + w,\ y_2 = u - v - w,\ y_3 = -u + v - w,\ y_4 = -u - v + w$$

die vier Nullstellen des Polynoms g vierten Grades. Die allgemeine Lösung, die man ohne Mühe aus der entsprechenden allgemeinen Lösung der zugehörigen Gleichung zusammenstellen kann, ist allerdings bereits wesentlich umfangreicher, so dass es noch schwieriger als bei Gleichungen dritten Grades ist, mit den so generierten Wurzelausdrücken weiterzurechnen. Für eine Klassifizierung an Hand der Parameter p, q, r nach der Anzahl reeller Nullstellen sei etwa auf [14] verwiesen.

Nachdem die Lösungsverfahren für Gleichungen dritten und vierten Grades wenigstens seit dem 16. Jahrhundert bekannt waren, versuchten die Mathematiker lange Zeit, auch für Gleichungen höheren Grades solche allgemeinen Formeln in Radikalen für die entsprechenden Nullstellen zu finden. Erst in den Jahren 1824 und 1826 gelang es N.H. ABEL den Beweis zu erbringen, dass es solche allgemeinen Formeln nicht geben kann. Heute gibt die Theorie der Galoisgruppen, die mit jeder solchen Gleichung eine entsprechende Permutationsgruppe verbindet, Antwort, ob und wie sich die Nullstellen eines bestimmten Polynoms fünften oder höheren Grades durch Radikale ausdrücken lassen.

5.4 Die ROOTOF-Notation

Doch kehren wir zu den Polynomen dritten Grades zurück und untersuchen, wie die einzelnen CAS die verschiedenen Fälle der allgemeinen Lösungsformel behandeln.

Es stellt sich heraus, dass von den betrachteten CAS nur DERIVE und MUPAD die beiden Fälle $D > 0$ und $D < 0$ bei der Gestaltung der Ausgabe unterscheiden, während sonst (außer REDUCE) die Cardanosche Formel auch für komplexe Radikanden angesetzt wird. So antwortet MUPAD im *casus irreducibilis* wie erwartet

```
solve(x^3-3*x+1,x);
```

$$\left\{2\cos\left(\tfrac{2\pi}{9}\right), -\cos\left(\tfrac{2\pi}{9}\right) + \sqrt{3}\sin\left(\tfrac{2\pi}{9}\right), -\cos\left(\tfrac{2\pi}{9}\right) - \sqrt{3}\sin\left(\tfrac{2\pi}{9}\right)\right\}$$

MAPLEs Antwort dagegen lautet

```
s:=[solve(x^3-3*x+1,x)];
```

$$\Bigg[1/2\,\sqrt[3]{-4+4\sqrt{-3}} + 2\,\frac{1}{\sqrt[3]{-4+4\sqrt{-3}}},\\ -1/4\,\sqrt[3]{-4+4\sqrt{-3}} - \frac{1}{\sqrt[3]{-4+4\sqrt{-3}}} + 1/2\sqrt{-3}\left(1/2\,\sqrt[3]{-4+4\sqrt{-3}} - 2\,\frac{1}{\sqrt[3]{-4+4\sqrt{-3}}}\right),\\ -1/4\,\sqrt[3]{-4+4\sqrt{-3}} - \frac{1}{\sqrt[3]{-4+4\sqrt{-3}}} - 1/2\sqrt{-3}\left(1/2\,\sqrt[3]{-4+4\sqrt{-3}} - 2\,\frac{1}{\sqrt[3]{-4+4\sqrt{-3}}}\right)\Bigg]$$

Letzteres hat den Nachteil, dass man der Lösung selbst nach einer näherungsweisen Berechnung nicht ansieht, dass es sich um reelle Zahlen handelt:

```
evalf(s);
```

$$[1.532088886 - .1\,10^{-9}\,I,\ -1.879385242,\ .3472963560]$$

REDUCE erlaubt, mit dem Schalter `trigform` zwischen beiden Darstellungsformen zu wechseln, wobei standardmäßig die trigonometrische Form eingestellt ist, was zwar für $D < 0$ zu dem gewünschten Ergebnis führt

```
on fullroots;
s:=solve(x^3-3*x+1,x);
```

$$\{x = 2\cdot\cos(\tfrac{2\cdot\pi}{9}), x = -\cos(\tfrac{2\cdot\pi}{9}) + \sqrt{3}\cdot\sin(\tfrac{2\cdot\pi}{9}), x = -\cos(\tfrac{2\cdot\pi}{9}) - \sqrt{3}\cdot\sin(\tfrac{2\cdot\pi}{9})\},$$

aber im Fall $D > 0$ das unverständliche Ergebnis

```
s:=solve(x^3+3*x+1,x);
```

$$\{x = 2 \cdot \cosh(\tfrac{\operatorname{arctanh}(\sqrt{5})}{3}) \cdot i, x = -\cosh(\tfrac{\operatorname{arctanh}(\sqrt{5})}{3}) \cdot i + \sqrt{3} \cdot \sinh(\tfrac{\operatorname{arctanh}(\sqrt{5})}{3}),$$
$$x = -\cosh(\tfrac{\operatorname{arctanh}(\sqrt{5})}{3}) \cdot i - \sqrt{3} \cdot \sinh(\tfrac{\operatorname{arctanh}(\sqrt{5})}{3})\}$$

liefert.

Die explizite Verwendung der Lösungsformel für kubische Gleichungen birgt allerdings auch Schwierigkeiten für die weitere Vereinfachung der dabei entstehenden Ausdrücke. Vergessen der Entstehungsgeschichte von Teilausdrücken führt zu dem bei DERIVE beobachteten Verhalten. Erzeugt man wie in MuPAD die beiden dritten Wurzeln u, v in der Cardanoschen Formel getrennt, so bekommt man zwar im Gegensatz zu den anderen Systemen etwas angenehmere Wurzelausdrücke (insbesondere solche mit rationalen Nennern), hat aber im Nachhinein Probleme mit Ausdrücken, die Produkte dieser Wurzeln enthalten, obwohl wir wissen, dass $u\,v = -\frac{p}{3}$ gilt, d.h. die beiden Wurzelausdrücke im Wesentlichen zueinander invers sind.

Noch schwieriger wird die Entscheidung, welche Art der Darstellung der Lösung gewählt werden soll, wenn die zu betrachtende Gleichung Parameter enthält. Betrachten wir etwa die Aufgabe

```
s:=solve(x^3+a*x+1,x);
```

die von einem Parameter a abhängt. Die Lösungsdarstellung sollte mit möglichen späteren Substitutionen konsistent sein, d.h. bei einer Ersetzung

```
subs(a=3,s);
```

zur Cardanoschen Formel und bei

```
subs(a=-3,s);
```

zur trigonometrischen Darstellung verzweigen. Um dies zu erreichen, müsste man aber die Entscheidung bis zur Substitution aufschieben, d.h. `s` müsste eine Liste sein, die drei neue Symbole enthält, von denen nur bekannt ist, dass sie Lösung einer bestimmten Gleichung drittes Grades sind.

Genau ein solches Verhalten zeigt etwa REDUCE in der Standardeinstellung, d.h. ohne den Schalter `fullroots` einzuschalten:

```
solve(x^3+x+1,x);
```

liefert die auf den ersten Blick verblüffende Antwort

$$\{x = root\_of(X^3 + X + 1, X)\}$$

Es wird ein Funktionsausdruck mit dem Funktionssymbol ROOTOF und dem zugehörigen Polynom als Parameter erzeugt, das stellvertretend für die einzelnen Nullstellen dieses Polynoms steht und von dem nichts weiter bekannt ist als die Gültigkeit der Ersetzungsregel $X^3 \Rightarrow -(X+1)$.

Neben einer höheren Konsistenz der Darstellung spricht für ein solches Vorgehen auch die Tatsache, dass die Wurzeln einer Gleichung dritten Grades schon ein recht kompliziertes Aussehen haben können, aus dem sich die genannte Ersetzungsinformation nur noch schwer rekonstruieren lässt. Dies gilt erst recht für Nullstellen einer Gleichung vierten Grades. Für Nullstellen allgemeiner Gleichungen höheren Grades gibt es gar keine Darstellung in Radikalen, so dass für solche Zahlen *nur* auf eine ROOTOF-Darstellung zurückgegriffen werden kann.

Neben diesen praktischen Erwägungen ist eine solche Darstellung auch aus theoretischen Gründen günstig, denn es gilt der folgende

Satz 11 *Sei R ein Körper mit kanonischer Form und a eine Nullstelle des über R irreduziblen Polynoms $p(x) = x^k - r(x) \in R[x]$, wobei $\deg r < k$ sei. Eine solche Nullstelle nennt man eine* algebraische Zahl.

Stellt man polynomiale Ausdrücke in $R[a]$ in distributiver Form dar und wendet zusätzlich die algebraische Regel `{ a^k => r(a) }` *an, so erhält man eine kanonische Form in $R[a]$.*

BEWEIS : Mit dem beschriebenen Verfahren kann man jeden Ausdruck $U \in R[a]$ in der Form $U = \sum_{i=0}^{k-1} r_i a^i$ darstellen. Es bleibt zu zeigen, dass diese Darstellung eindeutig ist.

Wie früher auch genügt es zu zeigen, dass aus $0 = \sum_{i=0}^{k-1} r_i a^i$ bereits $r_i = 0$ für alle $i < k$ folgt. Wäre das nicht so, so wäre a eine Nullstelle des Polynoms $q(x) := \sum_{i=0}^{k-1} r_i x^i$ vom Grad $deg\ q < k$. Berechnen wir etwa mit dem Euklidschen Algorithmus $g(x) = gcd(p(x), q(x))$ und eine entsprechende Darstellung

$$g(x) = u(x)\,p(x) + v(x)\,q(x)$$

des größten gemeinsamen Teilers als Linearkombination von p und q, so sehen wir, dass a auch Nullstelle von $g(x)$ ist. $g(x)$ ist also ein nichttrivialer Teiler von $p(x)$, das aber als irreduzibel vorausgesetzt wurde. □

Es stellt sich heraus, dass $R[a]$ nicht nur ein Ring, sondern seinerseits wieder ein Körper ist, wie wir weiter unten noch genauer untersuchen werden. Wendet man das beschriebene Verfahren rekursiv an, so erhält man eine für Rechnungen sehr brauchbare Darstellung für solche algebraischen Zahlen. Zur Einführung einer neuen algebraischen Zahl a muss man dazu jeweils “nur” ein über dem bisherigen Bereich irreduzibles Polynom finden, dessen Nullstelle a ist, d.h. im wesentlichen in solchen Bereichen faktorisieren können.

Die Verwendung eines ROOTOF-Symbols ist deshalb keine Besonderheit von REDUCE, sondern wird auch von den anderen Systemen verwendet. Der `Solve`-Operator von MAPLE und MUPAD unterscheidet zwischen Nullstellen einzelner Gleichungen und Nullstellen von Gleichungssystemen. Für erstere wird angenommen, dass der Nutzer nach einer möglichst expliziten Lösung sucht und die ROOTOF-Notation erst ab Grad 4 (MAPLE) bzw. Grad 5 (MUPAD) eingesetzt. Für zweiteren Fall werden in MAPLE standardmäßig alle nichtrationalen Nullstellen durch ROOTOF-Ausdrücke dargestellt, in MUPAD dagegen werden auch Quadratwurzel-Ausdrücke erzeugt. Dieses Verhalten kann man durch Setzen eines entsprechenden Parameters (`_EnvExplicit` in MAPLE, `MaxDegree` in MUPAD) ändern. Ähnlich gehen auch REDUCE (ROOTOF ab Grad 3, wie wir in obigem Beispiel gesehen hatten) und MATHEMATICA (ROOTOF ab Grad 5, was man aber durch die Optionen `Cubics -> False` und `Quartics -> False` abstellen kann) vor. Mit DERIVE kann man keine nichtlinearen Gleichungssysteme lösen. MACSYMA (Version 420) löste die Probleme mit algebraischen Nullstellen auf seine Weise: Wenn die Nullstellen nicht mehr durch entsprechende Radikale dargestellt werden konnten, so wurde zu näherungsweisen Lösungen übergegangen. Dieses Verhalten hat sich in neueren Versionen durch die Bereitstellung zweier neuer Pakete (`algsys` und `triangsys`) bereits geändert, so dass auch MACSYMA mittlerweile eine ROOTOF-Notation unterstützt.

Eine Besonderheit des ROOTOF-Symbols im Vergleich zu anderen Funktionssymbolen ist der Umstand, dass es in den meisten Zusammenhängen eigentlich für eine zusammengesetzte Datenstruktur, etwa die Liste (oder gar Substitutionsliste) aller Nullstellen steht. Die derzeit in den verschiedenen Systemen verwendete Notation ist in dieser Hinsicht unzureichend. So ist etwa die Antwort von REDUCE auf

```
solve(x^3+x+1,x);
```

$$\{x = root\_of(X^3 + X + 1, X)\}$$

unverständlich, denn die Gleichung hat ja nicht eine, sondern drei Lösungen. Auch die Antwort von MAPLE auf die Aufgabe

```
s:=[solve({x^2+y+z-3,y^2+x+z-3,z^2+x+y-3},{x,y,z})];
```

$$[\{z = 2\,RootOf(Z^2+1), y = 2\,RootOf(Z^2+1), x = 1 - 2\,RootOf(Z^2+1)\},$$

$$\{z = RootOf(2\,Z + Z^2 + 3), y = RootOf(2\,Z + Z^2 + 3), x = RootOf(2\,Z + Z^2 + 3)\},$$

$$\{z = 2\,RootOf(Z^2+1), y = 1 - 2\,RootOf(Z^2+1), x = 2\,RootOf(Z^2+1)\},$$

$$\{z = RootOf(5 - 2\,Z + Z^2), y = 1 - RootOf(5 - 2\,Z + Z^2), x = 1 - RootOf(5 - 2\,Z + Z^2)\}]$$

ist mathematisch wenig zufriedenstellend, denn *RootOf* steht ja für *mehrere* Wurzeln, die nacheinander und teilweise nicht einmal unabhängig voneinander an den entsprechenden Stellen einzusetzen sind.

Zwar besitzen sowohl REDUCE[3] als auch MAPLE[4] entsprechende Funktionen, um diese Mehrdeutigkeiten ggf. aufzulösen, allerdings setzt deren Anwendung eine gehörige Portion Wissen über interne Abläufe voraus.

