mathPAD

Vol 7 No 1, September 1997

In diesem Heft

Benno Fuchssteiner	CeBIT 1997: Eine Nachlese	1
Christian Heckler, Torsten Metzner, Paul Zimmermann	Progress Report on Parallelism in *MuPAD*	6
Andreas Sorgatz	Dynamic Modules – C++ Kernel Extensions	16
Meinolf Sellmann	Lineare Optimierung in *MuPAD*	21
Werner M. Seiler	Computer Algebra and Differential Equations — An Overview	34
Winfried Fakler	Algorithms for Solving Linear ODE's	50
Stefan Wehmeier	Polynomfaktorisierung in *MuPAD*	60
Jürgen Billing	Algebraische Eigenschaften in MuPAD	63
Frank Postel	Ein tutorielle Lernsoftware zur Algebra der Sekundarstufe I	73
Oliver Kluge	MuPAD Pro: Das Computeralgebra-System für Windows 95 / NT	80
Ralf Hillebrand	First *MuPAD* User Workshop '97	82
	Kalender	85
	Surfin' the Web	87

Redaktion:
Benno Fuchssteiner
Gerald Rüscher
Frank Postel
Universität-GH Paderborn
D-33095 Paderborn
e-mail:
benno@uni-paderborn.de,
jerry@uni-paderborn.de,
frankp@uni-paderborn.de

ISSN 0941-9187
V.i.S.d.P.: Benno Fuchssteiner
Nachdruck gegen Belegexemplar erlaubt

Liebe Leserin, lieber Leser,

sozusagen auf den letzten Drücker wurde diese mathPAD fertiggestellt, sie erscheint rechtzeitig vor dem ersten MuPAD Nutzer- und Expertentreffen, dem MuPAD User Workshop 97, der am 22. und 23. September im Paderborner Technologiepark stattfindet. Die vielfältigen Beiträge auf diesem Workshop in Form von Vorträgen, Poster-Sessions und Demonstrationen sind jedoch nicht Gegenstand der vorliegenden mathPAD, sondern werden nach dem Workshop separat veröffentlicht werden. Nichtsdestotrotz glauben wir, auch diesmal mit einer interessanten und gut gefüllten mathPAD aufwarten zu können.

Aufregend waren die letzten Wochen und Monaten, denn das MuPAD-Projekt verläßt - zumindest teilweise - die sicheren Gefilde der Hochschule. Mit der Gründung der Firma SciFace GmbH & Co KG wurde ein kommerzielles Split-Off durchgeführt, die Firma SciFace wird – in enger Kooperation mit der Hochschulgruppe – in Zukunft eine kommerzielle Version von MuPAD pflegen, weiterentwickeln und auf dem freien Markt anbieten. Weiterhin steigende Nutzerzahlen auf der Basis solider und moderner Technologie geben uns dabei allen Grund zum Optimismus. Die normale Fortentwicklung und Forschung an Institut und Hochschule wird dabei selbstverständlich fortgeführt, und wir legen Wert darauf, zu betonen, daß es auch in Zukunft stets eine für Forschung und Lehre freie Version von MuPAD geben wird. Einen Artikel zu dem neuen, kommerziellen Standbein des Projektes findet man in diesem Heft.

Herzlich danken möchten wir unseren Autoren, deren unermüdlicher Fleiß diese mathPAD erst ermöglicht hat. Ganz besonders hervorheben möchten wir in diesem Zusammenhang unsere Kollegen Werner Seiler und Winfried Fakler aus Karlsruhe, die nicht nur zwei Artikel zu der vorliegenden Ausgabe beigesteuert haben, sondern sich – zusammen mit anderen Kollegen – schon seit geraumer Zeit durch ihre Arbeiten zum Thema Differentialgleichungen sehr verdient gemacht haben.

Mit freundlichen Grüßen

B.F., G.R. & F.P.

CeBIT 1997: Eine Nachlese

Benno Fuchssteiner
Universität–Gesamthochschule Paderborn, benno@uni-paderborn.de

Halle 22

Früher war Halle 22 eher langweilig. Deshalb habe ich dieser Halle immer einen großen Teil der CeBIT-Besprechung gewidmet, um meine spöttische Ader zu befriedigen. Der Fairneß halber beginne ich damit auch diesmal. Und erstaunlicherweise ist es mir gelungen, dort eine ganze Reihe interessanter Exponate zu finden: Nicht mal auf die eigenen Vorurteile kann man sich noch verlassen!

Zuerst habe ich mich auf dem Stand Forschungsland NRW umgesehen, wo ich einfach wahllos etwas herausgriff. Ich bin bei der Fuzzy-Logik gelandet, wo man Regelungssysteme ausstellte. Zum Beispiel rollende Kugeln, die auf komplizierten ebenen Bahnen rollen sollten, sowie balancierende Stäbe, diesmal mit Fuzzy-Logik gerechnet, nicht mit Differentialgleichungen, was die erste schmerzhafte Erkenntnis beschert: Differentialgleichungen sind nicht alles, Fuzzy-Logik scheint schneller, und hat vor allem den Vorteil, daß sie nicht in instabile Zustände zu führen scheint. Ein dickes Kompliment an die Kollegen aus Dortmund für ihr interessantes Exponat.

Dann ging es zum Springer Verlag (der von der Wissenschaft, nicht von *Bild*). Ein Verlag, zu dem ich immer ein ambivalentes Verhältnis gehabt habe, der mir symptomatisch für deutsche Verlage erschien, mit ihrem behäbigen Eingehen auf neue technische Möglichkeiten. Aber da tut sich was! Ich habe mir *Link* angesehen, einen interessanten Service, der jedem Hochschulmitglied komfortablen Zugriff zu einer Vielzahl von Springer Publikationen gibt, sofern sie in der Bibliothek vorhanden sind. Gut gemacht! Datenstrukturen und Datenbankeigenschaften mögen zwar noch zu verbessern sein, aber das ganze ist ein brauchbares Werkzeug und keine elektronische Spielerei. Eine schmerzhafte Erkenntnis allerdings: Obwohl Springer das seit einiger Zeit anbietet, hat meine Paderborner Hochschule keinen Zugriff darauf, was nicht am Verlag liegt, sondern an uns, und das obwohl wir so fortschrittlich sind. Dann habe ich mir noch die CD *Die Vögel Europas* angesehen. Sehr gut gemacht diese CD! Schöne Vogelstimmen, bunte Bilder, vieles, was den Charme von Multimedia ausmacht. Ein schönes Spielzeug, aber auch etwas für den ernsthaften Ornithologen. Für 98 DM ein großartiges Weihnachtsgeschenk. Ein Kompliment muß ich dem Springer Verlag auch für sein Standpersonal machen: Die junge Dame von den *Life Sciences*, die mich durch die Angebote führte, war enthusiastisch, kritisch und unterschied sich von dem drögen Messepersonal, das man an anderen Ständen findet, aufs angenehmste. So jemand würde selbst unseren Stand noch verbessern.

Danach zur GMD, einem Lieblingsopfer meiner früheren CeBIT-Besprechungen, immer gut für eine spöttische Bemerkung. Dort meine Standardfrage: was das Aufregendste und Tollste sei, was man auf ihrem Stand zu bieten haben. Man offerierte mir Cyberstage. Aber sofort sehen konnte ich es nicht! Das Exponat war so umlagert, bzw. die Brillen, durch die man schauen mußte, daß ich trotz Presseschild nicht sofort drankam. In der Tat, die Schlange war so groß, daß man sich in Halle 1 wähnte! Ich lasse mir also einen Termin für das begehrte Exponat geben. Sonntag Morgen, 09.30 Uhr, und der Leser muß mit mir warten.

Die Süddeutsche führt zwar Halle 22 in ihrer Liste der Mauerblümchen auf, weist aber darauf hin, daß sich da doch etwas getan hat.

Halle 1 und ähnliches

Also jetzt mal in eine andere Halle. Grundsätzlich gehe ich immer zu den Firmen Siemens-Nixdorf, Apple, Sun und in diesem Jahr zum ersten Mal zu Vobis.

Bei Siemens-Nixdorf in Halle 1 fragte ich mich gezielt zu den Neuheiten im Parallelrechenbereich durch. Als erstes wurde mir die Relay (oder so ähnlich, ich mag mich heute nach einigen Monaten nicht mehr so genau erinnern) gezeigt. Gezeigt! Nur als Rechner, also als Gehäuse mit Einschüben ohne eine irgendwie dazu laufende Software oder Anwendung. Faszinierend diese vollständige Abstraktion von allem was ein Rechner tun soll und wofür er da ist. Erstaunlich, daß es wohl manchen Besucher geben muß, der so was nach der Ästhetik des Gehäuses bestellt!

Meine Frage nach der Architektur wird selbstsicher beantwortet: es gäbe keine Architektur. Nachfrage Shared Memory oder sowas. Nein, *Shared nothing*. Nun warum eigentlich nicht, keine Kommunikationsmechanismen, keine Verteilungsmechanismen!? Ein typisch tiefparalleles System, wie es im Prospekt heißt. Ich lasse nicht locker: Was sei denn nun der Vorteil gegenüber der gleichen Anzahl von Workstations durch eine schnelle Netzverbindung gekoppelt? Eigentlich keiner, bis auf die Tatsache, daß man nur eine Konsole brauche bzw. daß man zusätzlich noch einen *global namespace* habe. Hartnäckig komme ich auf meinen Vergleich mit den über das Netz gekoppelten Workstations zurück und frage, was der Unterschied des Programmiermodells bei beiden Plattformen sei. Nun, da gebe es keinen. Ein großer Vorteil sei aber, daß man eine Platte bei einem Prozessor rausziehen könne und beim anderen reinstecken könne. Das Netzwerk würde dann diese Daten finden, da es eben einen *global namespace* gäbe. Ob man nicht einen *global namespace* auch mit einem Cluster von Workstations realisieren könne? Ja, realisieren könne man alles, aber eben anders.

Also, ich kapier das einfach nicht (oder nicht alles): Ganz sicher ist das eine pfiffige Rechnerplattform, aber gut wäre es, dieses auch in leicht verständlichen - oder meinetwegen auch schwer verständlichen, aber immerhin verständlichen - Äußerungen von Standpersonal erklärt zu bekommen. Nebenbei bemerkt: ich hatte mein Presseschild an, da mußte der Herr vom Standpersonal doch sehen, daß ich ein blutiger Laie sei, so einem bedarften Pressevertreter muß man doch irgendwas geben, da reicht es doch nicht, einfach nur Selbstbewußtsein auszustrahlen. Es mag auch sein, daß ich mich dumm anstellte und die Subtilitäten der Parallelität bei typischen Datenbankanwendungen einfach nicht verstehe.

Frustriert geht es zum nächsten Rechner. Diesmal ein wirklicher Shared Memory-Rechner, nämlich der RM600 E Enterprise Server. Hier wieder meine Frage, was denn nun eine typische Anwendung sei. Eigentlich jede. Wieso denn? Nun, die Parallelisierung würde automatisch durch den Compiler besorgt. Meine Nachfrage: Datenflußanalyse? Ja, so etwas. Ob eine typische Anwendung laufe? Nein, eigentlich nicht. Das Besondere am Exponat sei, daß man hier die Verbindung zwischen dem RM600 E und Relay und RM1000 (so ähnlich waren die Typenbezeichnungen) geschafft habe. Und was das technisch Neue daran sei? Nun, man habe den Rechner im November angekündigt und jetzt stände er schon hier, wirklich und unzweifelbar. Immerhin auch eine Leistung! Frage nach *Speed up* oder ähnlichem. Nun, das könne man nicht sagen, da ja die Parallelisierung durch die Compiler vorgenommen würde. Also wieder dieselbe Erfahrung: Ich verstehe nichts von Parallelrechnern, was in der Tat sein mag.

Früher mutmaßte ich bei Exponaten der heimischen Computerindustrie meist, daß da von nicht ganz kundigem Standpersonal solide Technik durch Verwendung großer Worte zu heißer Luft aufgeblasen wurde. Diesmal nicht, ich kann es wohl einfach nicht beurteilen. Trotzdem, wie soll es dann der durchschnittliche Besucher von Halle 1 verstehen können?

Ich komme deshalb nochmal zurück und frage den Kollegen vom Stand, was denn das aufregendste auf dem großen Stand von SNI sei, das überzeugendste Exponat, das, wo er seine Freunde hinschicken würde und ihnen sagen würde, das müßt Ihr unbedingt sehen. Nun ja, etwas hätte ich ja gesehen, das andere kenne er nicht so genau.