Die mit dem mathematischen Gebrauch am besten übereinstimmenden Lösungen bieten derzeit MATHEMATICA (ab Version 3.0) und AXIOM an. MATHEMATICA erzeugt für jede Nullstelle einen eigenen Funktionsausdruck. Diese werden in einer Liste in einer mit der sonstigen Notation gut harmonierenden Form aufgesammelt:

```
s=Solve[x^5+a*x+1==0,x]
```

$$\begin{aligned}&\{\{x \to \mathtt{Root}(1 + a\,\#1 + \#1^5\,\&, 1)\}, \{x \to \mathtt{Root}(1 + a\,\#1 + \#1^5\,\&, 2)\},\\ &\quad \{x \to \mathtt{Root}(1 + a\,\#1 + \#1^5\,\&, 3)\}, \{x \to \mathtt{Root}(1 + a\,\#1 + \#1^5\,\&, 4)\},\\ &\quad \{x \to \mathtt{Root}(1 + a\,\#1 + \#1^5\,\&, 5)\}\}\end{aligned}$$

Es werden für die einzelnen (vom Parameter a abhängenden) Nullstellen fünf verschiedene Funktionsausdrücke mit jeweils zwei Argumenten gebildet. Eines davon ist das Polynom, das die Nullstelle beschreibt, und zwar in einer Notation als *reine Funktion* (die eine Beschreibung, aber keinen Namen hat, siehe Abschnitt 3.5.; #1 steht hierbei für einen formalen Parameter), das zweite ein Index[5]. Ersetzt man in dieser Liste $a = 1$, so lassen sich die fünf Nullstellen in die zwei Nullstellen eines Polynoms zweiten und die drei Nullstellen eines Polynoms dritten Grades zerlegen. Diese Umformungen ergeben sich hier auf natürliche Weise:

```
s/.a->1
```

$$\begin{aligned}&\{\{x \to \frac{-1 - i\sqrt{3}}{2}\}, \{x \to \frac{-1 + i\sqrt{3}}{2}\}, \{x \to \mathtt{Root}(1 - \#1^2 + \#1^3\,\&, 1)\},\\ &\quad \{x \to \mathtt{Root}(1 - \#1^2 + \#1^3\,\&, 2)\}, \{x \to \mathtt{Root}(1 - \#1^2 + \#1^3\,\&, 3)\}, \}\end{aligned}$$

Will man dasselbe in MAPLE erreichen

```
s:=[solve(x^5+a*x+1,x)]:
s1:=subs(a=1,s);
```

$$[\mathit{RootOf}(Z^5 + Z + 1)]$$

so genügt es nicht, die Funktion `simplify` aufzurufen, sondern es bedarf eines auf die spezielle Datenstruktur zugeschnittenen Aufrufs der Funktion `allvalues`, die in der Lage ist, ein einzelnes ROOTOF-Symbol in eine Ausdruckssequenz zu expandieren, d.h. aus *einem* ROOTOF-Eintrag *mehrere* sich dahinter verbergende Nullstellen zu erzeugen, womit sich die Länge der Liste `s1` ändert ! Mit `map` können wir den Aufruf dieser Funktion an die richtigen Stellen unserer Liste s plazieren, was aber schon einige Kenntnisse der Interna von MAPLE voraussetzt.

```
map(allvalues,s1);
```

[3] Mit `on fullroots;` wird der obige Ausdruck in eine Liste von drei Nullstellen verwandelt, die durch das Funktionssymbol `one_of` geklammert sind. Dieses kann man durch die Funktion `expand_cases` expandieren.

[4] Hier hilft `map(allvalues,s)` weiter, das auf jeden einzelnen Eintrag in s die Funktion `allvalues` anwendet. Diese Funktion versucht, Nullstellen bis zum Grad 4 durch entsprechende Radikale zu ersetzen und sonst wie in MACSYMA mit Näherungslösungen weiterzuhelfen.

[5] Die Nullstellen werden in konkreten Anwendungen an Hand von Näherungswerten unterschieden.

$$[-1/2 + 1/2\sqrt{-1}\sqrt{3}, -1/2 - 1/2\sqrt{-1}\sqrt{3}, -1/6\,\%1 - 2/3\,\tfrac{1}{\%1} + 1/3,$$
$$1/12\,\%1 + 1/3\,\tfrac{1}{\%1} + 1/3 + 1/2\sqrt{-1}\sqrt{3}\left(-1/6\,\%1 + 2/3\,\tfrac{1}{\%1}\right),$$
$$1/12\,\%1 + 1/3\,\tfrac{1}{\%1} + 1/3 - 1/2\sqrt{-1}\sqrt{3}\left(-1/6\,\%1 + 2/3\,\tfrac{1}{\%1}\right)]$$

mit

$$\%1 = \sqrt[3]{100 + 12\sqrt{69}}$$

In REDUCE erfolgt eine solche Konvertierung automatisch

```
s:=solve(x^5+a*x+1,x);
```

$$\{x = root\_of(a\,X + X^5 + 1, X)\}$$

```
sub(a=1,s);
```

$$\{x = one\_of(\{\frac{\sqrt{3}\cdot i - 1}{2}, x = \frac{-(\sqrt{3}\cdot i + 1)}{2}, x = root\_of(X^3 - X^2 + 1, X)\})\}$$

Die auf diese Weise durch Expandieren aus einem einzelnen ROOTOF-Symbol entstandene Liste von Lösungen wird allerdings noch durch ein (weiteres) Funktionssymbol `one_of` zusammengehalten, das im weiteren vom Nutzer mit Hilfe der Funktion `expand_cases` aufgelöst werden kann.

Der Ansatz von AXIOM besteht schließlich darin, überhaupt kein ROOTOF-Symbol einzuführen, sondern die Lösung nur soweit aufzuspalten, wie dies im gegebenen Kontext möglich ist.

```
s:=solve(x^5+a*x+1,x)

  [x^5  + a x + 1 = 0]

       Type: List Equation Fraction Polynomial Integer

s1:=solve(x^5+x+1,x)

  [x^2  + x + 1 = 0, x^3  - x^2  + 1 = 0]

        Type: List Equation Fraction Polynomial Integer
```

Die weitere Bearbeitung erfolgt durch Aufruf entsprechender Funktionen auf zu selektierenden Teilausdrücken. So liefert etwa

```
radicalSolve(s1.1)
```

$$\left[x = \frac{-\sqrt{-3} - 1}{2},\ x = \frac{\sqrt{-3} - 1}{2}\right]$$

die beiden Nullstellen des ersten Ausdrucks.

5.5 Das Rechnen mit algebraischen Zahlen

Wir wollen nun einige aus einem Kurs zur höheren Algebra wohlbekannte Eigenschaften algebraischer Zahlen zusammentragen und zu deren Beweis und Veranschaulichung Mittel der Computeralgebra einsetzen. Da hierbei die Demonstration eines CAS als Arbeitsinstrument im Mittelpunkt steht, werden wir uns in den folgenden Ausführungen auf MAPLE und REDUCE beschränken. Jedes andere der Systeme wäre aber sicher ähnlich gut geeignet, die erforderlichen symbolischen Transformationen zu vollziehen.

5.5.1 Definitionen und Beispiele

Wir hatten bereits weiter oben den Begriff der algebraischen Zahl erwähnt und wollen an dieser Stelle eine formale Definition nachholen.

DEFINITION: Seien $k \subset K$ zwei Körper. $\alpha \in K$ bezeichnet man als *algebraisch über* k, wenn es ein Polynom $p(x) \in k[x]$ mit $p(\alpha) = 0$ gibt.

Satz 12 *Sei $\alpha \in K$ eine algebraische Zahl und*

$$P := \{q(x) \in k[x] \,:\, q(\alpha) = 0 \text{ und } lc(q) = 1\}$$

In P gibt es genau ein Polynom $p(x) \in k[x]$ minimalen Grades. $p(x)$ ist irreduzibel in $k[x]$ und alle Polynome $q(x) \in P$ enthalten $p(x)$ als Faktor.

Dieses Polynom heißt charakteristisches Polynom *oder* Minimalpolynom*, sein Grad der* Grad *der algebraischen Zahl α.*

Der Beweis dieses Satzes ergibt sich unmittelbar aus dem Satz über Division mit Rest in $k[x]$.

Beispiele (über $k = \mathbf{Q}$):

$$\alpha_1 = \sqrt{2} \qquad p_1(x) = x^2 - 2$$
$$\alpha_2 = \sqrt[3]{5} \qquad p_2(x) = x^3 - 5$$
$$\alpha_3 = \sqrt{1 - \sqrt{2}} \qquad p_3(x) = x^4 - 2\,x^2 - 1$$

Die Irreduzibilität von p_3 ist nicht ganz offensichtlich, kann aber leicht mit MAPLE getestet werden:

```
factor(x^4-2*x^2-1);
```

$$x^4 - 2\,x^2 - 1$$

Analog erhalten wir für $\alpha_4 = \sqrt{9 + 4\sqrt{5}}$ ein Polynom $p_4(x) = x^4 - 18\,x^2 + 1$, dessen Nullstelle α_4 ist. Allerdings ist p_4 nicht irreduzibel

```
factor(x^4-18*x^2+1);
```

$$(\,x^2 - 4\,x - 1\,)\,(\,x^2 + 4\,x - 1\,)$$

und α_4 als Nullstelle des ersten Faktors in Wirklichkeit eine algebraische Zahl vom Grad 2. Das weiß MAPLE auch:

```
sqrt(9+4*sqrt(5));
```

$$\sqrt{5} + 2$$

Andere Beispiele algebraischer Zahlen hängen mit Winkelfunktionen spezieller Argumente zusammen. Aus der Schule bekannt sind die Werte von $\sin(x)$ und $\cos(x)$ für $x = \frac{\pi}{n}$ mit $n = 3, 4, 6$. MAPLE kennt jedoch auch für $n = 5$ interessante Ausdrücke:

```
cos(Pi/5); cos(2*Pi/5);
```

$$\frac{1}{4}\sqrt{5} + \frac{1}{4}, \frac{1}{4}\sqrt{5} - \frac{1}{4}$$

Zur Bestimmung des entsprechenden charakteristischen Polynoms benutzen wir $\cos(5 \cdot \frac{\pi}{5}) = \cos(\pi) = -1$. Wenden wir unsere Mehrfachwinkelformeln auf $\cos(5\,x) + 1$ an, so erhalten wir ein Polynom in $\cos(x)$, das für $x = \frac{\pi}{5}$ verschwindet.

```
p:=expand(cos(5*x)+1);
```

$$16\cos(x)^5 - 20\cos(x)^3 + 5\cos(x) + 1$$

```
p:=subs(cos(x)=z,p);
```

$$16\,z^5 - 20\,z^3 + 5\,z + 1$$

Diese Polynom ist allerdings noch nicht das Minimalpolynom.

```
factor(p);
```

$$(z+1)(-1-2\,z+4\,z^2)^2$$

Dasselbe Programm kann man für $\sin(\frac{\pi}{5})$ absolvieren.

```
sin(Pi/5);
```

$$\frac{1}{4}\sqrt{2}\sqrt{5-\sqrt{5}}$$

Zur Bestimmung der charakteristischen Gleichung gehen wir wie oben vor:

```
p:=expand(sin(5*x));
```

$$16\,\sin(x)\cos(x)^4 - 12\,\sin(x)\cos(x)^2 + \sin(x)$$

```
p:=simplify(p,{cos(x)^2=1-sin(x)^2});
```

$$5\,\sin(x) - 20\,\sin(x)^3 + 16\,\sin(x)^5$$

```
p:=subs(sin(x)=z,p);
```

$$5\,z - 20\,z^3 + 16\,z^5$$

```
factor(p);
```

$$z\left(5 - 20\,z^2 + 16\,z^4\right)$$

Also ist der zweite Faktor $p(z) := 5 - 20\,z^2 + 16\,z^4$ das irreduzible Polynom von $\sin(\frac{\pi}{5})$.

Eine algebraische Zahl ist jedoch durch ihr charakteristisches Polynom nicht eindeutig bestimmt, da ja jedes (irreduzible) Polynom vom Grad n genau n verschiedene komplexe Nullstellen hat. Es ergibt sich also z.B. die Frage, welche weiteren drei Nullstellen $p(z)$ hat und welche interssanten Beziehungen zwischen diesen vier algebraischen Zahlen bestehen. Da man für $x = \frac{k\pi}{5}$, $k \in \mathbf{Z}$ ebenfalls $\sin(5\,x) = 0$ hat, sind die Nullstellen des biquadratischen Polynoms neben $\alpha_{1,2} = \pm\sin(\frac{\pi}{5})$ gerade $\alpha_{3,4} = \pm\sin(\frac{2\pi}{5})$. Interessanterweise ergibt sich für das Produkt der beiden algebraischen Zahlen vom Grad 4

```
p1:=sin(Pi/5)*sin(2*Pi/5);
```

$$\frac{1}{8}\sqrt{5-\sqrt{5}}\sqrt{5+\sqrt{5}}$$

```
combine(p1);
```

$$\frac{1}{4}\sqrt{5}$$

eine algebraische Zahl vom Grad 2.

Untersuchen wir nun den Fall $n = 7$ und betrachten zunächst das Produkt der drei betragsmäsig verschiedenen Werte von $\cos(\frac{k\pi}{7}), k \in \mathbf{Z}$ (alle anderen Werte ergeben sich daraus über Quadrantenbeziehungen):

```
p2:=cos(Pi/7)*cos(2*Pi/7)*cos(3*Pi/7);
```

$$\cos(\frac{\pi}{7})\cos(\frac{4\pi}{7})\cos(\frac{5\pi}{7})$$

Hiermit kann Maple also nichts mehr anfangen. Gleichwohl erhalten wir näherungsweise

```
evalf(p2,20);
```

$$.12499999999999999999$$

und analog für

```
p3:=cos(Pi/7)-cos(2*Pi/7)+cos(3*Pi/7);
```

$$\cos(\frac{3\pi}{7}) - \cos(\frac{2\pi}{7}) + \cos(\frac{\pi}{7})$$

```
evalf(p3,20);
```

$$.50000000000000000000$$

Bestimmen wir nun nach dem bereits bekannten Schema das charakteristische Polynom von $\cos(\frac{\pi}{7})$.

```
factor(subs(cos(x)=z,expand(cos(7*x)+1)));
```

$$(z+1)(1-4z-4z^2+8z^3)^2$$

$\cos(\frac{\pi}{7})$ ist damit als Nullstelle des Polynoms $p(x) = 1-4z-4z^2+8z^3$ eine algebraische Zahl vom Grad 3. Die anderen beiden Nullstellen dieses Polynoms sind $\cos(\frac{3\pi}{7})$ und $\cos(\frac{5\pi}{7}) = -\cos(\frac{2\pi}{7})$, da wir nur $\cos(7x) + 1 = 0$ ausgenutzt haben. Obige Beziehungen ergeben sich nun unmittelbar aus dem bereits aus Kapitel 2 bekannten *Vietaschen Wurzelsatz.*

5.5.2 Der Ring der algebraischen Zahlen

Für die beiden algebraischen Zahlen $\sqrt{2}$ und $\sqrt{3}$ ist es nicht schwer, durch geeignete Umformungen ein Polynom zu finden, das deren Summe $a := \sqrt{2} + \sqrt{3}$ als Nullstelle hat:

$$a^2 = 5 + 2\sqrt{6} \;\Rightarrow\; (a^2-5)^2 = 24 \;\Rightarrow\; a^4 - 10\,a^2 + 1 = 0$$

a ist also Nullstelle des (in diesem Fall bereits irreduziblen) Polynoms $p(x) = x^4 - 10x^2 + 1$. Es stellt sich heraus, dass dies auch allgemein gilt:

Satz 13 *Die Summe und das Produkt zweier algebraischer Zahlen ist wieder eine algebraische Zahl.*

Vor dem allgemeinen Beweis des Satzes wollen wir die Aussage an einem etwas komplizierteren Beispiel studieren, in dem die auszuführenden Umformungen nicht so offensichtlich sind.

Betrachten wir dazu die beiden Zahlen $a = \sqrt{2}$ und $b = \sqrt[3]{5}$ und versuchen, ein Polynom $p := \sum_{i=0}^{n} r_i\, x^i$ zu konstruieren, das $c = a + b$ als Nullstelle hat, d.h. so dass $\sum_{i=0}^{n} r_i\, c^i = 0$ gilt. Um geeignete Koeffizienten r_i zu finden, berechnen wir zunächst die Potenzen $c^i = (a+b)^i$ als Ausdrücke in a und b. Dazu sind die binomischen Formeln sowie die Ersetzungen `{a^2=>2, b^3=>5}` anzuwenden, die sich aus den charakteristischen Polynomen von a bzw. b unmittelbar ergeben. Eine Rechnung mit Reduce ergibt

```
for n:=0:10 collect (a+b)^n where {a^2=>2, b^3=>5};
```

$$
\begin{aligned}
\{1,\ & a+b,\ 2\,a\,b+b^2+2,\ 3\,a\,b^2+2\,a+6\,b+5,\\
& 8\,a\,b+20\,a+12\,b^2+5\,b+4,\\
& 20\,a\,b^2+25\,a\,b+4\,a+5\,b^2+20\,b+100,\\
& 30\,a\,b^2+24\,a\,b+200\,a+60\,b^2+150\,b+33,\\
& 84\,a\,b^2+350\,a\,b+183\,a+210\,b^2+81\,b+700,\\
& 560\,a\,b^2+264\,a\,b+1120\,a+249\,b^2+1400\,b+1416,\\
& 513\,a\,b^2+2520\,a\,b+4216\,a+2520\,b^2+1944\,b+3485,\\
& 5040\,a\,b^2+6160\,a\,b+6050\,a+2970\,b^2+8525\,b+21032\}
\end{aligned}
$$

Wir sehen, dass sich alle Potenzen als Linearkombinationen von sechs Termen 1, a, b, b^2, ab, ab^2 darstellen lassen. Mehr als 6 solcher Ausdrücke sind dann sicher linear abhängig. Finden wir für unser Beispiel eine solche Abhängigkeitsrelation, indem wir eine Linearkombination von 7 verschiedenen Potenzen $(a+b)^i$ mit unbestimmten Koeffizienten aufstellen und diese dann so bestimmen, dass die 6 Koeffizienten vor 1, a, b, b^2, ab, ab^2 alle verschwinden:

```
factor a,b;
u:=for n:=0:6 sum mkid(r,n)*((a+b)^n where {a^2=>2, b^3=>5});
```

$$
\begin{aligned}
& ab^2(3r_3+20r_5+30r_6)+ab(2r_2+8r_4+25r_5+24r_6)+a(r_1+2r_3+20r_4+4r_5+200r_6)+b^2(r_2+\\
& 12r_4+5r_5+60r_6)+b(r_1+6r_3+5r_4+20r_5+150r_6)+(r_0+2r_2+5r_3+4r_4+100r_5+33r_6)
\end{aligned}
$$

```
v:=for each s in coeff(u,a) join coeff(s,b);
```

$$
\begin{aligned}
\{ & r_0+2r_2+5r_3+4r_4+100r_5+33r_6,\ r_1+6r_3+5r_4+20r_5+150r_6,\ r_2+12r_4+5r_5+60r_6,\\
& r_1+2r_3+20r_4+4r_5+200r_6,\ 2r_2+8r_4+25r_5+24r_6,\ 3r_3+20r_5+30r_6\}
\end{aligned}
$$

Das ist ein homogenes lineares Gleichungssystem, deshalb fixieren wir eines der r_i, ehe wir die Lösung bestimmen.

```
s:=solve(sub(r6=1,v));
```

$$\big\{\{r_0=17, r_1=-60, r_4=-6, r_2=12, r_5=0, r_3=-10\}\big\}$$

Ein Polynom mit der Nullstelle $c = a + b$ lautet also

```
f:=sub(first s,r6=1,for i:=0:6 sum mkid(r,i)*x^i);
```

$$x^6 - 6\,x^4 - 10\,x^3 + 12\,x^2 - 60\,x + 17$$

Testen wir schließlich noch, ob dieses Polynom irreduzibel ist:

```
factorize f;
```

Damit wissen wir, dass es sich bei diesem Polynom sogar um das Minimalpolynom von $c = a + b$ handelt.

Beweis : Der Beweis des Satzes geht vollkommen analog. Sind $\alpha_1, \ldots, \alpha_k$ algebraische Zahlen über k vom Grad $d_1, \ldots, d_k$, so ergeben sich aus den entsprechenden Minimalpolynomen

$$p_i(x) = x^{d_i} - q_i(x)$$

Ersetzungsformeln

$$\{\alpha_i^{d_i} \Rightarrow q_i(\alpha_i),\ i = 1, \ldots, k\},$$

die es erlauben, jeden polynomialen Ausdruck aus $R := k[\alpha_1, \ldots, \alpha_k]$ als Linearkombination der $D := d_1 \cdots d_k$ Produkte aus der Menge $T_{red} := \{\alpha_1^{j_1} \cdots \alpha_k^{j_k}\ :\ 0 \le j_i < d_i\}$ zu schreiben.

Ist nun $c \in R$ ein solcher polynomialer Ausdruck (also etwa Summe oder Produkt zweier algebraischer Zahlen), so kann man wie in obigem Beispiel eine nichttriviale lineare Abhängigkeitsrelation zwischen den Potenzen c^i, $i = 0, \ldots, D$ finden und erhält damit ein Polynom $p(x) \in k[x]$, dessen Nullstelle c ist. $\square$

Das gefundene Polynom muss allerdings nicht unbedingt das Minimalpolynom sein, da es in Faktoren zerfallen kann.

Aus dem im Beweis verwendeten konstruktiven Ansatz kann man sogar eine weitergehende Aussage ableiten:

Korollar 2 *Sind $\alpha_1, \ldots, \alpha_k$ algebraische Zahlen über k vom Grad $d_1, \ldots, d_k$, so bildet die Menge der k-linearen Kombinationen von Elementen aus T_{red} einen Ring.*

5.5.3 Die Inverse einer algebraischen Zahl

Von einfachen algebraischen Zahlen wie etwa $1+\sqrt{2}$ oder $\sqrt{2}+\sqrt{3}$ wissen wir, dass man die jeweilige Inverse dazu recht einfach darstellen kann, wenn man mit einer auf geeignete Weise definierten *konjugierten* Zahl erweitert. So gilt etwa

$$\frac{1}{1+\sqrt{2}} = \frac{1-\sqrt{2}}{1-2} = \sqrt{2}-1$$

und

$$\frac{1}{\sqrt{2}+\sqrt{3}} = \frac{\sqrt{3}-\sqrt{2}}{3-2} = \sqrt{3}-\sqrt{2}$$

Damit kann man in diesen Fällen auch die Inverse einer algebraischen Zahl (und damit beliebige Quotienten) als k-lineare Kombination der Produkte aus T_{red} darstellen. Es stellt sich die Frage, ob man auch kompliziertere rationale Ausdrücke mit algebraischen Zahlen auf ähnliche Weise vereinfachen kann. Wie sieht es z.B. mit

$$\frac{1}{\sqrt{2}+\sqrt{3}+\sqrt{5}}$$

aus?

Axiom und Derive liefern als Ergebnis sofort

$$\frac{1}{6}\sqrt{3} + \frac{1}{4}\sqrt{2} - \frac{1}{12}\sqrt{30}$$

Für die anderen Systeme sind dazu spezielle Funktionen, Schalter und/oder Pakete notwendig, so in Maple die Funktion `rationalize` aus der gleichnamigen Bibliothek (die in Version V.3 vorher aber geladen werden muss) und in MuPAD die Funktion `radsimp`. In Reduce (ab Version 3.5[6]) muss der Schalter `rationalize` eingeschaltet sein. In Macsyma benötigt man noch intimere Systemkenntnisse: Es ist der Schalter `algebraic` einzuschalten und die CRE-Form zu verwenden:

[6]In einigen Patch-Varianten der Version 3.6 (vor Februar 1998) wurde dies versehentlich wieder abgestellt.

```
algebraic:true$
a:sqrt(2)+sqrt(3)+sqrt(5)$
1/a;
```

liefert

$$\frac{1}{\sqrt{5}+\sqrt{3}+\sqrt{2}}$$

Erst

```
rat(1/a);
```

ergibt das gewünschte Ergebnis

$$-\frac{\sqrt{2}\,\sqrt{3}\,\sqrt{5}-2\,\sqrt{3}-3\,\sqrt{2}}{12}$$

Mathematica konnte ich nicht dazu veranlassen, eine entsprechende Darstellung zu finden.

Untersuchen wir, auf welchem Wege sich eine solche Darstellung finden ließe. Wie man leicht ermittelt, hat $a = \sqrt{2}+\sqrt{3}+\sqrt{5}$ das charakteristische Polynom $p(x) = x^8 - 40x^6 + 352x^4 - 960x^2 + 576$, d.h. es gilt

$$a^8 - 40a^6 + 352a^4 - 960a^2 + 576 = 0$$

oder

$$a^{-1} = \frac{-a^7 + 40a^5 - 352a^3 + 960a}{576}$$

Kennt das System das charakteristische Polynom, ist es also nicht schwer, a^{-1} zu berechnen. Es werden einzig noch Ringoperationen benötigt, um den Ausdruck zu vereinfachen:

$$a^{-1} = \mathtt{sub}\left(a = \sqrt{2}+\sqrt{3}+\sqrt{5}, \frac{-a^7 + 40a^5 - 352a^3 + 960a}{576}\right)$$

Es erhebt sich die Frage, ob die "schlauen" Systeme dasselbe Programm auch mit komplizierteren algebraischen Zahlen abspulen können. Betrachten wir dazu den Ausdruck

$$A := \frac{\sqrt[3]{4}-2\sqrt[3]{2}+1}{\sqrt[3]{4}+2\sqrt[3]{2}+1} = \frac{a^2-2a+1}{a^2+2a+1}$$

mit $a = \sqrt[3]{2}$, d.h. $a^3 = 2$.

Derive ist dazu nicht mehr in der Lage. Maple und MuPAD liefern mit obiger Bibliothek, ebenso wie Reduce und Macsyma

```
s:=subs(a=2^(1/3),(a^2-2*a+1)/(a^2+2*a+1)):
s1:=rationalize(s);
```

$$1/3\,\left(2^{2/3}-2\,\sqrt[3]{2}+1\right)\left(2^{2/3}-1\right)$$

```
expand(s1);
```

$$\frac{4\,\sqrt[3]{2}}{3} - 5/3$$

Offensichtlich ist $(2^{2/3}-1)$ der "richtige" Term, mit dem man A erweitern muss, um den Nenner in einen rationalen Ausdruck zu verwandeln. Wie kann man einen solchen Term finden? Offensichtlich ist obiger Zugang dafür weniger geeignet, denn wir würden mit jedem Nenner, dessen Minimalpolynom wir bestimmen würden, eine neue algebraische Zahl einführen, deren Beziehung zu den bisher schon verwendeten algebraischen Zahlen zu klären wäre. Wir hatten in verschiedenen Beispielen gesehen, dass das schwierig ist.

Wir wollen statt dessen wieder einen Zugang über unbestimmte Koeffizienten wählen, den wir im Nachgang dann auch theoretisch begründen. Die notwendigen Rechnungen wollen wir mit REDUCE an unserem Beispiel nachvollziehen. Wir wollen untersuchen, ob bzw. wie sich der rationale Ausdruck A als Linearkombination von $a^i,\ i = 0, 1, 2$ mit $a = \sqrt[3]{2}$ darstellen lässt. Wir beginnen mit dem Ansatz

$$A = \frac{a^2 - 2a + 1}{a^2 + 2a + 1} = a^2 r_2 + a r_1 + r_0$$

mit unbestimmten Koeffizienten r_2, r_1, r_0. Ausmultiplizieren und Koeffizientenvergleich liefern

```
u:=(a^2*r2+a*r1+r0);
poly:=((a^2-2*a+1)-(a^2+2*a+1)*u where {a^3=>2});
gls:=coeff(poly,a);
sol:=solve(gls,{r0,r1,r2});
```

$$\{\{r_2 = 0, r_1 = \frac{4}{3}, r_0 = \frac{-5}{3}\}\}$$

und damit als Ergebnis unserer Vereinfachung

```
sub(first sol, u);
```

$$\frac{4a - 5}{3}$$

Allgemein gilt der folgende Satz

Satz 14 *Ist α eine algebraische Zahl über k vom Grad d, so bildet die Menge $R := k[\alpha]$ der k-linearen Kombinationen von Termen aus $T_{red} := \{\alpha^i,\ i = 0, \ldots, d-1\}$ einen Körper.*

Kann man in k effektiv rechnen, so auch in R. Genauer: Existiert in k eine kanonische oder Normalform und ist das Minimalpolynom von α bekannt, so existiert auch in R eine kanonische oder Normalform.

BEWEIS : Sei $p(x) = x^d - q(x),\ deg\ q < d$ das Minimalpolynom von α.

Wir müssen zum Beweis des Satzes nur zeigen, dass unser Verfahren zur Berechnung der Inversen eines Ausdrucks $A(\alpha) \in R$, das wir oben an einem Beispiel demonstriert haben, stets zum Ziel führt. Es ist dazu das lineare Gleichungssystem in $r_0, \ldots, r_{d-1}$, das man aus

$$A(\alpha) \cdot \sum_{i=0}^{d-1} r_i \alpha^i = 1$$

nach (algebraischer) Ersetzung $\{\alpha^d \ \Rightarrow\ q(\alpha)\}$ durch Koeffizientenvergleich erhält, zu lösen.

Bei diesem Gleichungssystem handelt es sich um ein inhomogenes System von d Gleichungen mit d Unbekannten, dessen zugehöriges homogenes System nur die triviale Lösung besitzt, da jede nichttriviale Lösung des Systems

$$A(\alpha) \cdot \sum_{i=0}^{d-1} r_i \alpha^i = 0$$

zu einem Polynom vom Grad $< d$ mit Nullstelle α führt.