Ansonsten war der Stand OK, Bilder, Musik, Buntes und Dunkelgehäusiges gleichermaßen. Stoßseufzer: Warum wendet man sich in einem Land, in dem in Zukunft auch der Schwerpunkt auf pfiffigen wissenschaftlichen Anwendungen liegen sollte, in seinen Exponaten nur an die Finanzwelt, oder an den kritiklosen Internetgläubigen, nur an denjenigen, der wissen will, ob etwas funktioniert, und nicht auch mal an den, der wissen will, wie es funktioniert, oder warum es schnell oder langsam funktioniert?

Der gemischte Eindruck von SNI aus Halle 1 wird allerdings mehr als wettgemacht durch die pfiffigen Produk-

te und die gute Präsentation der Firma in Halle 12. Hier werde ich von einem waschechten Berliner, den es nach Bayern verschlagen hat, herumgeführt, dem man den Spaß an den Produkten anmerkt, der einem nicht das Blaue vom Himmel erzählt, sondern eine kritische Bewertung der eigenen Produkte vornimmt, über Perspektiven verfügt, allgemeine Dinge beurteilen kann. Fazit: Mein Weltbild über die heimische Firma gerät total ins Wanken. Es kann nur aufwärts gehen: Solide Rechnerzugangchipkarten, jede Karte paar Pfennige, das Lesegerät, je nach Ausführung, irgendwas zwischen 150 und 300 DM. Ein interessantes Video-Telefonsystem, gegenwärtiger Preis 800 DM. In Zukunft ein modifiziertes System für 200 DM oder weniger. Erforderlich nur ein normales Modem, eine ordentliche Qualität, ein pfiffiges Produkt für den Low End Markt. Endlich nehme ich auch eine neue DVD-CD in die Hand. Sie sieht genau aus wie die übliche CD und faßt bis zu maximal 17 Gigabyte, Lesegeschwindigkeit 1,5 Megabyte und demnächst 3 Megabyte pro Sekunde. Preis des Lesegeräts bei SNI 800 DM mit Anschlußmöglichkeit an ein normales Fernsehgerät. Ein neues Notebook, die Erweiterung von Mobile 700 mit abnehmbarem Bildschirm, der gleich als Overhead-LCD-Display benutzbar ist. Preisdifferenz zum konventionellen Display ca. 900 DM. Und so weiter, und so weiter.

Stoßseufzer als Nachtrag: Wie gern würde ich hören, daß diese pfiffigen Produkte aus dem Werk Paderborn stammen. Aber immer nur Augsburg oder sonst etwas im Süden. Schade eigentlich!

Langer Weg durch Halle 3: SAP, SAP überall. Ïhr Sprung ins nächste Jahrtausend", "Wir haben uns deshalb für SAP R3 entschieden, weil ... ". Schon nervig das Ganze! Ich wende mich deshalb vom SAP-Stand ab, in eine andere Richtung. Was steht da? SAP. Also in die dritte Richtung. Wieder SAP. Vierte Richtung, die Windrose ist durchschritten, immer wieder SAP.

Danach Ankunft vor dem blauen Triumphbogen. Die Aufschrift "Where do you want to go today?". Kein Firmenlogo, nichts, rein gar nichts. Blau, Sphärenmusik aus dem Inneren. Amphitheater-Atmosphäre. Ein Bildschirm im Rund, 180 Grad sind es mindestens, darauf schemenhaftes Huschen verschwimmender Konturen. Kein Logo, nichts. Ich weiß nicht, was es soll, bin aber beeindruckt. Vermutlich sind es Aufnahmen der Zellenmanipulation beim Klonen des modernen Menschen, so stelle ich mir das jedenfalls vor.

Eine ganze Weile geht das so, der Raum ist überfüllt, die Sitze sind besetzt, man steht, man ist fasziniert. Wir sind bei Microsoft! Bald sieht man es auch. Gar nicht mal aufdringlich. Es erscheint ein Bild an der Oberfläche, Windows oder Windows NT. Dann ein Bildschirm und darauf in der Mitte ein Paar roter voller Lippen. Fünfmal nebeneinander über den ganzen Halbkreis, tolle Lippen, symbolisierend den sex-appeal des Rechners. Sie öffnen sich, sie schließen sich. Ist das etwa sexistische Werbung? Nein, denn schaut man näher hin, sieht man, es sind keine Lippen, es sind Quallen, verschlingende Quallen. Wir sind ja bei Microsoft! Was es soll, weiß ich nicht. Scott McNealy mit seinen rüden Angriffen vom Vortag wird mir wieder sympathisch. Doch irgendwie gekonnt ist es schon.

Stoßseufzer: Hätte ich doch irgendwann mal die Souveränität, die Hallen 1 bis 3 gänzlich auszulassen.

Gang zu Apple: Wohl in der Hoffnung, etwas Neues zu finden. Eine Hoffnung, die nun schon oft enttäuscht wurde. Aber etwas Neues gibt es doch. Das Sammlerstück 20th Anniversary, ein Rechner mit LCD-Display, in einem Stück in der Größe eines Familienporträts von der Dicke 2 bis 3 cm, technisch gut ausgestattet, ein 250 Megahertz Prozessor. Nichts für den Technikfreak, wohl aber für denjenigen, der für seine Bang und Olafsen-Anlage Tausende von Mark ausgeben würde. 600 Stück sind für Deutschland gedacht, 7500 $ kostet es. Ein wirklich schickes Stück. Apple auf dem Weg zum Sammlermarkt, ein eher nostalgischer Markt und technikabgewandt. Der Rechner ist gleichzeitig Fernseher, usw. : das Gerät für alles. Was fehlt ist ein Telefon, ein Rasenmäher und ähnliches. Aber haben täte ich ihn schon gern tun.

Die absoluten Messerenner sind aber die Stände mit Gewinnspielen wie bei Olympia und Toshiba. Da drängt sich alles, da kommt man kaum vorbei, da gibt es was. Umlagert auch der Stand von Pearl in Halle 9 mit seinem Angebot von Billigtutorials gängiger Programme zum Sonderpreis. Die Verteilung des Publikumsinteresses ist etwas deprimierend für denjenigen, der von der CeBIT technische Durchbrüche erwartet. Die Unterscheidung zwischen CeBIT und CeBIT Home scheint nicht vollkommen geglückt.

Technisch pfiffig und neu erscheint der neue Drucker eines japanischen Druckerherstellers (Arps oder so), den es schon seit 1948 geben soll, der mir bisher aber nicht durch innovative Produkte besonders aufgefallen ist. Ein Drucker zum Preis von 1500 DM, der scannt (600 dpi), dokumentenecht druckt, fast alles bedruckt, von der Normalfolie über das T-Shirt bis zum Pappkarton. Ausprobieren müßte man das Ding mal.

Dann der Gang zu Vobis, um den neuen Highscreen Alpha 5000 zu sehen. Auf keiner Messe habe ich mir je den Vobis-Stand angesehen, aber wenn der Lieferant für den Massenmarkt im unteren Preissegment auf einmal einen Rechner mit 500 Megahertz herausbringt, das ist schon eine Neuigkeit. Welche Zielgruppe hat man da im Auge? Natürlich kommen die üblichen Kompatibilitätsfragen von meiner Seite, die von dem freundlichen und recht kompetenten Mann am Stand nicht beantwortet werden können. Erfreulich: ein Mitarbeiter einer Firma, der kritisch auf die eigenen Grenzen der eigenen Kenntnis hinweist, einen kompetent weiterleitet und selbst zugibt, daß er nicht weiß, was die Firma damit beabsichtige. Interessant ist es schon. Als Schreibtischrechner für mich, mit *MuPAD*, erscheint dieses das ideale Gerät.

Zurück in Halle 22

Mathematik interessiert mich natürlich besonders. Also schaue ich im Messeinformationssystem nach, was es da gibt. Leider nur sehr wenig, zumindest im Informationssystem. Einem Mathematiker tut es schon weh zu sehen, daß bei der größten Messe betreffend eine Technologie, von der es im Davis-Report heißt, sie sei wesentlich Mathematiktechnologie, das Keyword *Mathematik or mathematical or* was auch immer die Variation sein mag, nur von ganz wenigen Ausstellern geführt wird. Kein überzeugendes Argument für das, was meine Kollegen und ich so leicht im Munde führen, daß wir heute in einer mathematischen Welt leben würden, oder einer Welt, die durch die Fortschritte und Erfolge der Mathematik geprägt sei. Etwas finde ich aber unter Mathematik, nämlich, daß das Karl-Weierstraß-Institut auf dem Stand Berlin ausstellen würde.

Ich bin also zurck in Halle 22. Ich gehe hin und schaue mir die Simulation chemischer Anlagen an. Ich frage den Standbetreuer, was denn da gerechnet worden sei oder um was es gehe. Was holt er hervor? Eine Reihe von Papierausdrucken, darunter eine Skizze mit handgeschriebener Überschrift *4 Kurven*, die vom Computer ausgedruckt wurden, und bei denen nicht erläutert ist, um was es sich handelt. Die Kurven werden allerdings als großartiges Ergebnis gedeutet und angeboten.

Ich erfahre: Daß es sich um ein neues vom BMBF gefördertes Projekt handele, daß 5 Leute daraus bezahlt würden, daß man natürlich die Absicht habe, irgendwann ein fertiges Produkt daraus zu machen, daß es ansonsten das fertige Produkt schon gäbe von anderer Seite und daß es vielfach verwendet würde. Keineswegs überzeugend, doch begeistert bin ich eigentlich von der demonstrativen und rührenden Bescheidenheit des Exponatbetreuers. Ich beschließe also, nicht locker zu lassen und frage und frage und frage. Ich bekomme auch Antwort, durchaus nicht unfreundlich, sondern geduldig und mit immer größerer Präzision. Keine Information wird allerdings freiwillig gegeben. Alles muß man nachfragen. Nun, was erfahre ich, es handelt sich um ein paralleles Projekt, das auf einer Cray gerechnet wird. Man habe es mit einem neuartigen Multilevel-Newtonverfahren zu tun, und die Parallelität bestände darin, daß man dünn besetzte Matrizen in Blöcke zerlege. Recht interessant! Also will ich - das ist ja die durchschnittliche Mentalität des Messebesuchers - etwas auf dem Bildschirm sehen. Ich will irgend etwas sehen: etwas, das funktioniert, das Bild des Institutsleiters, irgendeine Bewegung. Mühsam werden Dateien eingeladen, es wird lange kompiliert, es erscheint eine Meldung am Bildschirm, daß kompiliert wird. Die Oberfläche ist dürftig, aber es passiert etwas! Ich glaube, was dahinter steckt, ist eine gute Sache. Allerdings stürzt es dann ab, und ich versuche, aus der Sache herauszukommen, murmele etwas davon, daß ich es im Prinzip verstanden habe, das es in Ordnung sei, daß es sicher so sei, daß es eines Tages eine großartige Sache wäre, erfahre noch, daß es sich durch geringe Latenzzeiten auszeichnet.

Na ja, das ist das andere Extrem eines Standbetreuers. Die einen versehen einen *nothing* mit gängigen Begriffen, blasen heiße Luft hinein, und verkünden diese mit bewundernswertem Selbstbewußtsein. Die anderen haben sicher was vorzuweisen, verpacken es schlecht, machen keine ordentlichen Oberflächen dazu, sind solide, gehen aber

auch nicht auf jemanden zu, der auf dem Stand erscheint, sondern sind erst in die Enge getrieben zu Auskünften bereit. Mischen müßte man sie!

GMD zweiter Teil: Sonntag morgen, 09.30 Uhr, ich bin pünktlich da. Dann heißt es, noch eine Minute. Sie sind ja auch eine Minute zu früh. Ach bitte, noch 10 Minuten, ein Sensor ist ausgefallen. Meine Frage, was macht Sie so hoffnungsfroh, daß Sie dieses in 10 Minuten beseitigen können? Der Glaube an die Stabilität des Systems! 5 Minuten später, vor 10 Uhr können wir keinesfalls anfangen. Ich lasse mir einen neuen Termin geben. Zur Sicherheit sollten Sie heute nachmittag um 15.30 Uhr wiederkommen. Das sind doch Unglückswürmer beim Stand der GMD, bin ich mal beeindruckt von den Menschenschlangen vor ihrem Exponat, dann fällt es sicher aus, wenn ich da bin. Später, am Nachmittag sehe ich das Ganze doch, virtual reality vom Feinsten.