Folglich hat das inhomogene System für jede rechte Seite genau eine Lösung, d.h. die Koeffizienten $r_0, \ldots, r_{d-1}$ lassen sich eindeutig bestimmen. $\square$

5.6 Algebraische Erweiterungstürme

Der letzte Satz zeigt, dass man auch mit mehreren algebraischen Zahlen $\alpha_1, \alpha_2, \ldots, \alpha_n$ effektiv rechnen kann, wenn man Schritt für Schritt auf die oben angegebene Art die Erweiterungen $k_i = k_{i-1}[\alpha_i]$ mit $k_0 = k$ konstruiert. Eine solche Folge aufeinander aufbauender algebraischer Körpererweiterungen bezeichnet man als *algebraischen Körperturm*. Zu seiner Konstruktion muss man einzig in der Lage sein, das Faktorisierungsproblem in $k_i[x]$ über den jeweils entstehenden Zwischenkörpern k_i zu lösen.

Dies wird insbesondere dann wichtig, wenn Rechnungen ausgeführt werden sollen, die mehrere algebraische Zahlen mit demselben Minimalpolynom enthalten. Sind nämlich α_1 und α_2 algebraische Zahlen (etwa über $k = \mathbf{Q}$) mit dem gemeinsamen Minimalpolynom $p(x)$, so ist α_2 zwar auch Nullstelle von $p(x)$ über $k_1 = k[\alpha_1]$, aber es gilt

$$p(x) = (x - \alpha_1)\, q(x)$$

für ein geeignetes Polynom $q(x) \in k_1[x]$. α_2 ist dann Nullstelle dieses zweiten Faktors, welcher aber noch immer nicht irreduzibel über k_1 sein muss.

Über entsprechende Algorithmen zur Faktorisierung über algebraischen Erweiterungen verfügen in der einen oder anderen Form außer Derive alle Systeme. Allerdings unterscheidet sich die Art, wie solche algebraischen Zahlen in die Rechnungen eingeführt werden, grundlegend.

MuPAD verfolgt zum Beispiel einen objektorientierten Ansatz, mit dem sich ein algebraischer Erweiterungskörper als Grundbereich konstruieren und an eine Variable binden lässt. Diese kann man dann als Parameter an entsprechende Faktorisierungsroutinen übergeben.

```
Q1:=Dom::AlgebraicExtension(Dom::Rational,a1^5-a1+1);
```

$$\text{Dom::AlgebraicExtension(Dom::Rational}, -a_1 + a_1^5 + 1 = 0, a_1)$$

Q_1 ist hier der algebraische Erweiterungskörper $Q_1 = \mathbf{Q}[a_1]$, dessen erzeugendes Element a_1 das Minimalpolynom $p := x^5 - x + 1$ besitzt. Faktorisieren wir dieses Polynom über Q_1, so zerfällt es in der Tat in zwei Faktoren:

```
Factor(poly(x^5-x+1,Q1));
```

$$(x - a_1)\left(x^4 + a_1^4 + x a_1^3 + x^3 a_1 + x^2 a_1^2 - 1\right)$$

Eine zweite Nullstelle a_2 von p hat also über Q_1 den zweiten Faktor p_1 als Minimalpolynom, so dass wir die entsprechende Erweiterung

```
Q2:=Dom::AlgebraicExtension(Q1,subs(p1,x=a2)):
```

als Körperturm erhalten, über dem wir wiederum versuchen können, die Faktorzerlegung unseres Ausgangspolynom $p(x)$ zu finden, die diesmal die beiden Linearfaktoren $(x - a_1)$ und $(x - a_2)$ enthalten sollte.

```
factor(poly(x^5-x+1,Q2));
```

Allerdings hatte ich nicht die Geduld, auf das Resultat zu warten.

Maple gestattet die Definition algebraischer Zahlen über einen alias-Mechanismus. Auf diese Weise wird mit einem entsprechenden RootOf-Ausdruck ein neues Symbol verbunden, das wie in MuPAD zugleich als Printname[7] dient, wodurch die entstehenden Formeln kompakt notiert werden können. Für obiges Beispiel erhält man

[7] Leider nicht bei der LaTeX-Ausgabe.

```
alias(a1=RootOf(x^5-x+1)):
s:=factor(x^5-x+1,a1);
```

$$(x^4 + a_1\,x^3 + a_1^2\,x^2 + a_1^3\,x - 1 + a_1^4)\,(x - a_1)$$

```
p1:=op(1,s):
alias(a2=RootOf(p1)):
s1:=factor(x^5-x+1,{a1,a2});
```

Nach bereits 36 s. (auf einer Sun UltraSparc mit MAPLE V.4) erhält man

$$(a_2\,x^2 + a_1^2\,x + a_1\,x^2 + x^3 + a_1^3 + a_1\,a_2\,x + a_2^3 + a_2^2\,x + a_1^2\,a_2 + a_1\,a_2^2)\,(x - a_2)\,(x - a_1)$$

das mit dem speziellen Befehl zur Simplifikation von Ausdrücken mit algebraischen Zahlen

```
evala(s1);
```

wieder zu

$$x^5 - x + 1$$

vereinfacht.

MATHEMATICA verfolgt einen ähnlichen Zugang, wobei allerdings für die langen ROOTOF-Ausdrücke nicht automatisch kurze Printsymbole eingeführt werden. Definieren wir wie oben a_1 als algebraische Zahl mit dem Minimalpolynom x^5-x+1 und geben sie beim Faktorisieren explizit an, so erhalten wir dieselbe Faktorzerlegung wie mit MAPLE

```
a1=Root[x^5-x+1,1];
u=Factor[x^5-x+1,Extension->{a1}]
```

$$(x - a_1)\left(-1 + x^4 + x^3\,a_1 + x^2\,a_1^2 + x\,a_1^3 + a_1^4\right)$$

Die zweite Nullstelle a_2 kann man nicht als Körperturm wie in MAPLE definieren:

```
a2=Root[u[[2]],1]
```

führt nämlich auf

$$\mathrm{Root}[1 - \#1 + \#1^5\&, 2],$$

also zu keinem anderen Ergebnis als wenn man die zweite Nullstelle a_2 des Polynoms p direkt hinzugefügt hätte. Faktorisiert man erneut

```
a2=Root[x^5-x+1,2];
s=Factor[x^5-x+1,Extension->{a1,a2}]
```

so erhält man nach geduldigem Warten als Ergebnis einen riesigen Ausdruck mit sehr langen Koeffizienten, der zwei lineare und einen kubischen Faktor enthält. Die beiden linearen Faktoren haben allerdings nicht die (syntaktische) Gestalt $x - a_1$ bzw. $x - a_2$, sondern enthalten als Absolutglied jeweils kompliziertere polynomiale Audrücke, in denen a_1 und a_2 jeweils bis zum Grad 4 vertreten sind. Das ist nicht verwunderlich, da die im Abschnitt 5.5.2 angegebene Menge T_{red} für mehr als eine algebraische Zahl nicht mehr linear unabhängig sein muss, d.h. nicht unbedingt eine kanonische Darstellung liefert. In der Tat hat a_2 über $\mathbf{Q}[a_1]$ ein Minimalpolynom vom Grad 4, d.h. auch Potenzen a_2^4 könnten weiter normiert werden.

Aus demselben Grund führt Ausklammern und Simplifizieren mit `Simplify@Expand[s]`[8] nicht zum Ausgangspolynom zurück.

Einzig AXIOM ist in der Lage, in diesem Beispiel schnell und zuverlässig die Minimalpolynome der einzelnen Nullstellen $a_i,\ i = 1, \ldots, 5,$ des Polynoms $x^5 - x + 1$ jeweils über $k_0 = \mathbf{Q}, k_1 = k_0[a_1]$ etc. auszurechnen, d.h. das sogenannte *Zerfällungskörper-Problem* zu lösen:

[8]Nach Information von M. Trott (Wolfram Research) ist wegen der internen Gestaltung von `Simplify` der Befehl `RootReduce@Expand[s]` angemessener.

```
l:=rootsOf(p,x)
```

```
       [%x8,%x9,%x10,%x11,- %x11 - %x10 - %x9 - %x8]

                                    Type: List Expression Integer
```

Wir erhalten eine Liste der 5 Nullstellen, für die AXIOM selbsttätig neue Symbole eingeführt hat. Auch die zugehörigen Minimalpolynome im Sinne eines Körperturms sind bekannt:

```
definingPolynomial %x8
```

$$x_8^5 - x_8 + 1$$

```
definingPolynomial %x9
```

$$x_8^4 + x_9 x_8^3 + x_9^2 x_8^2 + x_9^3 x_8 + x_9^4 - 1$$

```
definingPolynomial %x10
```

$$x_9^3 + (x_8 + x_{10}) x_9^2 + \left(x_8^2 + x_{10}\, x_8 + x_{10}^2\right) x_9 + x_8^3 + x_{10}\ x_8^2 + x_{10}^2\ x_8 + x_{10}^3$$

```
definingPolynomial %x11
```

$$x_{10}^2 + (x_9 + x_8 + x_{11})\, x_{10} + x_9^2 + (x_8 + x_{11})\, x_9 + x_8^2 + x_{11}\, x_8 + x_{11}^2$$

Die fünfte Nullstelle kann schließlich durch die anderen rational ausgedrückt werden, so dass keine weitere Körpererweiterung notwendig ist.

Will man dagegen die Faktorisierung von Ausdrücken, die algebraische Zahlen enthalten, wie in den anderen Systemen explizit ausführen, so gerät man schnell ins Dickicht der verschiedenen Datentypen, die wegen des verwendeten strengen Typkonzepts alle AXIOM-Rechnungen notwendiger Weise begleiten. So ergibt etwa

```
factor(p,%x8)
```

einen Typfehler, da die vom System selbst erzeugte algebraische Zahl $\%x8$ vom Typ `Expression Integer` ist, während beim Faktorisieren als Zusatzparameter eine Variable vom Typ `Algebraic Number` benötigt wird. Um die sich aufbauenden Hürden zu umschiffen, werden an einigen Stellen subtile explizite Typkonvertierungen notwendig, auf die wir hier nicht näher eingehen wollen.

MACSYMA vermag Polynome nur über einer einfachen Erweiterung der ganzen Zahlen zu faktorisieren. Dazu kann man als zweiten Parameter der Funktion `factor` das entsprechende Minimalpolynom angeben.

```
factor(x^5-x+1,a1^5-a1+1);
```

$$(x - a_1)\ \left(x^4 + a_1\, x^3 + a_1^2\, x^2 + a_1^3\, x + a_1^4 - 1\right)$$

REDUCE verfügt standardmäßig nur über rudimentäre Fähigkeiten, kompliziertere algebraische Zahlen einzuführen und mit ihnen zu rechnen. Definiert man etwa eine solche Zahl a über einen entsprechenden ROOTOF-Ausdruck `a:=root_of(x^d-q(x),x)`, so wird in nachfolgenden Rechnungen die algebraische Ersetzung $a^d \Rightarrow q(a)$ ausgeführt, aber bereits die oben beschriebene Inversenberechnung erfolgt nicht (selbst unter `on rationalize;`). Um solche Rechnungen doch noch auszuführen, steht das REDUCE-Paket `arnum` zur Verfügung. Allerdings ist dieses Paket nur ungenügend integriert und führt schnell zu relativ unverständlichen Fehlermeldungen. Außerdem ist es in seiner Leistungsfähigkeit nur streckenweise mit den bisher besprochenen Systemen vergleichbar.

5.7 Algebraische Zahlen mit gemeinsamem Minimalpolynom und symmetrische Funktionen

In diesem Abschnitt wollen wir die Beziehungen zwischen den verschiedenen Nullstellen eines (irreduziblen) Polynoms genauer studieren.

Wir hatten weiter oben gesehen, dass es durchaus interessante solche Beziehungen geben kann, deren Kenntnis für das qualifizierte Rechnen mit algebraischen Zahlen wichtig ist. So hatten wir rationale Ausdrücke für die Potenzsummen der Nullstellen eines Polynoms dritten Grades gefunden oder Beziehungen zwischen verschiedenen algebraischen Zahlen trigonometrischer Natur entdeckt.

Derartige Beziehungen bestehen auch zwischen algebraischen Zahlen höheren als dritten Grades und sind einigen der von uns untersuchten CAS wohlbekannt. Betrachten wir dazu zwei Beispiele in MAPLE. Berechnen wir zuerst die Potenzsummen über die fünf Wurzeln des Polynoms $p = x^5-x+1$ für $k = 1, \ldots, 9$:

```
u:=solve(x^5-x+1,x);
```

$$\mathit{RootOf}(Z^5 - Z + 1)$$

```
[seq(sum(v^k,'v'=u),k=1..9)];
```

$$[0, 0, 0, 4, -5, 0, 0, 4, -9]$$

Bis auf die aus mathematischer Sicht etwas komische Schreibweise ist das Ergebnis klarer ausgefallen als für ein Polynom dritten Grades, wo explizit mit Wurzelausdrücken gerechnet wurde. Allerdings konnte die Funktion `sum` hier das ROOTOF-Symbol explizit entdecken und so einen entsprechenden Algorithmus aufrufen.

Noch interessanter fällt das Ergebnis der folgenden Vereinfachung aus, wenn wir die Summe $\sum \frac{1}{x-\alpha}$ über die fünf Wurzeln obigen Polynoms bilden:

```
R:=sum(1/(x-v),'v'=u);
```

$$R = \frac{5\,x^4 - 1}{1 - x + x^5}$$

In diese Summe rationaler Ausdrücke gehen die verschiedenen Nullstellen in den Summanden auf gleiche Weise ein und es ergibt sich ein Ausdruck ganz ohne algebraische Zahlen!

5.7.1 Symmetrische Polynome

Im folgenden wollen wir eine Begründung für diese auf den ersten Blick vielleicht verblüffenden Beziehungen zwischen algebraischen Zahlen sehr komplizierter Bauart geben. Wir sehen daran noch einmal, dass die ROOTOF-Darstellung gegenüber einer solchen durch Radikale den großen Vorteil bietet, dass sich aus ihr wichtige Eigenschaften der eingeführten Symbole auf relativ einfache Weise extrahieren lassen.

All diesen Ausdrücken ist gemeinsam, dass die verschiedenen Nullstellen $Z = (z_1, \ldots, z_n)$, die sich hinter einem gemeinsamen ROOTOF-Symbol vom Grad n mit dem Minimalpolynom

$$p(x) = x^n + a_1 x^{n-1} + \ldots + a_{n-1} x + a_n$$

verbergen, in diese auf symmetrische Weise eingehen. Damit ergeben sich beim Ausmultiplizieren und Bilden eines Hauptnenners in einem Ausdruck wir R symmetrische Polynome in diesen Nullstellen als Koeffizienten. Nach dem *Vietaschen Wurzelsatz* (Kapitel 2) sind die elementarsymmetrischen Polynome $e_k = e_k(X)$ eng mit den Koeffizienten von $p(x)$ verbunden: Es gilt

$$a_i = (-1)^i\, e_i(Z), \quad i = 1, \ldots, n$$

Wir wollen deshalb zunächst untersuchen, wie *beliebige* symmetrische Polynome mit den elementarsymmetrischen Polynomen zusammenhängen.