Um 13 Uhr an meinem letzten Tag, dem Sonntag, hetze ich dann zu etwas ganz anderem, zur Pressekonferenz mit Dr. Ludewig, dem Beauftragten der Bundesregierung für den Aufbau Ost, der als Präsident zur Deutschen Bundesbahn wechselt. Ein interessantes Ereignis. Ein Politiker mit klaren, prägnanten und präzisen Vorstellungen, der diese auch äußern kann, der nicht die Souffleurdienste seiner Untergebenen braucht. Richtig was Neues hört man zwar nicht, aber man hat den Eindruck, die Dinge sind in guten Händen und sieht der Zukunft der Deutschen Bundesbahn zuversichtlich entgegen. Störend nur manche der kleinkarierten Fragen aus dem Journalistentroß, die dann in eine etwas ermüdende Diskussion über die Zukunft der Fahrradwege in Mecklenburg und auf der Insel Usedom endet.

Progress Report on Parallelism in *MuPAD*

Christian Heckler, Torsten Metzner, Paul Zimmermann
{chh,tom}@uni-paderborn.de, Universität Paderborn; zimmerma@loria.fr, INRIA Lorraine

MuPAD [8] is a general purpose computer algebra system with two programming concepts for parallel processing: micro-parallelism *for shared-memory machines and* macro-parallelism *for distributed architectures. This article describes language instructions for both concepts, the current state of implementation, together with some examples.*

Introduction

Most parallel computer algebra softwares (see [13] for a review) are either extensions of existing systems (ALTS and PACLIB for the SACLIB library, PARSAC-2 for SAC-2, ||MAPLE|| and Sugarbush for Maple, RR for Reduce), or systems specialized for some kinds of operations (PAC for linear algebra), or only tools for distributed computing, without any symbolic manipulation system (DSC for C or Lisp programs). On the contrary, *MuPAD* is a general purpose system whose language contains parallel instructions since the very beginning (see [6, pages 169–181] and [7, pages 160–161]). However, until now no parallel version was officially released (the last version, *MuPAD* 1.3, was released for sequential machines in December 1996), due to lack of access to computers with enough processors and efficient threads [16]. The situation changed in 1996, where an access was given to the Power Challenge Array (PCA) from the Centre Charles Hermite at INRIA-Lorraine (Nancy, France). This machine has 4 clusters with up to 18 processors each (see Figure 1), i.e. it is exactly the architecture for which *MuPAD* was designed for (interesting enough, at the time the *MuPAD* language was designed, i.e. around 1989, such machines did not exist!). Up to now, prototype shared-memory versions of *MuPAD* were ported on the following architectures: Sun Sparc [16], Sequent Symmetry [17], Convex SPP 1600-16 and Silicon Graphics Power Challenge.

The paper is organized as follows. Section describes the current state of the shared-memory version of *MuPAD* , and shows a real example with the PCA prototype version. Section describes the language constructs of the distributed version, and gives some hints how it will be implemented.

Micro-parallelism

In this section, we first describe the syntax of *MuPAD* parallel statements for shared-memory machines, then how these statements are implemented, and give some timings for polynomial factorization.

Language instructions

Two kinds of parallel constructs are available on shared-memory computers: parallel `for`-loops and parallel blocks. Parallel `for`-loops are exactly like sequential loops, except the keyword `do` is substituted by `parallel`:

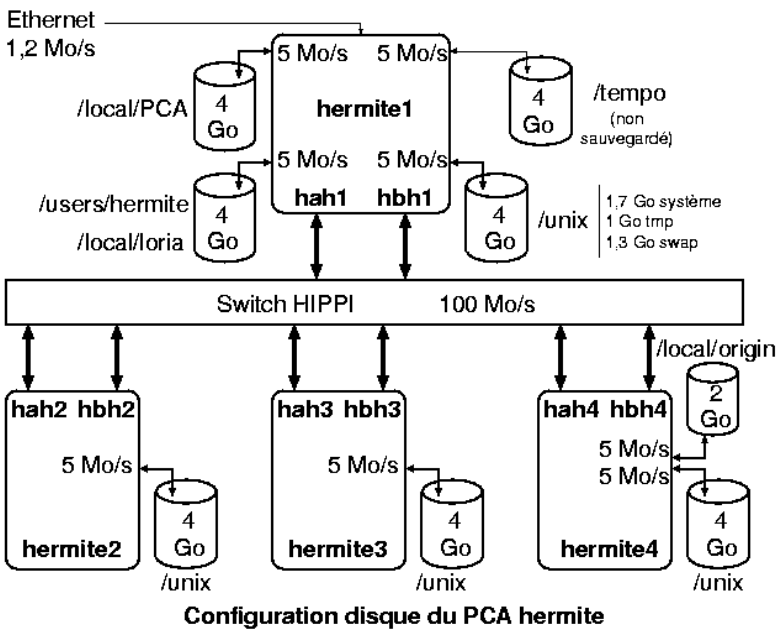


Figure 1: The Power Challenge Array architecture

for i **from** start **to** end **parallel**
  private j,k;
  stmts
end_for

The **private** keyword enables the user to declare auxiliary variables, which are local to each processor. Parallel blocks are defined as follows:

parbegin
  private j,k;
  $stmt_1$
  ⋮
  $stmt_n$
end_par

where the statements $stmt_1$ to $stmt_n$ are executed in parallel. One of these statements may itself consist of sequential instructions, in which case the **seqbegin** construct is needed:

seqbegin
  $stmt_1$
  ⋮
  $stmt_n$
end_seq

These three constructs (**parbegin**, **seqbegin** and parallel **for**-loop) are the only parallel constructs[1] available in the micro-parallel (shared-memory) version of *MuPAD* . They are enough to write some nice parallel programs, as we will see through polynomial factorization over finite fields.

Besides these constructs, since *MuPAD* version 1.3, *semaphores* and *locks* are available. A semaphore can be used to guard a resource with n instances. `sem:=Semaphore::sinit(n)` creates a semaphore `sem` and initializes it with n instances. `Semaphore::P(sem)` locks one instance of the desired resource, i.e. decrements the semaphore `sem`. The task blocks if the desired resource is not available, i.e. waits until the semaphore is greater than zero. `Semaphore::V(sem)` unlocks one instance of the desired resource, i.e. increments the semaphore. `Semaphore::N(sem)` returns the number of instances guarded by the semaphore `sem`.

A lock is a binary semaphore, the simplest type of a semaphore. `l:=Lock::linit()` initializes a lock `l`, `Lock::lock(l)` locks it, and `Lock::unlock(l)` unlocks it. For example a lock can be used to prevent multiple tasks from executing simultaneously a given section of a program (*critical section*), e.g. an access to a shared variable; the section begins with a lock operation and ends with an unlock operation.

Implementation

The implementation of the micro-parallelism concept is based on the memory management system MAMMUT [15]. MAMMUT offers a virtual shared memory machine, i.e. the programmer sees a set of parallel running MuPAD kernels sharing a common memory. In the shared memory, the stacks of the shared MuPAD variables and the *problem heap* of parallel executable MuPAD tasks are stored.

On Unix platforms the current version of MAMMUT works as follows. At the beginning the main MuPAD kernel process allocates shared memory using the Unix system call `mmap`. Shared variables are realized with pointers into that shared memory. After initializing some shared variables, parallel processes (also executing the MuPAD kernel) are created with a `fork` system call.[2] Thereby the number of processes is defined in the initialization file `.MMMinit`. The scheduling of the processes on the available processors is managed by the operating system.

In order to do some synchronisation between the processes a lock mechanism is necessary. For example the access to the stacks of the MuPAD variables in the shared memory must be coordinated. MAMMUT offers the possibility of locking a MAMMUT object, by setting its *lock bit*. The operation of testing and setting the lock bit must be protected using the operating systems lock (for example using the system functions `ussetlock()` and `usunsetlock()` on SGI machines), in order to prevent two processes from setting the lock bit of the same object simultaneously.

As mentioned above, MuPAD tasks that can be done in parallel are stored in a *problem heap* in the shared memory. When a process has nothing to compute it takes a task out of the heap. If there is no task in the problem heap the process goes to sleep and wakes up by a signal when new tasks come into the problem heap.

The garbage collection of MAMMUT is based on the principle of reference counting. Each object has a counter for the number of pointers pointing to it. If that counter is equal to zero the memory of the object can be deallocated. This method enables an *unique data representation*. Besides saving much memory this technique speeds up recognizing equal data, because equality can be decided by one pointer comparison. However, the frequently executed operation of changing and reading out the reference counter must be protected by a lock, which may be time consuming on some machines. This is the reason why the prototype parallel version for the Sun Sparc multiprocessor architecture gives no significant speedup [16]. Therefore, it is planned to develop a *parallel* mostly copying algorithm. The mostly copying garbage collection developed by Bartlett [4, 5] (see also [1]) is a variant

[1] These constructs are also available in the sequential version, where parallel tasks are processed in a random order. This enables the user to test parallel programs on a sequential machine.

[2] *Threads* could also be used in order to implement parallel running MuPAD kernels. However, when implementing the first parallel version no implementation of threads was available.

n	seq.	1 pr.	2 pr.	4 pr.	8 pr.	16 pr.	32 pr.
500	60s	124s	121s	91s	73s	49s	34s
1000	672s	851s	534s	445s	297s	191s	141s

Figure 2: Time needed for squaring a polynomial of degree n with coefficients of n bits.

of the classical *stop-and-copy* algorithm, where all accessible objects on the heap are copied into a new space. The advantages of this strategy are that it results in memory compaction and its running time is proportional to the amount of accessible memory. However, at the beginning an initial set of pointers to objects in the heap (the *root set*) must be known. Using the mostly copying strategy only a set of all possible pointers (the program stack and the registers) must be known, not the exact roots. It is planned to make a stable shared-memory port available for the public at the end of this year.

An example: polynomial factorization

Factorization of polynomials plays a central rôle in computer algebra, and in particular with coefficients in a finite field, since the factorization of polynomials with integer coefficients — either with the algorithm of Zassenhaus or that of Lenstra-Lenstra-Lovácz — begins with a factorization over a finite field [9]. For factoring polynomials of large degree n over finite fields, the best algorithm is currently that of Shoup [19], since it saves a factor of about $O(\sqrt{n})$ in both time and space with respect to Berlekamp's *big prime* algorithm [9].

One of the key operations in Shoup's algorithm is polynomial multiplication: indeed, the first step of the algorithm — computing $x^p \mod f$ where f is the input polynomial, which requires $O(\log p)$ multiplications of polynomials of the size of f — takes a constant part of the total time. More precisely, computing $x^p \mod f$ requires $\log p$ squarings and about $\frac{1}{2}\log p$ multiplications for an *average* p. Therefore we will focus on the squaring operation. The following *MuPAD* function computes the square of a polynomial a — with any kind of coefficients — using a parallel **for**-loop.

```
square := proc(a) local d,i,x,n;
begin
  d:=degree(a); x:=op(a,[2,1]);
  poly(_plus((for n from 0 to 2*d parallel
    private i,s;
    s:=(if n mod 2=0 then coeff(a,n/2)^2 else 0 end_if);
    for i from max(0,n-d) to min(d,iquo(n-1,2)) do
      s:=s+2*coeff(a,i)*coeff(a,n-i)
    end_for;
    s*x^n
  end_for)),op(a,2..3))
end_proc:
```

Each parallel task computes one term $s_n x^n$ of a^2; the parallel **for** loop returns the sequence of these terms, which is then transformed into a sum using `_plus`, and into a polynomial with `poly`.

Figure 2 shows the time needed for squaring a dense polynomial of degree n over a finite field F_p with p a prime of n bits, for $n = 500$ and $n = 1000$, on the Origin 2000 (32 processors R10000) of the Centre Charles Hermite. The timings for 1 to 32 processors were obtained with the prototype parallel version of *MuPAD* for Silicon Graphics machines, whereas the timings in the seq. column correspond to the standard sequential version, available on all *MuPAD* ftp sites.

Two comments can be made about these timings. Firstly, the one-processor parallel version runs much slower than the sequential one, especially for $n = 500$ (the cost for the lock operations may explain that). Secondly, the speed-up does not grow linearly with the number of processors. This may be due to a cache phenomena, since the parallel squaring function performs more concurrent memory accesses. Nevertheless, with 32 processors, a factor of about 5 can be saved for squaring degree 1000 polynomials, whence one can expect a similar speedup for polynomial factorization.

Macro-parallelism

Figure 3 explains the concepts of micro-parallelism and macro-parallelism in *MuPAD* . Each cluster is made of several processors working on a shared memory (micro-parallelism). The clusters can communicate by a kind of message-passing (macro-parallelism).