Sei dazu $h = h(X)$ ein symmetrisches Polynom in $X = (x_1, \ldots, x_n)$ mit Koeffizienten aus dem Körper k, das in distributiver Form mit lexikographisch angeordneten Termen dargestellt sei. Wegen der Symmetrie hat der erste Summand von h in einer solchen Darstellung die Gestalt

$$c\, x_1^{a_1} x_2^{a_2} \cdots x_n^{a_n}$$

mit $c \in k$ und $a_1 \geq a_2 \geq \ldots \geq a_n$. Da das ebenfalls symmetrische Polynom

$$c\, e_1^{a_1 - a_2} e_2^{a_2 - a_3} \cdots e_{n-1}^{a_{n-1} - a_n} e_n^{a_n}$$

denselben höchsten Term hat (wegen $e_i = x_1 \cdot \ldots \cdot x_i + \langle\text{kleinere Terme}\rangle$), so ergibt sich als Differenz beider Polynome ein neues symmetrisches Polynom, dessen höchster Term lexikographisch kleiner ist. Für die Summe der Quadrate der 5 Nullstellen von p

$$p_1 := \sum_{i=1}^{5} x_i^2 = x_1^2 + x_2^2 + x_3^2 + x_4^2 + x_5^2$$

gilt etwa $a_1 = 2$, $a_2 = \ldots = a_5 = 0$ und $e_1^2 = x_1^2 + \langle\text{kleinere Terme}\rangle$. Als Differenz der beiden symmetrischen Polynome erhalten wir das symmetrische Polynom

$$p_1 - e_1^2 = -2(x_1x_2 + x_1x_3 + x_1x_4 + x_1x_5 + x_2x_3 + x_2x_4 + x_2x_5 + x_3x_4 + x_3x_5 + x_4x_5),$$

dessen Leitterm x_1x_2 lexikographisch kleiner ist als x_1^2. Da e_2 denselben Leitterm x_1x_2 hat, erhalten wir schließlich

$$p_1 - e_1^2 + 2e_2 = 0,$$

also $p_1 = e_1^2 - 2e_2$.

Auf diese Weise kann man von jedem symmetrischen Polynom Schritt für Schritt Produkte elementarsymmetrischer Polynome abziehen, so dass sich der Leitterm jeweils verkleinert, bis sich das Nullpolynom ergibt. Für die Summe der Kuben der Nullstellen von p erhalten wir etwa

$$\begin{aligned}
&p_2 := x_1^3 + x_2^3 + x_3^3 + x_4^3 + x_5^3 \\
&p_2 - e_1^3 = \\
&\qquad 3(-x_1^2x_2 - x_1^2x_3 - x_1^2x_4 - x_1^2x_5 - x_1x_2^2 - 2x_1x_2x_3 - 2x_1x_2x_4 - 2x_1x_2x_5 - x_1x_3^2 - \\
&\qquad 2x_1x_3x_4 - 2x_1x_3x_5 - x_1x_4^2 - 2x_1x_4x_5 - x_1x_5^2 - x_2^2x_3 - x_2^2x_4 - x_2^2x_5 - x_2x_3^2 - \\
&\qquad 2x_2x_3x_4 - 2x_2x_3x_5 - x_2x_4^2 - 2x_2x_4x_5 - x_2x_5^2 - x_3^2x_4 - x_3^2x_5 - x_3x_4^2 - 2x_3x_4x_5 - \\
&\qquad x_3x_5^2 - x_4^2x_5 - x_4x_5^2) \\
&p_2 - e_1^3 + 3e_1e_2 = \\
&\qquad 3(x_1x_2x_3 + x_1x_2x_4 + x_1x_2x_5 + x_1x_3x_4 + x_1x_3x_5 + x_1x_4x_5 + x_2x_3x_4 + x_2x_3x_5 + \\
&\qquad x_2x_4x_5 + x_3x_4x_5) \\
&p_2 - e_1^3 + 3e_1e_2 - 3e_3 = 0,
\end{aligned}$$

also $p_2 = e_1^3 - 3e_1e_2 + 3e_3$.

Dieses Verfahren terminiert in jedem Fall, weil die lexikographische Ordnung noethersch ist. Damit haben wir den folgenden Satz bewiesen:

Satz 15 (Hauptsatz über symmetrische Polynome)
Jedes symmetrische Polynom $P(\mathbf{x}) \in k[x_1, \ldots, x_n]$ lässt sich (eindeutig) als polynomialer Ausdruck $Q(e_1, \ldots, e_n)$ in den elementarsymmetrischen Polynomen darstellen.

Als Schlußfolgerung dieses Satzes ergibt sich die folgende Aussage über polynomiale Ausdrücke in den Nullstellen eines gegebenen Polynoms.

Korollar 3 *Sind $z_1, \ldots, z_n$ die Nullstellen eines (nicht unbedingt irreduziblen) Polynoms $p(x) \in \mathbf{Q}[x]$ mit rationalen Koeffizienten, so lässt sich jeder symmetrische polynomiale Ausdruck in diesen Variablen $z_1, \ldots, z_n$ zu einer rationalen Zahl vereinfachen.*

Dies gilt insbesondere für die Potenzsummen und allgemeiner für Ausdrücke der Form

$$\sum_{a \in \mathrm{RootOf}(p(x))} f(a),$$

wenn $f(x)$ ein polynomialer Ausdruck ist.

Ausdrücke dieser Art, in denen die verschiedenen Nullstellen *separiert* in die einzelnen Summanden eingehen, lassen sich mit der von MAPLE angebotenen Kombination der Funktionssymbole `sum` und ROOTOF vereinfachen.

Allerdings gibt es auch Fragestellungen, in denen die Nullstellen auf kompliziertere Weise miteinander verbunden sind. Seien z.B. z_1, z_2, z_3, z_4 die vier Nullstellen des Polynoms $x^4 + x + 1$. Wir wollen ein Polynom $p(x)$ mit den sechs Nullstellen $\{z_i\, z_j \,:\, 1 \le i < j \le 4\}$ bestimmen. Dieses Polynom sechsten Grades ergibt sich als

$$p(x) = \prod_{1 \le i < j \le 4} (y \,-\, z_i\, z_j).$$

Expandieren wir die Faktoren und sammeln die Koeffizienten vor den entsprechenden Potenzen y^i zusammen, so zeigt sich, dass diese symmetrisch in $z_1, \ldots, z_4$ sind:

$$\begin{aligned}
p(x) = {} & \\
& y^6 - y^5(z_1 z_2 + z_1 z_3 + z_1 z_4 + z_2 z_3 + z_2 z_4 + z_3 z_4) \\
& + y^4(z_1^2 z_2 z_3 + z_1^2 z_2 z_4 + z_1^2 z_3 z_4 + z_1 z_2^2 z_3 + z_1 z_2^2 z_4 + z_1 z_2 z_3^2 + 3 z_1 z_2 z_3 z_4 \\
& \qquad + z_1 z_2 z_4^2 + z_1 z_3^2 z_4 + z_1 z_3 z_4^2 + z_2^2 z_3 z_4 + z_2 z_3^2 z_4 + z_2 z_3 z_4^2) \\
& + y^3(-z_1^3 z_2 z_3 z_4 - z_1^2 z_2^2 z_3^2 - 2 z_1^2 z_2^2 z_3 z_4 - z_1^2 z_2^2 z_4^2 - 2 z_1^2 z_2 z_3^2 z_4 - 2 z_1^2 z_2 z_3 z_4^2 - z_1^2 z_3^2 z_4^2 \\
& \qquad - z_1 z_2^3 z_3 z_4 - 2 z_1 z_2^2 z_3^2 z_4 - 2 z_1 z_2^2 z_3 z_4^2 - z_1 z_2 z_3^3 z_4 - 2 z_1 z_2 z_3^2 z_4^2 - z_1 z_2 z_3 z_4^3 - z_2^2 z_3^2 z_4^2) \\
& + y^2 z_1 z_2 z_3 z_4 (z_1^2 z_2 z_3 + z_1^2 z_2 z_4 + z_1^2 z_3 z_4 + z_1 z_2^2 z_3 + z_1 z_2^2 z_4 + z_1 z_2 z_3^2 + 3 z_1 z_2 z_3 z_4 + z_1 z_2 z_4^2 \\
& \qquad + z_1 z_3^2 z_4 + z_1 z_3 z_4^2 + z_2^2 z_3 z_4 + z_2 z_3^2 z_4 + z_2 z_3 z_4^2) \\
& - y z_1^2 z_2^2 z_3^2 z_4^2 (z_1 z_2 + z_1 z_3 + z_1 z_4 + z_2 x_3 + z_2 z_4 + z_3 z_4) + z_1^3 z_2^3 z_3^3 z_4^3 \\
= {} & y^6 - y^5 e_2 + y^4 h_1 - y^3 h_2 + y^2 e_4 h_1 - y e_2 e_4^2 + e_4^3,
\end{aligned}$$

wobei h_1 und h_2 für die entsprechenden symmetrischen Klammerausdrücke stehen. Mit unserer Expansionstechnik erhält man

$$h_1 = e_1 e_3 - e_4,$$

$$h_2 = e_1^2 e_4 + e_3^2 - 2 e_2 e_4$$

und damit

$$p(x) = y^6 - y^5 e_2 + y^4(e_1 e_3 - e_4) - y^3(e_1^2 e_4 + e_3^2 - 2 e_2 e_4) + y^2 e_4 (e_1 e_3 - e_4) - y e_2 e_4^2 + e_4^3.$$

Nach dem Vietaschen Wurzelsatz folgt $e_1 = e_2 = 0, e_3 = -1, e_4 = 1$ für das hier untersuchte Polynom $x^4 + x + 1$, also insgesamt

$$p(x) = y^6 - y^4 - y^3 - y^2 + 1.$$

5.7.2 Symmetrische Summen rationaler Ausdrücke

Wir hatten im einführenden Beispiel dieses Abschnitts gesehen, dass es Fälle gibt, in denen sich auch eine Summe von *rationalen* Ausdrücken, in die die verschiedenen Nullstellen ein und desselben Polynoms auf symmetrische Weise eingehen, auf rationale Weise durch die Koeffizienten der beteiligten Polynome ausdrücken lassen kann.

Wir wollen zum Abschluss die Summation allgemeiner Ausdrücke der Gestalt

$$\sum_{g(\alpha)=0} \frac{c(\alpha)}{x-\alpha}$$

mit Polynomen $c(x)$, $g(x) \in k[x]$ betrachten (in obigem Beispiel ist $c(x) = 1$).

Satz 16 *Sei $g(x) \in k[x]$ ein Polynom mit einfachen Nullstellen. Dann lässt sich jede Summe*

$$S = \sum_{g(\alpha)=0} \frac{c(\alpha)}{x-\alpha}$$

in der Form $\frac{f(x)}{g(x)}$ mit einer polynomialen Funktion $f(x) \in k[x]$ vom Grad $deg\ f < deg\ g$ darstellen.

Beweis : O.B.d.A. sei der Leitkoeffizient von g gleich 1. Da g nur einfache Nullstellen hat, gilt $g(x) = \prod_b (x-b)$, wobei sich das Produkt über alle Nullstellen von g erstreckt. Multiplizieren wir S mit $g(x)$, so kürzen sich in jedem Summanden die Nenner weg und wir erhalten

$$S \cdot g(x) = \sum c(a) \frac{g(x)}{x-a} = \sum c(a) \prod_{b \neq a} (x-b).$$

Die rechte Seite dieser Gleichung ist ein Polynom $f(x)$ vom Grad $< deg\ g$, für das zu zeigen bleibt, dass seine Koeffizienten bereits in k liegen.

$$f(x) = \sum c(a) \prod_{b \neq a} (x-b)$$

erinnert in seinem Aussehen stark an die Lagrange-Interpolationsformel, mit der wir es jetzt in Beziehung setzen wollen:

$$f(x) = \sum c(a) \cdot \prod_{b \neq a} (a-b) \cdot \prod_{b \neq a} \frac{x-b}{a-b} = \sum c(a) \cdot g'(a) \cdot \prod_{b \neq a} \frac{x-b}{a-b}.$$

wobei wir die Beziehung $g'(a) = \prod_{b \neq a} (a-b)$ verwendet haben, die sich aus $g(x) = \prod_b (x-b)$ und der Produktformel ergibt. $f(x)$ ist also das Interpolationspolynom, das an den Stellen $a \in$ RootOf$(g(x))$ die Werte $f(a) = c(a)\, g'(a)$ annimmt. Da einerseits $c(x)\, g'(x)\ (mod\ g(x))$ ein solches Polynom ist und andererseits das Interpolationspolynom $f(x)$ mit $deg\ f < deg\ g$ eindeutig bestimmt ist, schließen wir $f(x) = c(x)\, g'(x)\ (mod\ g(x)) \in k[x]$. □

Ähnlich kann man auch mit beliebigen Summen rationaler Ausdrücke verfahren, in die die verschiedenen Nullstellen eines (quadratfreien) Polynoms symmetrisch eingehen. Ist etwa

```
a:=RootOf(x^5-x+1,x);
```

$$RootOf(Z^5 - Z + 1)$$

die Menge der Nullstellen des Polynoms p fünften Grades, das wir eingangs dieses Abschnitts betrachtet haben, so gilt

```
sum(1/(x-u^2),u=a);
```

$$\frac{5\,x^4 - 6\,x^2 + 1}{x - 2\,x^3 + x^5 - 1}$$

```
sum(1/(x^2-u),u=a);
```

$$\frac{5\,x^8-1}{1-x^2+x^{10}}$$

```
sum((x+u)/(x-u^2),u=a);
```

$$\frac{5\,x^5-6\,x^3+x+1-5\,x^2}{x-2\,x^3+x^5-1}$$

Allgemein gilt der folgende Satz:

Satz 17 *Ist $f(x,y) \in k(x,y)$ eine rationale Funktion und $g(x) \in k[x]$ ein Polynom mit einfachen Nullstellen, so lässt sich*

$$S(x) = \sum_{g(\alpha)=0} f(x,\alpha)$$

als rationale Funktion $S(x) \in k(x)$ mit Koeffizienten aus dem Grundkörper k darstellen.