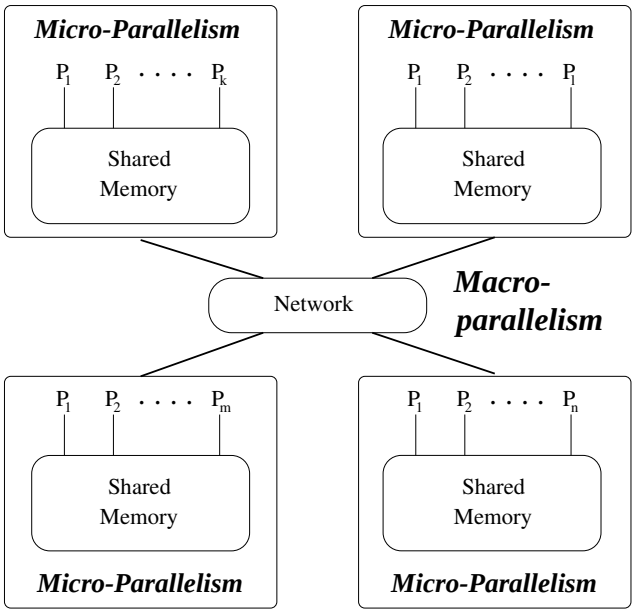


Figure 3: Micro-parallelism and macro-parallelism.

Language constructs

Due to the fact that we have concentrated our efforts on the (shared-memory) parallel version and other aspects of *MuPAD* there is no distributed (in the sense of message-passing) *MuPAD* version available so far.

From the viewpoint of macro-parallelism *MuPAD* consists of a set of independent clusters which are numbered from 1 to n. Each cluster in turn is formed of a number of running *MuPAD* kernels and micro-parallelism is (or can be) used inside every cluster. The *MuPAD* language provides to the user a special concept to program in this distributed environment. However the current user interface is not very pleasant to use. The interface consists of net variables as well as queues and pipes between different clusters.

The *net variables* are common to all clusters, so they can be considered as shared memory between the different clusters (i.e. global variables):

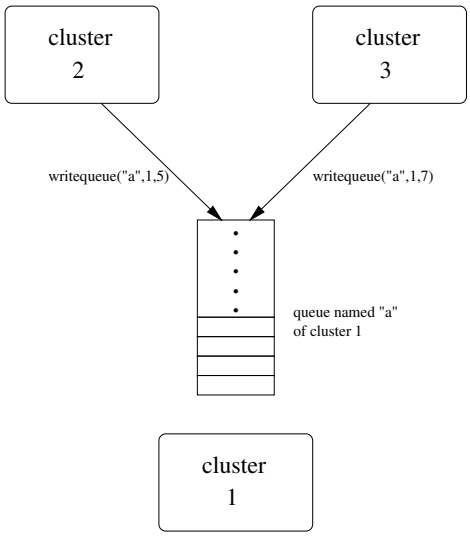


Figure 4: Queues

global(a)
global(a, b)

In the first form the net variable `a` is read out. Here `a` is considered to be a variable name so it is not evaluated. The result of the read operation is then evaluated. In the second form the value of `b` is assigned to `a`. Here again `a` is not evaluated and `b` evaluates as usual. During the evaluation of `b`, no other task can change the net variable `a`. Assigning a value to a net variable is therefore an atomic operation.

Each cluster can have any number of queues, and a cluster can write in any queue (see figure 4):

writequeue(a, i, value)

Here `value` is the expression which will be written to the queue `a` of the cluster `i`.

A cluster can only read the first element of its queue. There exists a blocking and a non-blocking read action on a queue:

readqueue(a)
readqueue(a, Block)

Here `a` is the name of the queue the cluster will read from. If the queue is empty the function call `readqueue()` will return a *MuPAD* object of type DOM_NULL. In the second case the function call `readqueue()` will block until there is a value available in the queue.

There exists one special queue named ``` ``work'' ``` in every cluster. After the system is initialized every cluster checks its queue with the name ``` ``work'' ``` periodically when it has no work to do. If the queue contains an expression, this one is read and evaluated. An exception is the cluster 1, which never checks its ``` ``work'' ``` queue, but communicates with the user and gets tasks in this way.

With queues it is possible to send an expression to every cluster, but it is not guaranteed that only one cluster writes in the queue of another cluster. So the cluster that reads from its queue cannot be sure that the value is sent from

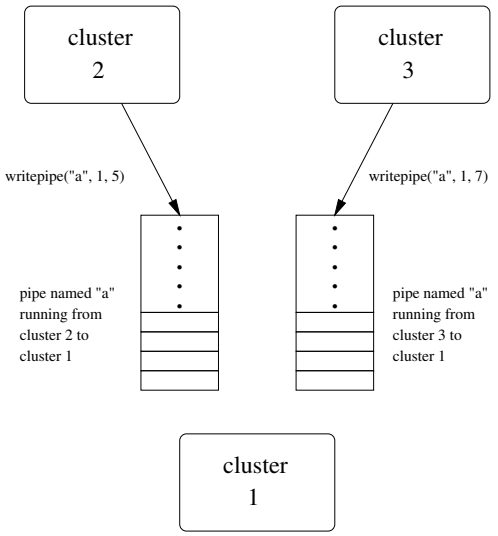


Figure 5: Pipes

a specific cluster it wants to receive an expression from. So there exists another data structure named *pipe*, which is similar to queues (see figure 5). For two clusters i and j there exists exactly one pipe of name `a` that runs from cluster i to cluster j. A pipe `a` running from cluster i to cluster j and a pipe `a` running from cluster j to cluster i are two different pipes:

writepipe(a, j, value)

Here the expression `value` is written in the pipe `a` of cluster `j`. If this instruction is performed by cluster `i`, it creates a pipe `a` which runs from cluster `i` to cluster `j`.

readpipe(a, i)
readpipe(a, i, Block)

With the first `readpipe()` call, a cluster reads an expression from the pipe `a` of cluster `i`. The second call is the blocking variant of the function call `readpipe()`.

The function `topology` enables the user or programmer to know the topology of the distributed system: `topology()` returns the number of clusters in the system; `topology(0)` returns the number of *MuPAD* kernels, i.e. the sum of the *MuPAD* kernels of individual clusters; `topology(n)`, where `n` is a positive integer between `1` and `topology()`, returns the number of *MuPAD* kernels in cluster `n`; finally `topology(Cluster)` returns the index of the current cluster.

Implementation

To implement the macro-parallelism it is necessary to transfer every *MuPAD* object in an efficient way. For this reason a binary encoding was chosen, following [3, 11]. Note that every *MuPAD* procedure is a regular *MuPAD* object. So this work was done for every *MuPAD* object. Internally a *MuPAD* object is represented by a directed acyclic graph (in some very special cases it is a cyclic structure), so common subobjects sharing should be also used. This is done in the obvious way. Every object is marked with a label. When a subobject occurs more than

once in a *MuPAD* object only the first occurrence will be transferred to another cluster; if the subobject occurs a second time the label is sent as a reference. Here we use similar techniques like MP [11] and OpenMath [21] but in contrast to them it is also possible to send cyclic structures. As a result, we got an efficient encoding from a *MuPAD* object to a well-defined byte stream of elementary C data types. So the basics for implementing the concepts of queues and pipes in an efficient way are ready. Now we can use a transport protocol like PVM [10], MPI [12] or MP to send *MuPAD* objects between clusters. For the first prototype of a distributed *MuPAD* version which implements exactly the language constructs of the macro-parallelism described in section , we have chosen MPI because it is widely available. To integrate the transport layer of the distributed version, the concept of dynamic modules [20] is used.

It still remains the question how net variables could be implemented in an efficient way. For this purpose, it is planned to use *DIVA* (The **DI**stributed **VA**riables Library[3]), which is part of the A2 project of the SFB 376.

Future developments

Besides the work described in section , an asynchronous object oriented communication protocol named IPP is developed and partly implemented. This protocol has inherited a lot of ideas of PVM and the UNIX I/O concept. On the one hand a lot of ideas of PVM, because PVM is a distributed system and *MuPAD* is also a distributed (computer algebra) system. So the concept of dynamic host configuration and the creation of new processes are undertaken. On the other hand the concept of file descriptors in an UNIX environment is elegant. Once one has opened a file the interaction with a file, a network device or a screen is similar from a programmer's point of view. So we decided that this should be also possible in IPP. The main communication object in IPP is the *channel*. A channel represents a bidirectional communication connection. After one has opened a channel between two (or more) processes, a process and a file, or between two sockets, there is no difference in using this channel. One can send and receive data with simple function calls like `get()` and `put()`. One special, *MuPAD* specific aspect of the protocol is that it can transfer the data type *S_Pointer* which is the main *MuPAD* data type. Internally every *MuPAD* object is represented by an S_Pointer. When the implementation of the protocol is finished, it could be used to implement a better user interface to the macro-parallelism: for example to start a specific *MuPAD* procedure on some of the available clusters or on some given cluster, adding a cluster to the system, making the communication between clusters easier and so on. At the moment PVM is used to implement IPP, because it is widely available and supports process creation and dynamic system configuration.

Future Work

As already said in section , the first priority is to make available to the whole scientific community a (shared-memory) parallel version of *MuPAD* version 1.3, and to put that version on the *MuPAD* ftp servers.

Then a second objective will be to develop some nice applications and libraries for this parallel version. Good candidates are the algorithms presented at the first Pasco conference, for example the modular algorithm for sparse multivariate polynomial interpolation [14], parallel Buchberger algorithms [2], and more generally all algorithms on polynomials presented in Paul Wang's overview [22]. At the University of Paderborn, Christopher Creutzig is currently developing a general framework for constraint logic programming using parallel instructions, which can be used in turn in a parallel computation of Gröbner bases.

For the distributed version, we plan to have a first prototype version implementing the macro-parallel language at the end of this year.

[3] `http://www.uni-paderborn.de/sfb376/projects/a2/TP_A2_DIVA.html`

Acknowledgements. Work on the parallel versions of *MuPAD* is supported by the SFB project 376 *Massive Parallelism*[4] and by the European Procope project *Design and Implementation of Efficient Algorithms for Computer Algebra*[5]. For providing us an access to parallel computers, we would like to thank: the Centre Charles Hermite[6] (Power Challenge with 18 processors and Origin 2000 with 32 processors), especially Alain Filbois and Olivier Coulaud; Ulrich Schwardmann from the GWD in Göttingen (Silicon Graphics computer with 4 processors); Thomas Schramm and Peter Junglas from the University of Hamburg-Harburg (Convex with 16 processors).

References

[1] ATTARDI, G., AND FLAGELLA, T. Customising object allocation. In *ECOOP '94 – Object-Oriented Programming* (Berlin, July 1994), M. Tokoro and R. Pareschi, Eds., vol. 821 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 299–319.

[2] ATTARDI, G., AND TRAVERSO, C. Strategy-accurate parallel Buchberger algorithms. *Journal of Symbolic Computation 21* (1996), 411–425.

[3] BACHMANN, O., SCHÖNEMANN, H., AND GRAY, S. MPP: A Framework for Distributed Polynomial Computations. In *Proceedings of the 1996 International Symposium on Symbolic and Algebraic Computation: ISSAC'96* (1996), L. Y.N., Ed., ACM Press, pp. 103–112.

[4] BARTLETT, J. F. Compacting garbage collection with ambiguous roots. Tech. Rep. 88/2, DEC Western Research Laboratory, Palo Alto, CA, Feb. 1988.

[5] BARTLETT, J. F. Mostly-copying garbage collection picks up generations and C++. Technical Note TN-12, DEC Western Research Laboratory, Palo Alto, CA, 1989.

[6] FUCHSSTEINER, B., AND AL. *MuPAD Benutzerhandbuch*. Birkhäuser, Basel, 1993. In german.

[7] FUCHSSTEINER, B., AND AL. *MuPAD Tutorial*. Birkhäuser, Basel, 1994.

[8] FUCHSSTEINER, B., AND AL. *MuPAD User's Manual*. Wiley Ltd., 1996.

[9] GEDDES, K. O., CZAPOR, S. R., AND LABAHN, G. *Algorithms for computer algebra*. Kluwer Academic Publishers, 1992.

[10] GEIST, A., BEGUELIN, A., DONGARRA, J., JIANG, W., MANCHEK, R., AND SUNDERAM, V. *PVM: Parallel Virtual Machine – A User's Guide and Tutorial for Networked Parallel Computing*. MIT Press, 1994.

[11] GRAY, S., KAJLER, N., AND WANG, P. S. Design and Implementation of MP, a Protocol for Efficient Exchange of Mathematical Expressions. *Journal of Symbolic Computation*. To appear.

[12] MESSAGE PASSING INTERFACE FORUM. MPI: A Message-Passing Interface Standard. Tech. rep., University of Tennessee, Knoxville, Tennessee, June 1995. Available from `ftp.mcs.anl.gov` in `pub/mpi/mpi-1.jun95/mpi-report.ps`.

[13] MORISSE, K. Parallele Computeralgebra-Systeme. *mathPAD 4*, 3 (Dec. 1994), 13–19. In german. URL `http://math-www.uni-paderborn.de/ cube/MATHPADS`.

[14] MURAO, H., AND FUJISE, T. Modular algorithm for sparse multivariate polynomial interpolation and its parallel implementation. *Journal of Symbolic Computation 21* (1996), 377–396.

[4] Cf `http://www.uni-paderborn.de/SFB376`
[5] See `http://www.loria.fr/ zimmerma/mupad/procope.html`
[6] Cf `http://www.loria.fr/CCH`

[15] NAUNDORF, H. *MAMMUT – Eine verteilte Speicherverwaltung für symbolische Manipulation*. MuPAD Reports. Teubner, Stuttgart. In german. To appear.

[16] NAUNDORF, H. Threads unter Solaris. *mathPAD 4*, 3 (Dec. 1994), 20–23. In german. URL `http://math-www.uni-paderborn.de/ cube/MATHPADS`.

[17] NAUNDORF, H. Parallelism in MuPAD . In *Electronic Proceedings of the 1st International IMACS Conference on Applications of Computer Algebra* (may 1995), M. Wester, S. Steinberg, and M. Jahn, Eds., http://math.unm.edu/ACA/Proceedings/MainPage.html.

[18] NAUNDORF, H. *Ein denotationales Modell für parallele objektbasierte Systeme*. MuPAD Reports. Teubner, Stuttgart, Dec. 1996. PhD Thesis. In german.

[19] SHOUP, V. A new polynomial factorization algorithm and its implementation. *Journal of Symbolic Computation 20*, 4 (Oct. 1995), 363–397.

[20] SORGATZ, A. *Dynamische Module*. MuPAD Reports. Teubner, Stuttgart, 1996. In german.

[21] VORKÖTTER, S. Proposed OpenMath Specification. URL `http://www.can.nl/ abbott/OpenMath`, July 1995.

[22] WANG, P. S. Parallel polynomial operations on SMPs: an overview. *Journal of Symbolic Computation 21* (1996), 397–410.

Dynamic Modules – C++ Kernel Extensions

Andreas Sorgatz
Universität-GH Paderborn, *andi@mupad.de*

Software integration gains more and more relevance for computer algebra systems (CAS): Users write or copy algorithms from the Internet and want to use them in their favourite CAS. Dynamic modules allow the user to integrate C/C++ routines as well as complete software packages into a CAS and to use them like usual CAS functions. The variety of different packages and algorithms which can be used simultaneously within a CAS increases drastically. In this sense dynamic modules apply the principle of software integration to the broad field of mathematical applications.

The Usual Way

So far, the general concept to integrate software into a CAS is to use IPC protocols like MP [4] or MATHLINK [6] which usually base on socket or file communication. They are useful especially to communicate with tools running on different hosts in a network. However, they are a bit clumsy and require a lot of communication overhead. Using such protocols the following actions have to be carried out to pass data between processes or to call an external function of a partner process:

1. The CAS data are converted into data packages (*information encoding*) and
2. transfered (*copied physically*) between the processes.
3. The data packages are reconverted into CAS data or any other representation used by the communication partner (*information decoding*).

Information en- and decoding may be costly. Furthermore passing information doubles the amount of memory space because data are copied and stored twice. Especially in computer algebra, where symbolic computations often require much memory (*intermediate data swell*), this may lead to problems. Last not least, partner processes cannot call internal CAS routines on a C/C++ language level in an efficient way.

Problems and Missing Features of IPC

Looking at some applications of CAS, limitations and problems of conventional CAS and IPC protocols become visible. For example:

Arbitrary Precision Arithmetic: The general way of writing fast algorithms using multi precision numbers is to install a package like GMP [8] or PARI [9] and to implement C/C++ routines -utilizing this package- as well as a simple user interface. But what about using these algorithms and their results in the user's favourite CAS? For this, one has to use either copy & paste, to write and read files or to use an IPC protocol. A more efficient and convenient way is to provide an interface to the internal arithmetic of a CAS on a C/C++ language level, allowing users to integrate their algorithms into the CAS and to use them like usual CAS functions.

Efficient Call of External Functions: Users may also want to integrate very efficient algorithms with short execution times into the CAS, to be used frequently by higher level algorithms (e.g. the summation of polynomials over a residue class ring F_p (MAGNUM [2], GF2FAC [10]). Using an IPC protocol here may dramatically reduce the efficiency of the high-level algorithm.

Support of Various IPC Protocols: Even though IPC protocols are still important for many applications there are some more questions to consider: Which protocols are supported by the user's favourite CAS? What about a new protocol or one implemented by a special interest group? Will it be supported by the CAS in the future? How long the user has to wait for it? How big will be the CAS kernel that supports all protocols needed by different user groups? E.g. the machine code of an MP protocol needs at least 300Kb. In comparison to this, the complete *MuPAD* kernel is about 1200Kb large. That means: including only this IPC protocol statically will encrease the CAS kernel size by more than 25%. Even if it is never used by

the user! Therefore, a more flexible and dynamically concept should be used which allows both developers and users(!) to extend the CAS without changing any kernel sources. Such packages should be available as add-ons, loaded into the CAS at run-time on demand only.

Kernel Extensions: Usual C/C++ based CAS are black boxes in the sense that they cannot be extended by the user. Here, users can only write library functions based on the CAS language. Providing an interface to the internal methods of the CAS kernel would enable users to implement much more efficient C/C++ algorithms and to integrate special purpose functions.

Dynamic Modules in *MuPAD*

From the user's point of view, a dynamic module is similar to a library package (figure 3). It contains so-called *module functions* which are, in contrast to *library functions*, not written in the *MuPAD* programming language but are compiled machine code functions written in C/C++, just like the *built-in functions* of the kernel. Programming of dynamic modules is facilitated by the *MuPAD Application Programming Interface* (MAPI).

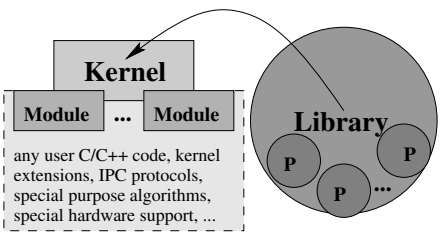


Figure 6: Using the Library and Dynamic Modules

From a technical point of view, a dynamic module is a special kind of a machine code library (on most systems implemented as a *shared library*) that can be linked at run-time. Its module functions are made public to the interpreter and propagated to the user like usual *MuPAD* functions. The machine code of dynamic modules resides within the process environment of the *MuPAD* kernel and can directly access internal kernel methods.

Dynamic modules can be unloaded at any time. They are automatically re-*loaded on demand*. Unloading a module means to unlink its machine code from the kernel and to remove it from the main memory. A technical description of the CAS independent concept as well as its implementation for *MuPAD* (Rel. 1.2.2) is given in *Dynamische Module* [11].

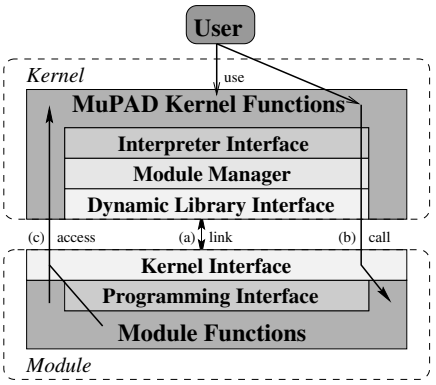


Figure 7: Scheme of the Module Management

Advantages and Features

The concept of dynamic modules is not a replacement for IPC, but it extends the user's possiblities drastically:

Flexibility: Modules are written in C/C++ with no restrictions due to the CAS language. The fact that C is widely used and many software packages are written in it, makes it easy to integrate them as dynamic modules.

Efficiency: Module functions are as fast as built-in functions. Data exchange between the kernel and modules is done by simply passing references (C/C++ pointers). This is the fastest method possible, because no data must be encoded or copied. No further communication overhead is needed and time and memory usage is independent from the size of the data to be exchanged.

Dis-/Replacement: Modules can be displaced automatically and transparently to the user by using various strategies. E.g. *aging*: inactive modules are displaced after a specified time in order to save memory resources.

A Modular Kernel: The kernel can be split into several modules which are loaded on demand only. This leads to a smaller and freely configurable kernel that needs less resources and is easier to maintain. Especially IPC protocols and special purpose functions are good candidates to be integrated in this way. Furthermore, dynamic modules can be added and maintained by users and special interest groups independently from the CAS developers. There is no need to change any kernel sources, which are in fact not public for most systems.

An Open System: Developers of dynamic modules can directly access all (documented) parts of the kernel, e.g. the arithmetic, the memory management, the io-system and further more. It is also possible to call library functions. This enables the user to extend the kernel by nearly any feature that is desired.

A Universal Shell: Dynamic modules offer a new view on a CAS. On the one hand it works in the conventional way, providing its sophisticated capabilities of doing symbolic computation and on the other hand it can be used as a powerful and flexible *universal mathematical shell* to integrate, utilize or test various software packages in order to use them all together to solve mathematical problems. E.g. for any kind of engineering it is useful to integrate very efficient numerical algorithms like the NAGC library (refer to the example below) to combine the sophisticated capabilities of symbolic computation with the strength of a numerical package.

How to Use Dynamic Modules

After loading a dynamic module -by using the *MuPAD* function `module()`- it can be used exactly like any *MuPAD* library package. Additionally, dynamic modules can be displaced. They are automatically re-*loaded on demand* as demonstrated in figure 3.

```
>> module("demo");
                        demo
>> info(demo):

Module: 'demo' created on 03.Sep.97
A date function

Interface:
demo::date, demo::doc

>> demo::date();

          "Wed Sep  3 19:21:48 MET 1997"

>> module::displace(demo):
>> export(demo):
>> date();

          "Wed Sep  3 19:22:03 MET 1997"
```

Figure 8: Using a Dynamic Module in *MuPAD*

Writing dynamic modules for *MuPAD* is quite easy. Figure 4 lists the complete source code of the module `demo`.

```
// demo.C -- a very simple dynamic module
MFUNC( date, MCnop )
{
    MFnargsCheck( 0 );

    time_t clock;
    time( &clock );

    char* cstrg = ctime( &clock );
    cstrg[24] = '\0';

    MFreturn( MFstring(cstrg) );
} MFEND
```

Figure 9: Source Code of the Dynamic Module `demo`

All keywords starting with the prefix `MC`, `MF` and `MT` belongs to the *MuPAD Application Programming Interface* (MAPI). Any other code is plain C/C++.

How to Create Dynamic Modules

To create an executable module, the so-called *module generator* (`mmg`) is used. It is started with the name of the source file to be processed as its argument (figure 5) and places the output file `demo.mdm` in the current directory. Option `-v` (verbose) tells `mmg` to log all actions that are carried out and `-V` (Version) includes a version information string (refer to `info()`, figure 3).

```
andi> mmg -v -V 'A date function' demo.C

MMG -- MuPAD-Module-Generator -- V-1.4.0
Mesg.: Scanning source file ...
Mesg.: 1 function(s) found in 17 lines
Mesg.: Creating  extended module code ...
Mesg.: Compiling extended module code ...

CC -pic -G -DSOLARIS -c MMGdemo.C -o demo.o
   -DLONG_IS_32BIT -DPOINTER_IS_32BIT
   -I/usr/MuPAD/share/mmg/include/kernel
   -I/usr/MuPAD/share/mmg/include/pari
   -I/use/MuPAD/share/mmg/include/mmt

Mesg.: Linking dynamic module ...

CC -pic -G -lC -o demo.mdm demo.o
   -L/usr/MuPAD/solaris/lib

Mesg.: Ok
```

Figure 10: Creating the Dynamic Module `demo`

The module generator adds some administrative information for the module manager and compiles and links

the extended source code to a special kind of a machine code library that can be used within *MuPAD* instantly. For this, a standard C++ compiler (the GNU g++ is also supported) and a standard linker are used.

Integration of MAGNUM in *MuPAD*

MAGNUM [2] provides fast factorization algorithms for univariate polynomials over a residue class ring F_p, with p is a prime less than 2^{16}. Figure 6 demonstrates the usage of the dynamic module `magnum`. The function `magnum::doc` was created by the module generator and allows to display plain text online documentation, placed in the file `magnum.mdh` in the same directory as the module.

```
>> module("magnum"):
>> P:= poly( x^200+x+1, [x], IntMod(65437) ):
>> magnum::irreducible(P);
                                FALSE
>> time( magnum::factor(P) );
                                30470
>> magnum::doc("factor"):

NAME
    factor - Factorizes a polynomial

SYNOPSIS
    magnum::factor( poly ) ...
```

Figure 11: Using `magnum` in *MuPAD*

The *MuPAD* interface of the magnum module needs less than 180 lines of C++ code. The complete code including help pages and a detailed description is available at the *MuPAD* web [1] . . . `/PAPERS/MODULES/MAGNUM/`.