Kapitel 6

Die Stammfunktion einer rationalen Funktion

Aus dem Grundkurs Analysis ist bekannt, dass jede rationale Funktion $r(x) = \frac{f(x)}{g(x)}$ mit teilerfremden f, g eine Stammfunktion besitzt, die sich durch elementare Funktionen (genauer: rationale Funktionen und Logarithmen) ausdrücken lässt. Dort wird gewöhnlich auch ein Verfahren zur Bestimmung dieser Stammfunktion erläutert, das auf NEWTON und LEIBNIZ zurückgeht und erstmals von J.BERNOULLI (1703) vollständig begründet wurde. Es geht von der Zerlegung des Nenners $g(x)$ in komplexe Linearfaktoren bzw. reelle lineare und quadratische Faktoren aus und verwendet eine *Partialbruchzerlegung*, deren Summanden bekannte Stammfunktionen besitzen. Für den komplexen Fall, auf den wir uns im Weiteren der Einfachheit halber beschränken wollen, gilt dann folgender Satz

Satz 18 (Newton-Leibniz-Bernoulli) *Seien $\alpha_1, \ldots, \alpha_d \in \mathbf{C}$ die verschiedenen Nullstellen von $g(x)$ mit den Vielfachheiten $\nu_1, \ldots, \nu_d$. Dann existieren Polynome $p(x), q_1(x), \ldots, q_d(x) \in \mathbf{C}[x]$ und Konstanten $c_1, \ldots, c_d \in \mathbf{C}$ mit*

$$\int r(x)\, dx = p(x) + \sum_i \frac{q_i(x)}{(x-\alpha_i)^{\nu_i - 1}} + \sum_i c_i \log(x - \alpha_i)$$

und $deg(p) \leq deg(r) := deg(f) - deg(g)$ sowie $deg(q_i) < \nu_i - 1$.

Die Stammfunktion besteht also aus einem *rationalen Anteil*

$$r_1(x) = p(x) + \sum_i \frac{q_i(x)}{(x-\alpha_i)^{\nu_i - 1}}$$

und einem *transzendenten Anteil*

$$r_2(x) = \sum_i c_i \log(x - \alpha_i).$$

Um eine solche Stammfunktion bei vorgegebener rationaler Funktion $r(x)$ zu konstruieren, könnte man die einzelnen Parameter p, q_i, c_i in obiger Formel mit unbestimmten Koeffizienten ansetzen, die abgeleitete Gleichung

$$f(x) = \left(p'(x) + \left(\sum_i \frac{q_i(x)}{(x-\alpha_i)^{\nu_i - 1}} \right)' + \sum_i \frac{c_i}{(x-\alpha_i)} \right) g(x)$$

aufstellen, in der der Ausdruck auf der rechten Seite ebenfalls polynomial ist, und durch Koeffizientenvergleich ein lineares Gleichungssystem gewinnen, dem die unbestimmten Koeffizienten genügen müssen.

Unter der Annahme, dass $deg(f) < deg(g) =: n$ gilt, also $p(x) = 0$ gesetzt werden kann, sind die verbleibenden Koeffizienten sogar eindeutig bestimmt. In der Tat, dann ergeben sich nichttriviale Bedingungen auf die $\nu_1 + \ldots + \nu_d = n$ unbestimmten Koeffizienten genau für die n Potenzen $x^0, \ldots, x^{n-1}$. Wir erhalten also ein lineares Gleichungssystem mit n Unbestimmten und n Gleichungen, das nach obigem Satz für jede rechte Seite $f(x)$ eine Lösung besitzt, welche demzufolge eindeutig bestimmt ist.

Allerdings ist ein solcher Ansatz wenig praktikabel, da stets mit allen Nullstellen der Gleichung $g(x)$ zu rechnen ist, d.h. komplizierte und voneinander stark abhängige algebraische Zahlen einzuführen sind, während z.B. in der Stammfunktion

$$\int \frac{3x^2-1}{x^3-x+1}\,dx = \ln(x^3-x+1)$$

überhaupt keine algebraischen Zahlen auftreten.

Wir wollen deshalb, ausgehend von einem Körper k mit $\mathbf{Q} \subseteq k \subset \mathbf{C}$, in dem effektiv gerechnet werden kann, und einer rationalen Funktion $r(x) = \frac{f(x)}{g(x)} \in k(x)$ im Folgenden die einzelnen Schritte des Algorithmus daraufhin untersuchen, inwieweit sie ohne die Einführung (neuer) algebraischer Zahlen ausgeführt werden können.

Für eine detailliertere Diskussion der Integration rationaler Funktionen, insbesondere auch die Behandlung des reellen Falls, sei auf [2, Kap. 2] verwiesen.

6.1 Der rationale Anteil der Stammfunktion

Wir können zunächst $deg(f) < deg(g)$ voraussetzen, da man andernfalls von $r(x)$ durch Division mit Rest einen polynomialen Anteil abspalten kann, dessen Stammfunktion sich leicht berechnen lässt wie in folgendem Beispiel:

$$\int \frac{x^5-x+1}{x^3-x+1}\,dx = \int (x^2+1)\,dx - \int \frac{x^2}{x^3-x+1}\,dx = \frac{x^3}{3} + x - \int \frac{x^2}{x^3-x+1}\,dx$$

Als zweiten Schritt können wir ohne Einführung zusätzlicher algebraischer Zahlen die Berechnung von $\int r(x)\,dx$ auf die Berechnung eines Integral mit *quadratfreiem* Nenner reduzieren, d.h. $g(x)$ durch ein Polynom ersetzen, das nur einfache Nullstellen besitzt. Es stellt sich heraus, dass bei diesem auf M.W.Ostrogradski zurückgehenden Ansatz bereits der gesamte rationale Anteil der Stammfunktion berechnet wird, d.h. dafür im Gegensatz zur Darstellung im Satz von Newton-Leibniz-Bernoulli keine neuen algebraischen Zahlen eingeführt werden müssen.

Wir wollen o.B.d.A. annehmen, dass $g(x)$ monisch ist, also

$$g(x) = (x-\alpha_1)^{\nu_1} \cdot \ldots \cdot (x-\alpha_d)^{\nu_d}$$

gilt. Dann ist

$$g_0(x) = (x-\alpha_1) \cdot \ldots \cdot (x-\alpha_d)$$

ein Polynom, das dieselben Nullstellen besitzt, jedoch jede nur mit der Vielfachheit 1, und schließlich

$$g_1(x) = \frac{g(x)}{g_0(x)} = (x-\alpha_1)^{\nu_1-1} \cdot \ldots \cdot (x-\alpha_d)^{\nu_d-1}$$

der zugehörige Kofaktor.

Satz 19 *Es gilt* $\gcd(g(x), g'(x)) = g_1(x)$ *und damit* $g_0, g_1 \in k[x]$.

BEWEIS : Bezeichnen wir mit

$$g^{(i)}(x) = \frac{g_0(x)}{x - \alpha_i},\ i = 1, \ldots, d$$

das Produkt der Linearfaktoren $(x - \alpha_j)$, $j \neq i$, so gilt nach der Produktregel für die Ableitung

$$g'(x) = \left(\sum_i \nu_i g^{(i)}(x)\right) g_1(x).$$

Da andererseits $g^{(i)}(x) = 0$ für $x = \alpha_j$, $j \neq i$, und $g^{(i)}(\alpha_i) \neq 0$ gilt, ist keine der Nullstellen von $g(x)$ eine Nullstelle des ersten Faktors. Folglich gilt $\gcd(g(x), g'(x)) = g_1(x)$.

Da der größte gemeinsame Teiler z.B. mit dem Euklidschen Algorithmus berechnet werden kann, entstehen seine Koeffizienten durch arithmetische Operationen aus denen von $g(x)$ und $g'(x)$, liegen also ebenfalls in k. $\square$

Ein Vergleich mit dem rationalen Anteil $r_1(x)$ der Stammfunktion von $r(x)$ im Satz von Newton-Leibniz-Bernoulli zeigt, dass als Hauptnenner von r_1 gerade $g_1(x)$ auftritt. Es mag deshalb nicht verwundern, dass $r_1 = \frac{f_1}{g_1} \in k(x)$ gilt und die obige Darstellung durch Partialbrüche an dieser Stelle unnötige algebraische Zahlen einführt. Den Zähler f_1 kann man als Lösung eines entsprechenden linearen Gleichungssystems bestimmen. Dazu machen wir den Ansatz

$$r(x) = \frac{f(x)}{g(x)} = \left(\frac{f_1(x)}{g_1(x)}\right)' + \frac{f_0(x)}{g_0(x)}$$

mit unbestimmten Polynomen f_0, f_1 vom Grad $deg(f_0) < deg(g_0)$ und $deg(f_1) < deg(g_1)$.

Betrachten wir wiederum zunächst ein Beispiel. Sei

$$r(x) = \frac{x^5 - x + 1}{(x^3 - x + 1)^2 (x^2 + x + 1)^3}$$

also

$$\begin{aligned} g(x) &= (x^3 - x + 1)^2 (x^2 + x + 1)^3, \\ g_1(x) &= gcd(g, g') = (x^3 - x + 1)\,(x^2 + x + 1)^2, \\ g_0(x) &= \frac{g}{g_1} = (x^3 - x + 1)\,(x^2 + x + 1) \end{aligned}$$

Formulieren wir die Aufgabenstellung in REDUCE:

```
g:=(x^3-x+1)^2*(x^2+x+1)^3;
g1:=gcd(g,df(g,x));
g0:=g/g1;
f:=x^5-x+1;
f1:=for i:=0:deg(g1,x)-1 sum mkid(a,i)*x^i;
f0:=for i:=0:deg(g0,x)-1 sum mkid(b,i)*x^i;
```

Die beiden letzten Anweisungen erzeugen die Polynome f_1 und f_0 mit unbestimmten Koeffizienten

$$\begin{aligned} f_1 &= a_0 + a_1\,x + a_2\,x^2 + a_3\,x^3 + a_4\,x^4 + a_5\,x^5 + a_6\,x^6 \\ f_0 &= b_0 + b_1\,x + b_2\,x^2 + b_3\,x^3 + b_4\,x^4 \end{aligned}$$

Statt obige rationale Funktionen zu verwenden wollen wir gleich mit $g(x)$ durchmultiplizieren und die Differenz der beiden Seiten bilden. Zuvor ist jedoch noch auf die von REDUCE nicht standardmäßig verwendete *gekürzte Normalform* für rationale Funktionen umzuschalten.

```
on gcd;
s:=f-g*(df(f1/g1,x)+f0/g0);
```

s ist tatsächlich polynomial (vom Grad 11 in x, mit 78 Termen). Koeffizientenvergleich liefert ein lineares Gleichungssystem mit 12 Gleichungen in den ebenfalls 12 unbestimmten Koeffizienten von f_1 und f_0

```
gls:=coeff(s,x);
s1:=solve gls;
```

$$\{\{a_0 = \tfrac{2295}{7889}, a_1 = \tfrac{10394}{23667}, a_2 = -\tfrac{470}{3381}, a_3 = -\tfrac{137}{3381}, a_4 = \tfrac{5548}{23667}, a_5 = \tfrac{4908}{7889}, a_6 = \tfrac{11143}{23667}, \\ b_0 = \tfrac{20158}{23667}, b_1 = -\tfrac{19966}{23667}, b_2 = -\tfrac{1327}{7889}, b_3 = \tfrac{11143}{23667}, b_4 = 0\}\}$$

Setzen wir diese Lösung in f_0 und f_1 ein, so erhalten wir die gesuchte Zerlegung.

```
f1:=sub(first s1,f1);
f0:=sub(first s1,f0);
```

$$f_1 = \frac{11143\,x^6 + 14724\,x^5 + 5548\,x^4 - 959\,x^3 - 3290\,x^2 + 10394\,x + 6885}{23667}$$

$$f_0 = \frac{11143\,x^3 - 3981\,x^2 - 19966\,x + 20158}{23667}$$

Die Probe zeigt, dass wir bis hierher richtig gerechnet haben:

```
f/g-(df(f1/g1,x)+f0/g0);
```

$$0$$

Auf dieselbe Weise können wir auch im allgemeinen Fall vorgehen.

Satz 20 *Sei $r(x) = \frac{f(x)}{g(x)} \in k(x)$ eine rationale Funktion mit $deg(f) < n := deg(g)$ und $g_0, g_1 \in k[x]$ wie oben definiert. Dann gibt es eindeutig bestimmte Polynome $f_0, f_1 \in k[x]$ vom Grad $deg(f_0) < deg(g_0)$, $deg(f_1) < deg(g_1)$, so dass*

$$\int r(x)\,dx = \frac{f_1(x)}{g_1(x)} + \int \frac{f_0(x)}{g_0(x)}\,dx$$

gilt.

Die Koeffizienten der Polynome f_0, f_1 kann man durch Ansatz mit unbestimmten Koeffizienten aus einem eindeutig lösbaren linearen Gleichungssystem gewinnen.

Beweis : Wir betrachten die Gleichung

$$f(x) = \left(\frac{f_1(x)}{g_1(x)}\right)' g(x) + f_0(x)\,g_1(x),$$

die äquivalent zu der Gleichung ist, deren Existenz es zu beweisen gilt. Der gebrochen rationale erste Summand ist in Wirklichkeit auch polynomial. In der Tat, die Quotientenregel und $g = g_0\,g_1$ ergeben

$$\left(\frac{f_1}{g_1}\right)' g = f_1'\,g_0 - f_1\,g_2 \quad \text{mit} \quad g_2 := \frac{g_1'\,g_0}{g_1} \in k(x)$$

Wegen

$$g_1' = \sum_i (\nu_i - 1) g^{(i)}(x) \prod_i (x - \alpha_i)^{\nu_i - 2}$$

ist $g_2 = \sum_i (\nu_i - 1) g^{(i)}(x)$ also sogar ein Polynom aus $k[x]$. Setzen wir nun in

$$f = f_1'\,g_0 - f_1\,g_2 + f_0\,g_1$$

die Polynome f_0 und f_1 mit unbestimmten Koeffizienten an und machen Koeffizientenvergleich, so erhalten wir wiederum ein lineares Gleichungssystem mit n Gleichungen in $deg(g_0) + deg(g_1) = n$ Unbestimmten, das nach dem Satz von Newton-Leibniz-Bernoulli für jedes f wenigstens eine komplexe Lösung hat. Damit besitzt das entsprechende lineare Gleichungssystem eine eindeutige Lösung, die bereits in k^n liegt, da sie z.B. mit dem Gauss-Algorithmus berechnet werden kann. $\square$

Beispiel: Berechnen wir den rationalen Anteil von

$$u := \int \frac{1}{(x^5 - x + 1)^3}\, dx.$$

Die entsprechenden Schritte sind dieselben wie oben

```
g:=(x^5-x+1)^3;
g1:=gcd(g,df(g,x));
g0:=g/g1;
f1:=for i:=0:deg(g1,x)-1 sum mkid(a,i)*x^i;
f0:=for i:=0:deg(g0,x)-1 sum mkid(b,i)*x^i;
on gcd;
s:=1-g*(df(f1/g1,x)+f0/g0);
gls:=coeff(s,x);
s1:=solve gls;
f1:=sub(first s1,f1);
f0:=sub(first s1,f0);
```

Als Endergebnis erhalten wir

$$f_1 = \frac{\begin{pmatrix} 6000000\,x^9 + 6122880\,x^8 + 5645300\,x^7 + 4187625\,x^6 - 10800000\,x^5 + \\ 795200\,x^4 + 1625180\,x^3 + 2892175\,x^2 + 10780750\,x - 5534464 \end{pmatrix}}{16462322}$$

$$f_0 = \frac{3000000\,x^3 + 6122880\,x^2 + 8467950\,x + 8375250}{8231161}$$

und auch die Probe zeigt die Richtigkeit der bisherigen Rechnung

```
1/g-(df(f1/g1,x)+f0/g0);
```

so dass wir schlussfolgern können

$$u = \frac{f_1}{(x^5 - x + 1)^2} + \int \frac{f_0}{x^5 - x + 1}\, dx$$

6.2 Der transzendente Anteil der Stammfunktion

Für die weiteren Betrachtungen können wir den Nenner $g = g_0$ der zu integrierenden rationalen Funktion als quadratfrei voraussetzen. Sei wie oben $\{\alpha_1, \ldots, \alpha_d\}$ die Menge der Nullstellen von g und somit o.B.d.A. $g(x) = \prod_i (x - \alpha_i)$.