Using the NAGC Library within *MuPAD*

The NAGC library [3] is a widely known package for numeric calculations. Using dynamic modules, one can combine the power of NAGC (numerical calculations) with the strength of *MuPAD* (symbolic computations) to solve mathematical problems. *MuPAD* acts as a flexible and comfortable interface to user-defined algorithms utilizing NAGC. Their integration is facilitated by the header file `mnag.h` which contains some conversion routines for basic NAGC data types and supports a *MuPAD* like error handling. Figure 7 demonstrates a module which utilizes the NAGC routine `c02afc` to find all roots of a complex polynomial. The file `mnag.h` and further information are available at the *MuPAD* web site [1] . . . `/PAPERS/MODULES/NAGC/`.

```
>> module("nagc"):
>> nagc::roots( poly(2*x^3+5*x^2+3) );
   [- 2.7050011710 + 5.551115123e-17 I,
      0.1025005859 - 0.7375784977 I,
      0.1025005859 + 0.7375784977 I]
```

Figure 12: Using NAGC to Find Complex Roots

Interprocess Communication Using MP

MP [4] is a protocol to exchange mathematical data binary encoded as attributed trees. A first integration of MP was done using dynamic modules. It supports the transfer of basic C/C++ data types and provides essential MP routines as *MuPAD* functions to allow prototyping of MP dictionaries on a *MuPAD* language level. Furthermore, the MPP [5] dictionary of MP is used to interact with the SINGULAR [7] system for accomplishing very fast Gröbner basis and polynomial computation. The progress of this project is documented at the *MuPAD* web site [1] . . . `/PAPERS/MODULES/SINGULAR/`.

```
>> loadlib("sing"):
>> Cycl:= Ideal(
   poly( a^5-1*b^5, [a,b,c,d,e] ),
   poly( b^5-1*c^5, [a,b,c,d,e] ),
   poly( c^5-1*d^5, [a,b,c,d,e] ),
   poly( d^5-1*e^5, [a,b,c,d,e] ),
   poly( a^4*b+b^4*c+c^4*d+d^4*e+a*e^4,
        [a,b,c,d,e] ) );
>> time(sing::std(Cycl));
```

Figure 13: Using MP to Interact with SINGULAR

Availability

With *MuPAD* 1.3 the concept of dynamic modules is available for most UNIX platforms, especially for Linux. With developers versions, also Windows 95/NT and the Apple Macintosh PowerPC are supported. The module generator `mmg` and the programming interface MAPI are part of the *MuPAD* distribution for UNIX systems. The programming interface (MAPI) is described in *Dynamic Modules – User's Manual and Programming Guide for MuPAD 1.4* [12].

It is the developer's matter of concern to provide *MuPAD* as an open system which can easily be integrated into the user's working environment and dynamic modules are the basic concept to interact with *Real-World-Applications* and *Problem-Solving-Environments*.

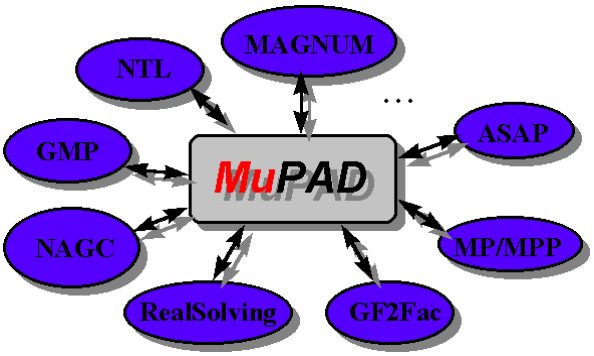


Figure 14: Connectivity to a broad field of applications

Projects

As part of the *Sonderforschungsbereich 376* [13] dynamic modules are used to integrate fast parallel algorithms for polynomials over residue class rings $F_p[X]$ (GF2FAC [10]) as well as the IPC protocol MP in order to interact within Problem-Solving-Environments.

Next steps will be a better support of the NAGC and MP interface and the further improvement of the programming interface MAPI. The future aim is to provide two programming levels for *MuPAD*: first, the easy to use and conventient *MuPAD* language and second, the sight on the *MuPAD* kernel as on a very efficient comuputer algebra C++ class library. Combining both techniques provides a flexible concept to implement complex as well as very efficient computer algebra algorithms.

External projects: Fabrice Rouiller at INRIA Lorraine (Nancy, France) is developing a *MuPAD* interface to his *RealSolving* package. Refer to his web pages at `http://www.loria.fr/~rouillie/software.html` to read the documentation and to copy binaries.

Further projects: It is planned to develope a *MuPAD* library package for commutative algebra and singularity theory, which bases on an MP link to SINGULAR. This will be done in cooperation with Olaf Bachmann and Hans Schönemann from the University of Kaiserslautern. An example of using a prototype of this library package is given in figure 8.

References

[1] MUPAD, Multi Processing Algebra Data tool, MuPAD Group, Automath, University of Paderborn, Web: `http://www.mupad.de`, Ftp: `ftp://ftp.mupad.de/MuPAD/`.

[2] MAGNUM, C++ Class Library for Polynomial Factorization, Wolfgang Roth, University of Mannheim, Web: `ftp://obelix.statistik.uni-mannheim.de/public/magnum/`.

[3] NAGC, The NAG C Library, The Numerical Algorithms Group Ltd., Oxford, UK, Web: `http://www.nag.co.uk:80/numeric/CL.html`.

[4] MP, A Protocol for Efficient Exchange of Mathematical Expressions, S. Gray, N. Kajler and P. S. Wang, In Proc. of the *ISSAC'94*, Oxford, GB, M. Giesbrecht, Ed., ACM Press, pp. 330-335.

[5] MPP, A Framework for Distributed Polynomial Computations, O. Bachman, H. Schönemann, S. Gray, In the Proc. of *ISSAC'96*, pp. 103-112.

[6] MATHLINK, Reference Guide, Ver. 2.2, WOLFRAM RESEARCH, INC., Mathematica Technical Report, 1993.

[7] SINGULAR, A System for Computation in Algebraic Geometry and Singularity Theory, G.-M. Greul, G. Pfister and H. Schönemann.

[8] GMP, GNU MP, A Library for Arbitrary Precision Arithmetic, Ftp: `ftp://ftp.fu-berlin.de/unix/gnu/gmp/`.

[9] PARI, *User's Guide to PARI - GP*, C. Batut, D. Bernardi, H. Cohen, and M. Olivier, Version 1.39, Jan. 1995, Ftp: `ftp://megrez.math.u-bordeaux.fr/pub/pari`.

[10] GF2FAC, Arithmetic and Factorization of Polynomials over F_2, J. von zur Gathen, J. Gerhard, In Proc. of the *ISSAC'96*, Zürich, Switzerland, ACM press, pp. 1-9.

[11] *Dynamische Module — Eine Verwaltung für Maschinencode-Objekte zur Steigerung der Effizienz und Flexibilität von Computeralgebra-Systemen*, A. Sorgatz, MuPAD Reports, B.G. Teubner Verlag, Stuttgart, Oct. 1996, Web: `http://www.mupad.de/BIB/sorgatz96.html`.

[12] *Dynamic Modules — User's Manual and Programming Guide for MuPAD 1.4*, A. Sorgatz, Universität-GH Paderborn, Preprint, Sep. 1997.

[13] SFB 376, Sonderforschungsbereich 376, Massive Parallelität, Algorithmen - Entwurfmethoden - Anwendungen, University of Paderborn, Germany `http://www.uni-paderborn.de/SFB376/`.

[14] *MuPAD - Dynamische Module, Spiel (fast) ohne Grenzen*, A. Sorgatz, Linux Magazin, Jan. 1997, `http://www.mupad.de/BIB/sorgatz97a.html`.

Lineare Optimierung in *MuPAD*

Meinolf Sellmann
Universität-GH Paderborn, *sello@uni-paderborn.de*

Im folgenden Artikel, der einen überarbeiteten Ausschnitt einer Diplomarbeit darstellt [1], sollen speziell die Kriterien zur Auswahl der Branchingschnitte bei die Entwicklung einer Library zur (gemischt-)ganzzahligen Optimierung beschrieben werden. Dazu wird zunächst ein Überblick über den Algorithmus von Dakin gegeben und dann die Problematik der Bestimmung günstiger Branchingvariablen erläutert. Es werden bekannte Strategien und eine Neuentwicklung, die sogenannten „Raumorientierten Strafkosten", vorgestellt und an einigen graphischen Beispielen veranschaulicht. Schließlich gibt eine Beispielsitzung einen Einblick in die Funktionalität der neuen Library.

Definitionen

Seien $m, k, l \in \mathbb{N}, A \in \mathbb{Q}^{m \times k}, B \in \mathbb{Q}^m, c \in \mathbb{Q}^k, d \in \mathbb{Q}^1$ und $M = \{\binom{x}{y} | Ax + By \leq b, x \in \mathbb{N}^k, y \in \mathbb{Q}_+\}$ Dann heißt

Maximiere $(c^T, d^T) \cdot \binom{x}{y} unter \binom{x}{y} \in M$

ein *gemischt-ganzzahliges Optimierungsproblem*.
m gibt dabei die Anzahl der Restriktionen, $n := k + l$ die Anzahl der Variablen an. M heißt die Menge der zulässigen Lösungen oder *zulässiger Bereich*.

- Für $k = 0$ sprechen wir auch von einem *linearen Optimierungsproblem*. Minimiere $c^T x$ unter $x \geq 0$ und $Ax \leq b$ nennen wir dann *Simplexform* eines linearen Optimierungsproblems.
- Für $l = 0$ heißt obiges Problem entsprechend *ganzzahliges Optimierungsproblem*.

Das Verfahren: Branch & Bound

Der Algorithmus von Dakin

Für die (gemischt-)ganzzahlige Optimierung ohne weitere Vorkenntnisse über das gegebene Problem zeitigt das Branch-and-Bound-Verfahren nach Dakin die besten Resultate und wird daher in kommerzieller Software fast ausschließlich benutzt, wie etwa in MOPS oder CPLEX. Dabei wird das gegebene Problem durch die abgeschwächte Forderung $x \in \mathbb{Q}_+^k$ relaxiert, wodurch ein Problem der linearen Programmierung entsteht, das etwa mittels des Simplex-Algorithmus gelöst werden kann. Man erhält eine obere Schranke $z^* = (c^T, d^T) \cdot \binom{x^*}{y^*}$ für ein gegebenes Teilproblem, mit Hilfe derer abgeschätzt wird, ob durch die Betrachtung eines Knotens im Branch-and-Bound-Baum noch eine Verbesserung zu einer bereits gefundenen zulässigen Lösung möglich ist. Falls nicht, wird dieses Teilproblem nicht weiter betrachtet. Andernfalls werden ausgehend von unserer relaxierten Lösung x^* zwei neue Teilprobleme dadurch generiert, daß für eine noch nicht ganzzahlige Komponente i von x^* mit $x_i^* = n_i + f_i; n_i \in \mathbb{N}, 0 \leq f_i \leq 1$ jeweils eine der beiden zusätzlichen Restriktionen $x_i \leq n_i$ oder $x_i \geq n_i + 1$ hinzugefügt wird. Bei diesem Vorgehen ergeben sich an zwei Stellen Wahlmöglichkeiten:

[1] Dabei sei stets $o \in \mathbb{N}$ und $0 \in \mathbb{Q}_+$.

1. Welche der nicht ganzzahligen Komponenten i von x^* soll als Ausgangspunkt für die Generierung der neuen Restriktionen gewählt, und

2. bei welchem Knoten im Branch-and-Bound-Baum soll mit der Berechnung fortgefahren werden?