Wir wollen zunächst eine Darstellung für die Koeffizienten $c_i \in \mathbf{C}$ in der Formel

$$\int \frac{f(x)}{g(x)}\, dx = \sum_i c_i \, \ln(x - \alpha_i)$$

finden. Diese ist äquivalent zu

$$\frac{f(x)}{g(x)} = \sum_i \frac{c_i}{x - \alpha_i}$$

oder, wegen $g(x) = \prod_i (x - \alpha_i)$, zu

$$f(x) = \sum_i c_i \prod_{j \neq i} (x - \alpha_j).$$

Wie in Abschnitt 5.7. lässt sich dieser Ausdruck mit der Lagrange-Interpolationsformel verbinden

$$f(x) = \sum_i c_i \cdot g'(\alpha_i) \cdot \prod_{j \neq i} \frac{x - \alpha_j}{\alpha_i - \alpha_j}.$$

Die rechte Seite der Formel beschreibt gerade das Interpolationspolynom, das an den Stellen $\alpha_i,\ i = 1, \ldots, n$ die Werte $c_i \cdot g'(\alpha_i)$ hat und mit Blick auf die linke Seite die Werte $f(\alpha_i)$ haben soll. Da das Interpolationspolynom mit $deg(f) < deg(g)$ eindeutig bestimmt ist, ergibt sich für die unbekannten Koeffizienten $c_i = \frac{f(\alpha_i)}{g'(\alpha_i)}$, wobei wir bemerken, dass $g(x)$ nur einfache Nullstellen hat, also $g'(\alpha_i) \neq 0$ gilt.

Wir haben damit den folgenden Satz bewiesen:

Satz 21 *Ist $g(x)$ quadratfrei und $deg(f) < deg(g)$, so gilt*

$$\int \frac{f(x)}{g(x)}\, dx = \sum_\alpha \frac{f(\alpha)}{g'(\alpha)} \ln(x - \alpha),$$

wobei über $\alpha \in \mathrm{RootOf}(g(x))$, die Menge der Nullstellen von g, summiert wird.

Korollar 4

$$\int \frac{1}{x^n - 1}\, dx = \sum_\alpha \frac{\alpha}{n} \ln(x - \alpha),$$

wobei über alle $\alpha \in \mathrm{RootOf}(x^n - 1)$ zu summieren ist.

Dies ergibt sich sofort aus obiger Formel für $f = 1$ und $g = x^n - 1$ unter Beachtung, dass $\alpha^n = 1$ gilt.

Wir haben im letzten Kapitel gesehen, dass man den Quotienten zweier algebraischer Zahlen, hier also $c_i = \frac{f(\alpha)}{g'(\alpha)}$, stets als Linearkombination von Elementen aus T_{red}, hier also von Potenzen $1, \alpha^1, \ldots, \alpha^{d-1}$ darstellen kann. Eine solche polynomiale Darstellung von c_i kann man explizit berechnen, ohne dabei bereits neue algebraische Zahlen einzuführen:

Da g nur einfache Nullstellen besitzt, sind g und g' teilerfremd. Wir finden (etwa mit dem erweiterten Euklidschen Algorithmus) also zwei Polynome $u(x), v(x) \in k[x]$ mit

$$1 = u(x)\, g(x) + v(x)\, g'(x).$$

Da α eine Nullstelle von g ist, gilt mithin $g'(\alpha)\, v(\alpha) = 1$. Als Kofaktor c_i in obiger Integrationsformel kann man also auch den Wert des Polynoms $f(x)\, v(x)$ an der Stelle α einsetzen. Wegen $g(\alpha) = 0$ kann man dieses Polynom sogar noch $(mod\ g(x))$ reduzieren:

Satz 22 *Ist $g(x)$ quadratfrei und $deg(f) < deg(g)$, so existiert ein (effektiv berechenbares) Polynom $c(x) \in k[x]$ vom Grad $deg(c) < deg(g)$, für welches*

$$\int \frac{f(x)}{g(x)}\, dx = \sum_\alpha c(\alpha) \ln(x - \alpha)$$

gilt, wobei wieder über $\alpha \in \mathrm{RootOf}(g(x))$ zu summieren ist.

Auch das Polynom $c(x)$ kann man über einen Ansatz mit unbestimmten Koeffizienten und Koeffizientenvergleich aus einem linearen Gleichungssystem gewinnen. Erfahrungsgemäß sind die Koeffizienten, die in einem solchen Polynom auftreten, wesentlich größer als die in der rationalen Darstellung. Betrachten wir etwa das Beispiel

$$\int \frac{1}{(x^5-x+1)}\,dx = \sum_\alpha \frac{1}{5\alpha^4-1}\ln(x-\alpha),$$

wobei über $\alpha \in \mathrm{RootOf}(x^5-x+1)$ summiert wird. Das entsprechende Polynom $c(x)$ können wir aus $1-c(x)\,g'(x) \equiv 0 \ (mod\ g(x))$, wobei $c(x)$ mit unbestimmten Koeffizienten anzusetzen ist, gewinnen. Mit Reduce erhalten wir

```
g:=(x^5-x+1);
c:=for i:=0:deg(g,x)-1 sum mkid(c,i)*x^i;
s:=remainder(1-c*df(g,x),g);
coeff(s,x);
s1:=solve ws;
c:=sub(first s1,c);
```

$$c(x) = \frac{-320\,x^4-400\,x^3-500\,x^2-625\,x+256}{2869},$$

also

$$\int \frac{1}{(x^5-x+1)}\,dx = \sum_\alpha \frac{-320\,\alpha^4-400\,\alpha^3-500\,\alpha^2-625\,\alpha+256}{2869}\ln(x-\alpha)$$

Die Darstellung des transzendenten Teils in der bisherigen Form hat allerdings immer noch den Nachteil, dass er etwa im Fall $f(x)=g'(x)$ algebraische Zahlen einführt, während zur Darstellung des Ergebnisses solche nicht notwendig sind. Der Satz liefert für $g(x)=x^5-x+1$ und $f(x)=g'(x)$

$$\int \frac{5x^4-1}{(x^5-x+1)}\,dx = \sum_{\alpha\,:\,g(\alpha)=0} \ln(x-\alpha) = \ln(x^5-x+1),$$

wenn man nach den Logarithmengesetzen die Summe von Logarithmen in den Logarithmus eines Produkts verwandelt. Ähnlich kann man vorgehen, wenn einige der Kofaktoren $c(\alpha)$ übereinstimmen, wobei die Hoffnung berechtigt ist, dass dabei einfachere algebraische Zahlen entstehen.

Da es sich bei den $\beta = c(\alpha) = \frac{f(\alpha)}{g'(\alpha)}$ wiederum um algebraische Zahlen handelt, wollen wir zunächst die Frage nach deren Minimalpolynom untersuchen. Offensichtlich ist ein solches Paar (α,β) eine Lösung des Gleichungssystems $\{g(x)=0,\ f(x)=y\cdot g'(x)\}$ und umgekehrt, so dass wir nach einem Polynom $p(y)$ suchen können, das alle y-Komponenten von derartigen Lösungspaaren erfüllen. Das Minimalpolynom von β ist dann ein Faktor dieses Polynoms.

Die Berechnung eines solchen Polynoms führt auf die Elimination der Variablen x aus obigem Gleichungssystem. Diese Eliminationsaufgabe kann man z.B. über die Berechnung der *Resultante* lösen (wobei es genügt, deren quadratfreien Teil zu nehmen):

$$p(y) = \mathrm{res}_x(g(x), f(x)-y\cdot g'(x)).$$

Für jede Nullstelle β von $p(y)$ haben die beiden Polynome $g(x)$ und $f(x)-\beta\cdot g'(x)$ die gemeinsamen Nullstellen

$$\mathcal{R}_\beta := \{\alpha\ :\ g(\alpha)=0\ \wedge\ \frac{f(\alpha)}{g'(\alpha)}=\beta\}.$$

Damit gilt

$$\prod_{\alpha\in\mathcal{R}_\beta}(x-\alpha) = \gcd\,(g(x), f(x)-\beta\cdot g'(x)),$$

so dass wir insgesamt folgende dritte Darstellung für den transzendenten Anteil des Integrals einer rationalen Funktion erhalten, siehe [2, Thm 2.4.1.]

Satz 23 (Rothstein-Trager) *Ist $g(x)$ quadratfrei, $deg(f) < deg(g)$, $\gcd(f,g) = 1$ und $p(y)$ der quadratfreie Teil der Resultante $\mathrm{res}_x(g(x), f(x) - y \cdot g'(x))$, so gilt*

$$\int \frac{f(x)}{g(x)}\,dx = \sum_\beta \beta \cdot \ln\bigl(\gcd(g(x), f(x) - \beta \cdot g'(x))\bigr),$$

wobei über die Nullstellen $\beta \in \mathrm{ROOTOF}(p(y))$ zu summieren ist.

Im Fall $f(x) = g'(x)$ gilt $p(y) = (y-1)$, also $\beta = 1$ und wir erhalten nun unmittelbar

$$\int \frac{g'(x)}{g(x)}\,dx = \ln(g(x)).$$

Doch auch für andere Integrale können sich deutliche Vereinfachungen ergeben wie etwa im folgenden Beispiel [2, 2.4.1.]:

Gesucht ist die Stammfunktion von

$$r(x) = \frac{x^4 - 3x^2 + 6}{x^6 - 5x^4 + 5x^2 + 4}.$$

Der Nenner $g(x) = x^6 - 5x^4 + 5x^2 + 4$ ist quadratfrei und irreduzibel, so dass nach Satz 21 die Antwort algebraische Zahlen vom Grad 6 enthalten würde. Die Berechnung der Resultante $p(y)$ liefert

```
f:=x^4-3x^2+6;
g:=x^6-5x^4+5x^2+4;
p:=resultant(g,f-y*df(g,x),x);
```

$$p(y) = 45796\,(4y^2 + 1)^3$$

und damit $\beta = \pm\frac{i}{2}$, also nur eine quadratische Irrationalität. Von den 6 Summanden, die nach Satz 21 entstehen würden, haben je 3 denselben Kofaktor und können deshalb zusammengefasst werden. Die Berechnung der größten gemeinsamen Teiler über der entsprechenden algebraischen Erweiterung liefert

```
on complex;
gcd(g,f-i/2*df(g,x));
gcd(g,f+i/2*df(g,x));
```

$$x^3 + i\,x^2 - 3\,x - 2\,i$$

$$x^3 - i\,x^2 - 3\,x + 2\,i$$

und schließlich

$$\int r(x)\,dx = \frac{i}{2}\ln(x^3 + i\,x^2 - 3\,x - 2\,i) - \frac{i}{2}\ln(x^3 - i\,x^2 - 3\,x + 2\,i)$$

Wir sehen an diesem Beispiel noch einmal deutlich, welche Rolle algebraische Zahlen im symbolischen Rechnen spielen, wie schwierig der Umgang mit ihnen stellenweise ist und welche Methoden angewendet werden können, um den Einsatz algebraischer Zahlen auf das notwendige Minimun zu beschränken.

Literaturverzeichnis

[1] *Brockhaus Enzyklopädie in 26 Bänden.* F.A. Brockhaus, Mannheim, 1994.

[2] M. Bronstein. *Symbolic integration I — transcendental functions.* Springer, Berlin, 1997.

[3] B. Buchberger. Symbolisches Rechnen. In P. Rechenberg and H. Pomberger, editors, *Informatik-Handbuch*, chapter E5, pages 799 – 817. Hanser, München, 1997.

[4] B. Buchberger and R. Loos. Algebraic simplification. In B. Buchberger, G.E. Collins, and R. Loos, editors, *Computer Algebra - Symbolic and Algebraic Computation*, pages 11–43. Springer, Wien, second edition, 1983.

[5] H. Engesser, editor. *Duden Informatik.* Dudenverlag, Mannheim, 1993.

[6] K.O. Geddes, S.R. Czapor, and G. Labahn. *Algorithms for Computer Algebra.* Kluwer Acad. Publisher, 2 edition, 1992.

[7] J. Grabmeier. Computeralgebra – eine Säule des Wissenschaftlichen Rechnens. *it + ti*, 6:5 – 20, 1995.

[8] J. Grabmeier and V. Weispfenning, editors. *Computeralgebra in Deutschland – Bestandsaufnahme, Möglichkeiten, Perspektiven.* Fachgruppe der GI, DMV, GAMM, Passau and Heidelberg, 1993.

[9] D. Gruntz and M. Monagan. Introduction to Gauss. *Maple Technical Newsletter*, 9, 1993.

[10] D.E. Knuth. *The art of computer programming.* Addison Wesley, 1991.

[11] R. Loos. Introduction. In B. Buchberger, G.E. Collins, and R. Loos, editors, *Computer Algebra - Symbolic and Algebraic Computation*, pages 1–10. Springer, Wien, second edition, 1983.

[12] T. Ottmann and P. Widmeyer. *Algorithmen und Datenstrukturen.* BI Wissenschaftsverlag, 2 edition, 1993.

[13] R. Pavelle, M. Rothstein, and J.P. Fitch. Computer algebra. *Scientific American*, 245(6):102 – 113, dec 1981.

[14] H. Pieper. *Die komplexen Zahlen.* Dt. Verlag der Wissenschaften, Berlin, 1988.

[15] J.K. Prentice and M. Wester. Code generation using Computer Algebra Systems. In M. Wester, editor, *Computer Algebra Systems: A Practical Guide*, chapter 13, pages 233 – 254. Wiley, Chichester, 1999.

[16] P. Rechenberg and H. Pomberger, editors. *Informatik-Handbuch.* Hanser, München, 1997.

[17] Schneider, editor. *Lexikon Informatik.* Oldenbourg, München, 4.0 edition, 1997.

[18] U. Schwardmann. *Computeralgebrasysteme.* Addison-Wesley, 1995.