Beide Entscheidungen, besonders aber die erste, beeinflussen stark das Laufzeitverhalten unseres Algorithmus, wie empirische Resultate zeigen. Ein wesentlicher Aspekt bei der Implementation eines Algorithmus zur Bestimmung (gemischt-)ganzzahliger Lösungen linearer Optimierungsaufgaben besteht deshalb darin, die verschiedenen Heuristiken zur Wahl der Branchingvariablen und zur Bestimmung des nächsten zu bearbeitenden Knotens gegeneinander abzuwägen und wenn möglich auch eigene, neue Ideen zu entwickeln.
Für einen Algorithmus, der in ein Computeralgebrasystem eingebettet ist, sollte stets berücksichtigt werden, daß es nicht darum geht, ein für große Probleme etwa aus der betriebswirtschaftlichen Praxis möglichst schnelles Programm zu schreiben, sondern daß die zu entwickelnde Library im Kontext eines Computeralgebrasystems hinreichend gute Resultate erzielen muß. So ist eine der wesentlichen Annahmen über die gegebenen Optimierungsprobleme, daß sie, unter anderem weil sie interaktiv eingegeben werden, vermutlich klein aber dafür recht dicht sein werden; unter einem „dichten“ Problem soll dabei eines verstanden werden, dessen Matrizen A und B nicht hauptsächlich aus Nullen bestehen (siehe Definitionen). ber die Struktur wird jedoch nicht spekuliert, um die Breite der auf das gemischt-ganzzahlige Optimieren reduzierbaren Probleme (wie etwa Maxflow, Knapsack oder TSP) nicht im vorhinein einzuschränken.
In diesem Artikel sollen nun die Aspekte zur Auswahl der Branchingvariablen näher beleuchtet werden: Um zu untermauern, daß die Entscheidung, welche Komponente des relaxierten Lösungsvektors zur Generierung der Branchingschnitte herangezogen wird, für die Effizienz des Algorithmus von Dakin von großer Bedeutung ist, betrachte zunächst folgendes

Beispiel

Maximiere $-x + 20y$ unter $x, y \in \mathbb{N}$ und $x + 7y \leq 20, -x + 3y \leq 5$.

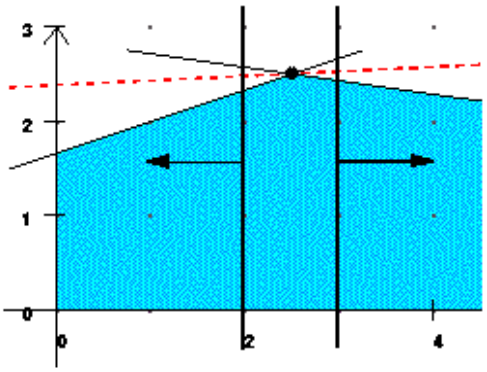


Abbildung 15: Graphische Darstellung des Beispiels.

Der Simplex-Algorithmus liefert für die Relaxation des Problems das Optimum $(\frac{5}{2}, \frac{5}{2})$. Würde nun die Zeile zu y im optimalen Tableu zur Generierung der Branchingebenen benutzt, so wäre das „obere“ Problem leer, und das „untere“ würde sofort zur optimalen Lösung (1,2) führen, die dann schon ganzzahlig wäre. Wählt man jedoch x als Branchingvariable (siehe Abbildung 1), erhält man zwei neue Optimierungsprobleme, deren kleinster Lösungsbaum nicht weniger tief geworden ist als der des Ausgangsproblems. Selbst wenn das „rechte“ Problem nie wieder bearbeitet würde - was hier nicht der Fall ist, da die Relaxation des rechten Problems „besser“ ist als die des linken -, wäre trotzdem noch dieselbe Arbeit nötig, die schon für das Vaterproblem zu einer optimalen Lösung geführt hätte. Aber es kommt noch schlimmer: Durch die falsche Wahl der Branchingvariablen kann unser Problem eine derart unglückliche Stückelung erfahren, daß sie ohne weiteres den Berechnungsaufwand verdoppelt! Im obigen Beispiel etwa wird nun zunächst das rechte Problem weiterverarbeitet. Man kann sich leicht

höherdimensionale Probleme vorstellen, bei denen ein unglückliches Branching zwei Probleme erzeugt, an denen von da an immer abwechselnd gearbeitet wird, obwohl die Aufteilung in dieser Form gar nicht hätte gemacht werden müssen.
Die Motivation, die „richtige“ Branchingvariable zu wählen, ist also sehr groß. Im Gegensatz zu aufwendigen Pivotspaltenberechnungen im Simplex-Algorithmus erscheint es an dieser Stelle also durchaus angemessen, etwas mehr Rechenzeit in die „korrekte“ Wahl der Branchingvariablen zu investieren, um durch eine geringere Anzahl von Branchingschritten die Gesamtlaufzeit des Algorithmus trotz der längeren Dauer der einzelnen Schritte zu verbessern. Einige Strategien zur Bestimmung der Branchingebenen sollen nun vorgestellt werden.

Benutzerdefinierte Prioritäten

Für einige (gemischt-)ganzzahlige Optimierungsprobleme gilt, daß die Ganzzahligkeit einiger Variablen schon die der anderen nach sich zieht. Man denke etwa an eine Modellierung, bei der die Entscheidung, ob ein gewisses Projekt durchgeführt wird oder nicht, die Entscheidungen innerhalb dieses Projektes überlagert. Für einen allgemeinen Algorithmus ist es sehr schwer, solche Beziehungen zwischen den Variablen herauszufinden, und darum besteht eine Idee darin, eine benutzerdefinierte Ordnung auf den Variablen zu verwenden. Es wird dann diejenige ganzzahlige Variable zur Generierung der Branchingebenen benutzt, die von den in der optimalen Lösung der Relaxation noch gebrochen-rationalen die höchste Priorität aufweist.
Allerdings hat dieses Vorgehen einige unschöne Nebeneffekte: Nicht immer kann ja der Anwender eine so vernünftige Klassifizierung wie im obigen Beispiel angeben, einfach weil er selbst keine genaue Kenntnis über die Struktur des Problems hat, oder weil er einige Variablen eigentlich mit der gleichen Priorität belegen müßte. Der Algorithmus ist dann gegenüber dem Anwender im Vorteil, da er selbst durch seine Berechnungen Informationen über das gegebene Problem erhält, und sollte daher in erster Priorität selbst entscheiden und benutzerdefinierte Prioritäten lediglich bei Mehrdeutigkeiten verwenden. In der *MuPAD*-Implementierung ermittelt aber schon ein zweistufiges Verfahren die Branchingvariable, bei dem eine Regel mit erster Priorität zur Anwendung gelangt und eine zweite die Auswahl nur bei nicht eindeutiger Wahl der ersten übernimmt. Es erscheint daher nicht sinnvoll, noch eine dritte Stufe zu etablieren, da diese so gut wie nie zum Einsatz käme und weil sie, selbst wenn sie einen Einfluß erhielte, bedingt durch die getroffene Vorauswahl kaum Alternativen zur Verfügung hätte, so daß die benutzerdefinierten Prioritäten kaum eine signifikante Bedeutung für den Algorithmus hätten.

Strafkosten

Bei dieser Idee[2] wird der Versuch unternommen, für eine gegebene Variable zu schätzen, welcher Verlust in der Zielfunktion ausgegangen von der optimalen relaxierten Lösung hingenommen werden muß, um der Ganzzahligkeitsrestriktion zu genügen. Der geschätzte Verlust kann zum Beispiel auf Erfahrungen während früherer Branchingschritte mit dieser Variablen beruhen, wobei allerdings offen bleibt, wie diese Idee formal zu einem Verfahren präzisiert werden kann. Eine mehr praxisorientierte Möglichkeit besteht darin, die Variable mit der stärksten Abweichung von der Ganzzahligkeitsrestriktion zu wählen. Formal lautet dieses Kriterium:

Maximale Ganzzahligkeitsunzulässigkeit

Sei $(x^*, y^*)^T$ Lösung der Relaxation mit

$$\begin{array}{lcll} x_i^* & = & n_i + f_i & \text{, wobei } n_i \in \mathbb{N}, o \leq f_i \leq 1 \quad \forall 1 \leq i \leq h \leq k, \text{ und} \\ x_i^* & = & n_i & \text{, wobei } n_i \in \mathbb{N} \qquad\qquad \forall h \leq i \leq k. \end{array}$$

Dann wähle x_j mit $1 \leq j \leq h$ und $min(f_j, 1 - f_j) \geq min(f_i, 1 - fi) \quad \forall 1 \leq i \leq h$.
Dieses Kriterium ist zwar leicht zu berechnen, dafür hat das Resultat aber auch kaum Aussagekraft, bleibt doch die Zielfunktion bei diesem Vorgehen völlig unberücksichtigt. Daher ist eine Erweiterung der obigen Idee geschaffen worden, indem noch Koeffizienten an die f_i geschlagen werden.

[2] siehe [2], [3] und [4].

Zielfunktionsorientierte Strafkosten

Um den Strafkosten aber tatsächlich relevante Information abringen zu können, ist noch etwas mehr Anstrengung nötig als die Multiplikation mit konstanten Koeffizienten: In der *MuPAD*-Implementierung werden die Strafkosten einer Variablen nach dem Verlust in der Zielfunktion ermittelt, der mindestens hingenommen werden muß und durch einen Reoptimierungsschritt berechnet wird. Man erhält so genaugenommen zwei Strafkosten für jede Variable x_i, eine für die „untere“ und eine für die „obere“ Restriktion:
Sei p mit $A_{i,p} > 0$ so gewählt, daß

$$\frac{r_p}{A_{i,p}} \leq \frac{r_q}{A_{i,q}} \quad \forall q \text{ mit } A_{i,q} \geq 0.$$

Dann ist p eine zulässige Pivotspalte zu Zeile i im dualen Simplex-Algorithmus, und wir definieren die unteren Strafkosten durch $P_i^D = \frac{r_p \cdot f_i}{A_{i,p}} \geq 0$, was dem Wert entspricht, um den sich in diesem Schritt die Zielfunktion ändert. Da sie im dualen Simplex-Algorithmus monoton steigt, ist dieser Wert zugleich eine untere Schranke für den Verlust in der Kostenfunktion.
Ganz analog gehen wir zur Bestimmung der oberen Strafkosten vor: Sei s mit $A_{i,s} \leq 0$ so gewählt, daß

$$\frac{r_s}{A_{i,s}} \geq \frac{r_t}{A_{i,t}} \quad \forall t \text{ mit } A_{i,t} \leq 0.$$

Dann beläuft sich der Verlust in der Zielfunktion auf $P_i^D = \frac{r_s \cdot (f_i - 1)}{A_{i,s}} \geq 0$.
Nun stellt sich die Frage, wie diese zwei Strafkosten zu einem Wert zusammengefaßt werden sollen. Grundsätzlich stehen sich zwei Alternativen gegenüber: wir können entweder das Minimum oder das Maximum der beiden Werte als Strafkosten der Variable x_i annehmen. Wählen wir das Maximum, so ist die Wahrscheinlichkeit groß, daß das mit diesem Wert assoziierte Sohnproblem bald von der Berechnung ausgeschlossen werden kann, da der Wert seiner Relaxation schon nach wenigen Branchingschritten den der besten bis dahin bekannten Lösung übersteigt. Wählen wir das Minimum, so erhalten wir eine Aussage darüber, was uns die Ganzzahligkeitsbedingung von jedem x_i mindestens Kosten wird. Da für eine Eingabe in ein Computeralgebrasystem davon ausgegangen werden kann, daß der Algorithmus mit relativ kleinen Eingaben konfrontiert sein wird, ist es nicht zwingend, den Branch-and-Bound-Baum besonders klein zu halten, weswegen die Wahl des Maximums der beiden Werte keine echten Vorteile bringt. Die Strafkosten berechnen sich also durch $P_i = Min(P_i^D, P_i^U)$.[3]
Um nun schnell festzustellen, daß in dem gegebenen Problem keine bessere als eine bereits bekannte Lösung mehr zu finden ist, bzw. schnell zur besten Lösung des in Arbeit befindlichen Teilproblems zu gelangen, wird nun natürlich die Variable mit den größten zugehörigen Strafkosten als Branchingvariable gewählt. Nun, was heißt hier „schnell“? Wenn man in einem Ast des Branch-and-Bound-Baumes die aufsteigenden Werte der relaxierten Optima betrachtet, dann wird dieser Ast wahrscheinlich um so kürzer sein, je größer die Sprünge in der Zielfunktion zwischen je zwei aufeinanderfolgenden Knoten sind. Je größer also der Verlust in der Zielfunktion ist, der im nächsten Schritt für beide Sohnprobleme garantiert werden kann, um so niedriger wird wahrscheinlich der vom aktuellen Knoten ausgehende Teilbaum sein.