[19] B. Simon. Comparative CAS review. *Notices AMS*, 39:700 – 710, sept 1992.

[20] B. Stroustrup. *Die C++-Programmiersprache*. Addison-Wesley, 2 edition, 1992.

[21] J.A. van Hulzen and J. Calmet. Computer Algebra Systems. In B. Buchberger, G.E. Collins, and R. Loos, editors, *Computer Algebra - Symbolic and Algebraic Computation*, pages 221 – 243. Springer, Wien, second edition, 1983.

[22] J. Weizenbaum. *Die Macht der Computer und die Ohnmacht der Vernunft*, volume 274 of *Taschenbuch Wissenschaft*. Suhrkamp, 9 edition, 1994.

[23] M. Wester. A review of CAS mathematical capabilities. *CAN Nieuwsbrief*, 13:41–48, dec 1994.

[24] M. Wester, editor. *Computer Algebra Systems: A Practical Guide*. Wiley, Chichester, 1999.

[25] M. Wester. A critique of the mathematical abilities of CA systems. In M. Wester, editor, *Computer Algebra Systems: A Practical Guide*, chapter 3, pages 25 – 60. Wiley, Chichester, 1999.

[26] N. Wirth. *Algorithmen und Datenstrukturen*. B.G. Teubner Verlag Stuttgart, 4 edition, 1986.

[27] S. Wolfram. *The Mathematica Book*. Cambridge University Press, 1996.

Anhang: Die Übungsaufgaben

1. Bestimmen Sie die Anzahl der Stellen von $n\,!$ für $n = 10,\ 100,\ 1000$.

2. Bestimmen Sie die kleinste Primzahl, die größer als $12\,! = 479001600$ ist.

3. Sei $a_n = 3^n - 2$.

   a. Für welche $n < 100$ ist a_n eine Primzahl ?

   b. Bestimmen Sie für $n < 40$ die Primfaktorzerlegung der Zahlen a_n.

   c. Leiten Sie aus den berechneten Zerlegungen Vermutungen her, wann a_n durch 5 und wann durch 7 teilbar ist.

   d. Beweisen Sie diese Vermutungen.

4. Untersuchen Sie, für welche $n < 30$ die Faktorzerlegung von $f(n) = n\,! - 1$ Primfaktoren mehrfach enthält.

5. Eine Zahl $2^p - 1$ ist höchstens dann eine Primzahl, wenn p selbst prim ist. Primzahlen dieser Form nennt man *Mersennesche Primzahlen*. Die größten bekannten Primzahlen haben diese Form. Es ist unbekannt, ob es unendlich viele Mersennesche Primzahlen gibt.

   Bestimmen Sie für $p < 100$ alle diese Zahlen.

   Geben Sie diese Zahlen als Tabelle $[p \quad 2^p - 1]$ an.

6. Die Folge
$$s_1 := 1,\ s_{n+1} := \frac{1}{2}\left(s_n + \frac{2}{s_n}\right)$$
   konvergiert bekanntlich gegen $\sqrt{2}$.

   a. Bestimmen Sie die ersten 8 Werte der Folge als exakte Brüche.

   b. Bestimmen Sie, wieviele Ziffern die Zähler von s_n für $n = 1, \ldots, 12$ enthalten. Analysieren Sie deren Wachstumsordnung.

c. Bestimmen Sie, wie schnell sich diese exakten Werte an $\sqrt{2}$ annähern. (Beachten Sie, dass die Standardgenauigkeit Ihres CAS für numerische Rechnungen dafür möglicherweise nicht ausreicht)

7. Zeigen Sie, dass eine ungerade perfekte Zahl wenigstens drei Primteiler haben muss. Ist sie nicht durch 3 teilbar, so müssen es sogar mindestens 7 Primteiler sein.

   Hinweis: Zeigen Sie, dass für eine perfekte Zahl $n = p_1^{a_1} \cdot \ldots \cdot p_m^{a_m}$ stets
   $$2 < \prod_{i=1}^{m} \frac{p_i}{p_i - 1}$$
   gelten muss.

8. Führen Sie für die Funktion $f(x) = \sin(x) - x \cdot \tan(x)$ eine Kurvendiskussion durch:

   a. Bestimmen Sie die Null- und Polstellen der Funktion $f(x)$ (notfalls näherungsweise).

   b. Zeigen Sie, dass die Funktion außerhalb des Intervalls $]-\frac{\pi}{2}, \frac{\pi}{2}[$ in ihren Stetigkeitsintervallen monoton ist.

   c. Untersuchen Sie das Verhalten der Funktion im Intervall $]-\frac{\pi}{2}, \frac{\pi}{2}[$.

9. $a = e^{\pi\sqrt{163}}$ kommt einer ganzen Zahl sehr nahe.

   a. Bestimmen Sie diese Zahl und die Größenordnung der Abweichung.

   b. Es gilt $\sqrt[3]{a} \sim 640\,320$ auf 9 Dezimalstellen genau. Finden Sie die ganze Zahl b, so dass die Approximation $\sqrt[3]{a+b} \sim 640\,320$ bestmöglich ist und geben Sie auch hier die Größenordnung der Abweichung an.

10. Bestimmen Sie die Wachstumsordnung d und -rate C der Funktion
    $$f(x) := \sin(\tan(x)) - \tan(\sin(x))$$
    in der Nähe von $x = 0$, d.h. solche Zahlen, dass $f(x) = C \cdot x^d + \langle \text{kleinere Terme} \rangle$ gilt.

10. In der Vorlesung wurde der rationale Ausdruck
    $$u_n := \frac{a^n}{(a-b)*(a-c)} + \frac{b^n}{(b-c)*(b-a)} + \frac{c^n}{(c-a)*(c-b)}$$
    betrachtet und festgestellt, dass er sich für $n \leq 5$ zu einem Polynom in a, b, c vereinfachen lässt.

    Beweisen Sie diese Eigenschaft allgemein.

    a. Zeigen Sie zunächst, dass sich ein Polynom $p(x, y)$ mit der Eigenschaft
    $$p(x, y) = -p(y, x)$$
    in der Form
    $$p(x, y) = (x - y) \cdot q(x, y)$$
    mit einem anderen Polynom $q(x, y)$ darstellen lässt.

    b. Wenden Sie diese Eigenschaft auf das Zählerpolynom von u_n bzgl. der Paare (a, b), (a, c) und (b, c) an, nachdem Sie den Hauptnenner gebildet haben, um damit den geforderten Beweis zu erbringen.

    c. Führen Sie einen alternativen Beweis, der die (vorher zu beweisende) Beziehung
    $$u_n = \frac{b^{n-1} - c^{n-1}}{b - c} + a \cdot u_{n-1}$$
    verwendet.

d. Leiten Sie daraus her, dass u_n mit der vollen symmetrischen Funktion h_{n-1} übereinstimmt, d.h.

$$u_n = h_{n-1}(\{a, b, c\})$$

gilt.

11. Mit dieser Aufgabe soll ein CAS als Problemlösungsumgebung eingesetzt werden. Wir wollen dazu die Frage studieren, ob es Fibonaccizahlen gibt, die mit vielen Neunen enden. Die Fibonaccizahlen sind bekanntlich durch die Rekursionsrelation

$$F_0 = F_1 = 1, \quad F_n = F_{n-1} + F_{n-2} \text{ für } n > 1$$

definiert.

a. Finden Sie die erste Fibonaccizahl, die auf 0 endet.

b. Finden Sie die erste Fibonaccizahl, die auf 00 endet.

c. Untersuchen Sie, ob es Fibonaccizahlen gibt, die auf 0000 enden. Überlegen Sie sich dazu einen geeigneten Ansatz, mit dem die auszuführenden Rechnungen überschaubar bleiben. Geben Sie alle $n < 100000$ an, für die F_n auf 0000 endet.

d. Beweisen Sie, dass es Fibonaccizahlen mit beliebig vielen Nullen nach dem Komma gibt.

12. Geben Sie für die folgenden mathematischen Ausdrücke an, wie sie in Ihrem CAS einzugeben sind und finden Sie jeweils die interne Darstellung des Ergebnisses als geschachtelte Liste heraus.

a) $x - y + z, \ a/b * c$,

b) Intervall der Zahlen zwischen 0 und 2,

c) Matrix $\begin{pmatrix} 1 & 1 \\ 2 & 3 \end{pmatrix}$,

d) $\frac{d}{dx} f(g(x))$,

13. Es sei

$$S_n := \sum_{k=1}^{n} \frac{1}{\sin(k)}.$$

a) Bestimmen Sie die Struktur des Ausdrucks, den Ihr CAS für S_{10} ausgibt.

b) Bestimmen Sie die Struktur des Ausdrucks, den Ihr CAS für S_n ausgibt.

14. a) Lösen Sie in Ihrem CAS das Gleichungssystem

$$x^2 + y^2 = 4, \ x + y = 1.$$

Die gefundene Lösung soll explizit sein und keine `RootOf`-Symbole enthalten.

b) Bestimmen Sie die Struktur des Ausdrucks, der als Lösungsmenge angegeben wird.

c) Wie kann mit Ihrem CAS durch eine Probe die Korrektheit der Antwort geprüft werden ?

15. Stellen Sie die x-Werte, an denen die Funktion

$$g(x) = 12 - 24x + 22x^2 - 8x^3 + x^4$$

lokale Extrema hat, in einer Liste l zusammen und erzeugen Sie durch geeignete Listenmanipulationen die Liste der Extremwertkoordinaten (x_i, y_i) der lokalen Extrema von $g(x)$.

16. Zerlegen Sie das Polynom $x^4 + 4$ in Faktoren

    a. über den ganzen Zahlen,
    b. über $\mathbf{Z}/13\mathbf{Z}$,
    c. über $\mathbf{Z}/17\mathbf{Z}$,

    und prüfen Sie das Ergebnis jeweils durch Ausmultiplizieren.

    Handelt es sich bei dem von Ihnen verwendeten Befehl um einen Funktionsausdruck, einen Funktionsaufruf oder eine Funktionstransformation ? Begründen Sie Ihre Antwort.

17. Erstellen Sie in Ihrem CAS unter Verwendung des Listengenerators ein quadratisches Schema der Ausdrücke $gcd(x^m - 1, x^n - 1)$ für $m, n = 1, \ldots, 5$.

    Verwandeln Sie anschließend dieses Schema in eine Matrix.

18. Finden Sie mit geeigneten Listenoperationen alle Lösungen des Gleichungssystems

    $$\{x^3 + y = 2,\ y^3 + x = 2\},$$

    für die $x \neq y$ gilt.

    Hinweis: Verwenden Sie den Operator `solve`, organisieren Sie dessen Ausgabe in eine Liste, ersetzen Sie "suspekte" Terme durch numerische Näherungswerte und wählen Sie dann diejenigen Terme aus, für die $x \neq y$ gilt. Das sind 6 der 9 (komplexen) Lösungen des gegebenen Gleichungssystems.

19. Die meisten CAS kennen exakte Werte von Winkelfunktionen mit dem Argument $\frac{m}{n}\pi$ und $n \leq 6$. Bestimmen Sie daraus einen exakten Wert von $\sin(6^o)$.

    Hinweis: Wegen $6^o = \frac{\pi}{30}$ gilt $\sin(6^o) = \sin(\frac{\pi}{5} - \frac{\pi}{6})$. Versuchen Sie mit `expand` diesen Ausdruck entsprechend den Additionstheoremen zu zerlegen.

Beim Vereinfachen geschachtelter Wurzelausdrücke zeigen sich die Systeme oftmals unerwartet schwerfällig. Um so überraschender mag es sein, daß Vereinfachungen wie

$$\begin{aligned}
\sqrt{-6\sqrt{2} + 11} + \sqrt{6\sqrt{2} + 11} &= 6 \\
\sqrt{-2\sqrt{6} + 5} + \sqrt{2\sqrt{6} + 5} &= 2\sqrt{3} \\
(6\sqrt{3} - 10)^{\frac{1}{3}} + (6\sqrt{3} + 10)^{\frac{1}{3}} &= 2\sqrt{3}
\end{aligned}$$

teilweise automatisch ausgeführt werden.

20. Finden Sie eine allgemeine Regel, wann

    $$\sqrt{a + 2 \cdot \sqrt{b}} = \sqrt{c} + \sqrt{d}$$

    gilt, wobei $a, b, c, d \in \mathbf{N}$ gelte und b kein volles Quadrat sei.

    Kann man Ihre Antwort auf rationale Funktionen a, b, c, d mit ganzzahligen Koeffizienten erweitern ?

21. Finden Sie eine allgemeine Regel, wann

    $$\sqrt{a + \sqrt{b}} + \sqrt{a - \sqrt{b}}$$

    zu einer ganzen Zahl vereinfacht werden kann ($a, b \in \mathbf{N},\ b$ kein volles Quadrat).

    Kann man Ihre Antwort auf rationale Funktionen a, b, c, d mit ganzzahligen Koeffizienten erweitern ?

22. Vereinfachen Sie die folgenden trigonometrischen Ausdrücke so weit wie möglich, indem Sie geeignete Regelsysteme zur Anwendung bringen.

    a) $\tan(a)\,\tan(b) + \tan(b)\,\tan(c) + \tan(c)\,\tan(a)$, wenn $a+b+c = \frac{\pi}{2}$ gilt,

    b) $\tan(x) + 2\,\tan(2x) + 4\,\tan(4x) + 8\,\cot(8x)$,

    c) $\sin(x) \cdot \left(1 + \tan(x)\,\tan\left(\frac{x}{2}\right)\right)$.

23. Lösen Sie die folgenden Gleichungen. Achten Sie darauf, dass die jeweilige Lösungsmenge vollständig ist.

    a) $\sin(x)\sin(2x)\sin(3x) = \frac{1}{4}\sin(4x)$,

    b) $\sin(x)\sin(3x) = \frac{1}{2}$

24. Bestimmen Sie das Minimalpolynom von $c := \sqrt{p} + \sqrt{q}$ für $p, q \in \mathbf{N}$. Zeigen Sie dazu insbesondere, dass c außer für triviale Fälle stets eine algebraische Zahl vom Grad 4 ist.

25. Finden Sie ähnlich wie in der Vorlesung die charakteristischen Polynome für die speziellen Winkelfunktionswerte $\sin(\frac{\pi}{9})$ und $\cos(\frac{\pi}{9})$.

26. Finden Sie die charakteristischen Polynome für die speziellen Winkelfunktionswerte $\tan(\frac{\pi}{7})$ und $\tan(\frac{\pi}{9})$.

27. Bestimmen Sie das Minimalpolynom von $c := \sqrt{5} + \sqrt[3]{7}$. (Hinweis: Ermitteln Sie die kleinstmögliche lineare Abhängigkeitsbeziehung zwischen den Potenzen von c.)

28. Vereinfachen Sie $(2^{1/2} + 3^{1/3})^{-1}$, indem Sie diesem Ausdruck als Summe von Einfachwurzeln darstellen.