Raumorientierte Strafkosten

An dieser Stelle drängt sich nun eine andere Vorstellung eines „schnellen“ Hinlangens zu einer ganzzahligen Lösung auf: eine euklidische nämlich. Die Wahl der Branchingschnitte $x_i \leq n_i$ bzw. $x_i \geq n_i + 1$ hat sicher viele Vorteile; die Zerlegung des zulässigen Bereichs ist eine Partition und die neuen Restriktionen sind leicht durch die implizite Behandlung beidseitig-beschränkter Variablen im Simplex-Algorithmus zu behandeln und vergrößern so das Tableau nicht. Wenn man sich aber einen Polyeder vorstellt, der auf diese Weise von einer Ecke ausgehend zerlegt wird, dann gewinnt man den Eindruck, daß die beiden neuen Optima wahrscheinlich ganz in der (euklidischen) Nähe des alten liegen werden. Was also, wenn das gesuchte ganzzahlige Optimum sehr weit entfernt von unserer initialen relaxierten Lösung liegt? Die Hoffnung geht natürlich dahin, daß mit einem großen Verlust in der Kostenfunktion auch ein entsprechender „Raumgewinn“ einhergeht. Und weil unsere bounding-Funktion auf den Zielfunktionswerten beruht, macht eine Wahl der Branchingvariablen nach dem oben skizzierten Verfahren in

[3] siehe [2], [3] und [4].

erster Priorität auch sehr viel Sinn. Für den Fall aber, daß dieses Kriterium zu keiner eindeutigen Entscheidung führt, erfolgt die Wahl nach euklidischen Gesichtspunkten:

Angenommen, unsere zielfunktionsorientierte Selektionsprozedur läßt uns die Wahl zwischen den beiden Variablen x_i und x_k mit $P_i = P_k \geq P_s \quad \forall s \neq i,k$.[4] Da sich die Strafkosten P durch den Verlust in der Kostenfunktion nach einem dualen Simplexschritt im Tableau zum „oberen“ oder zum „unteren“ Problem berechnen, können die möglichen Pivotspalten - und damit Vektoren - in diesem simulierten Schritt mit den in Frage kommenden Variablen assoziiert werden; nehmen wir an, diese Spalten tragen die Indizes $j_1 \ldots j_t$ für x_i und $l_1 \ldots l_u$ für x_k. Ohne Einschränkung sei für beide Variablen angenommen, daß für den Wert von P der von P^D maßgeblich war. Dann kann durch den Schnitt der Vektoren zu den Spalten $j_1 \ldots j_t$ mit der Ebene $x_i = \lfloor x_i^* \rfloor$ und der Vektoren zu den Spalten $l_1 \ldots l_u$ mit der Ebene $x_k = \lfloor x_k^* \rfloor$ herausgefunden werden, welcher der Strahle ausgehend vom relaxierten Optimum uns im ersten Schritt zur am weitesten entfernten Nachbarecke führen wird. Die Variable zu dem Vektor wird dann Branchingvariable, und es wird ein Pivotschritt um ein Element aus der zugehörigen Spalte ausgeführt.

An dieser Stelle sei noch bemerkt, daß die Idee, die Information zur ersten Pivotspalte der Reoptimierung aus der Prozedur zur zielfunktionsorientierten Wahl der Branchingvariable in die Reoptimierungsfunktion hinüberzuretten, sich ganz unabhängig von der zweiten, euklidischen Stufe zeitsparend auf den Algorithmus auswirkt und daher in jedem Fall erfolgen sollte. Leider gilt für die euklidische Wahl der Branchingvariable nicht die analoge Eigenschaft wie für die zielfunktionsorientierte, daß wir den im ersten Reoptimierungsschritt gewonnen Abstand zum relaxierten Optimum garantieren können, da der duale Simplex-Algorithmus sich in den nächsten Schritten durchaus wieder auf das alte Optimum zu bewegen kann, während die Zielfunktion aber weiterhin monoton steigt. Es handelt sich hierbei also um eine reine Heuristik.

Man könnte nun noch eine dritte und vierte Stufe etablieren, die für den Fall auftretender Mehrdeutigkeiten nach der euklidischen Wahl die Branchingschnitte nach dem Zielfunktionsanstieg bzw. der Entfernung zur ersten Nachbarecke im Bruderproblem bestimmt, also im obigen Beispiel jeweils in P^U. Es gilt hier aber wieder das bereits für die Wahl nach Benutzerprioritäten gesagte: diese Stufen kämen wahrscheinlich fast niemals zum Tragen und werden daher einfach weggelassen. Es gibt einen Fall, für den die euklidische Wahl der Branchingvariable von großer Wichtigkeit ist. Nehmen wir an, die erste Relaxation eines (gemischt-)ganzzahligen Optimierungsproblems ist unbeschränkt. Dann gibt es zwei Möglichkeiten: Entweder, das gegebene Problem ist leer. Oder es hat eine zulässige Lösung, dann ist es auch unbeschränkt, denn ausgehend von dieser Lösung können wir demselben Vektor wie in der Relaxation folgen und erhalten unendlich viele weitere zulässige Lösungen, die immer kleinere Zielfunktionswerte haben, da die Zielfunktion im Verlauf dieses Vektors streng monoton fällt. Unser Problem reduziert sich also darauf, eine zulässige Lösung zu finden, oder nachzuweisen, daß es keine gibt. Die Zielfunktion spielt also überhaupt keine Rolle mehr, und sie wird daher einfach gleich Null gesetzt. Damit aber kollabiert nun die Wahl der Branchingvariablen unter Berücksichtigung der Zielfunktion, und das ist auch so gewollt, da sie ja kein Kriterium für das Vorhandensein einer Lösung darstellt. Also kommt die euklidische Wahl zum Tragen. Die Hoffnung ist nun, daß sich der zulässige Bereich der Relaxation in weiter Entfernung vom alten Optimum x^* zusammen mit dem geometrischen Simplex mit der Spitze x^* so weit geöffnet hat, daß wir auf ganzzahlige Lösungen in der Nähe hoffen können.

Die Branchingvariable wird also festgelegt werden, indem zunächst für jede nicht-ganzzahlige Komponente i von x zwei Strafkosten P_i^O und P_i^U (jeweils für das „obere“, mit $x_i \geq n_i + 1$ assoziierte, beziehungsweise das „untere“, mit $x_i \leq n_i$ assoziierte Teilproblem) bestimmt werden, und dann dasjenige i ausgewählt wird, für das $Minimum\{P_i^O, P_i^U\}$ maximal ist.

Dabei belaufen sich die Strafkosten auf den Verlust in der Zielfunktion, der durch das Hinzufügen der neuen Restriktion mindestens hingenommen werden muß. Sie können etwa durch die Betrachtung der Änderung in der Zielfunktion nach einem dualen Simplexschritt berechnet werden: Durch das Hinzufügen neuer Restriktionen im optimalen Tableau ändert sich die duale Zulässigkeit nicht, daher empfiehlt es sich, mittels der dualen Simplexmethode zu reoptimieren. Die Beobachtung, daß wahrscheinlich nur wenige Simplexschritte ausreichen, um wieder zu einem optimalen Tableau zu gelangen, rechtfertigt auch den Ansatz, unser relaxiertes Problem mit Hilfe des Simplex-Algorithmus zu lösen, obwohl andere, zum Teil effizientere Verfahren, wie etwa Interior-Point-Methoden, inzwischen bekannt sind. Diese lassen jedoch eine so einfache Reoptimierung nicht zu.

[4] Der besseren Übersichtlichkeit halber wird nur der Fall für zwei Variablen beschrieben; falls mehr zur Wahl stehen, ist das Vorgehen ganz analog.

Sofern nach dieser Wahl Mehrdeutigkeiten auftreten, erfolgt die Auswahl der Branchingvariablen dann nach raumorientierten Strafkosten.

Verschiedene Szenarien

Zur Veranschaulichung und um die gewählte Strategie glaubhaft zu machen, seien nun einige zweidimensionale Beispiele und die gewählten Branchingebenen graphisch dargestellt. Auf die mathematische Formulierung der gegebenen Probleme wird dabei verzichtet:

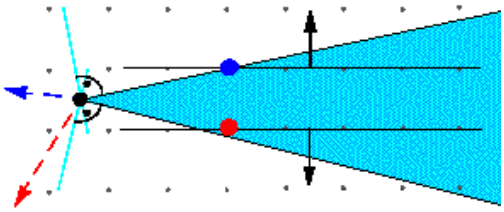


Abbildung 16: Mögliche Zielfunktionsvektoren

Der schwarze Punkt markiert die optimale Lösung der Relaxation. Die cyan-farbenen Linien zeigen den Winkel an, in dem der Zielfunktionsvektor liegen kann, damit sich an diesem Optimum für den gegebenen zulässigen Bereich nichts ändert. Es sind nun durch den blauen und den roten gestrichelten Pfeil zwei mögliche Zielfunktionsvektoren angegeben. Der blaue und der rote Punkt markieren jeweils das Optimum der ganzzahligen Aufgabe. In diesem Fall finden die euklidische und - unabhängig von der exakten Kostenfunktion im möglichen Bereich - auch die zielfunktionsorientierte Prozedur zur Wahl der Branchingvariablen die eingezeichneten Schnitte. Das alternative horizontale Branching würde, obwohl es nur ein Sohnproblem erzeugt, mehr Schritte bis zur optimalen Lösung benötigen.

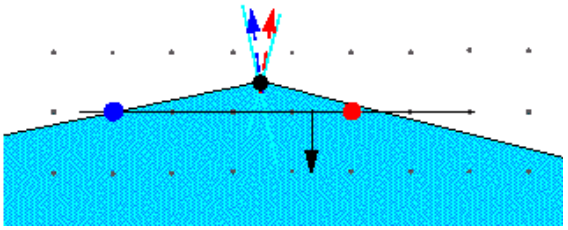


Abbildung 17: Mögliche Zielfunktionsvektoren

Analog zu Abb.2 sind zwei alternative Zielfunktionen mit ihren jeweiligen zugehörigen ganzzahligen Optima in rot und blau angegeben. Wieder entscheiden sich beide besprochenen Verfahren gleich, nämlich für das eingezeichnete Branching nach unten (und oben). Für die zielfunktionsorientierte Wahl spielt wiederum der exakte Vektor im möglichen Bereich keine Rolle. Auch hier ist die getroffene Wahl geschickt, da ein Branching nach rechts und links eine unnötige Stückelung des zulässigen Bereichs bewirkt.

In diesem Beispiel wird die Wahl der Branchingvariablen nun zielfunktionsabhängig. Zum roten Zielfunktionsvektor gehören die vertikalen, zum blauen die horizontalen Schnitte. Der blaue und der rote Kreis markieren jeweils die Ecke nach einem Reoptimierungsschritt im Sohnproblem mit den kleineren assoziierten Strafkosten. Durch diese beiden Ecken ist eine magentafarbene Gerade gelegt. Das gefällte Lot auf diese Gerade durch das relaxierte Optimum markiert den Übergang von einem Branching zum anderen. Falls der Zielfunktionsvektor kollinear zum Lot verliefe, würde die zielfunktionsorientierte Wahl der Branchingvariablen keine Entscheidung mehr liefern, und es würden nach euklidischen Gesichtspunkten die blauen Branchingschnitte eingeführt.

Ein weiterer Fall, in dem das zielfunktionsorientierte Kriterium versagt: Falls der (hier magentafarbene) Zielfunktionsvektor kollinear zu einem Randvektor des für das angegebene relaxierte Optimum gültigen Bereichs (cyan-

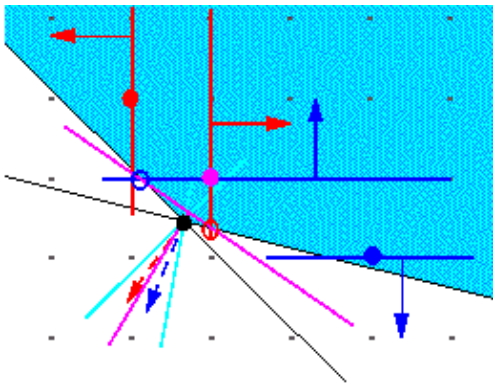


Abbildung 18: Mögliche Zielfunktionsvektoren

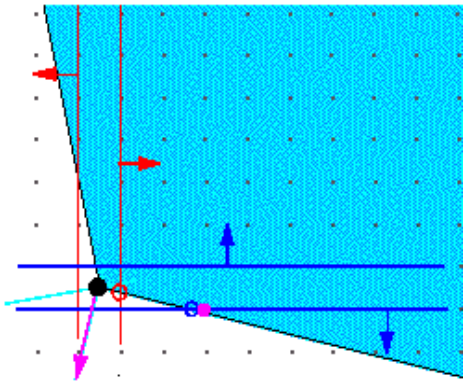


Abbildung 19: Mögliche Zielfunktionsvektoren

farben dargestellt) verläuft, so liefert die zielfunktionsorientierte Prozedur ebenfalls keine Entscheidung zwischen den beiden möglichen roten oder blauen Branchingschnitten, da die Ecken nach einem Reoptimierungsschritt im mit den geringeren Kosten verbundenen Sohnproblem (hier mit dem roten und dem blauen Kreis markiert) mit denselben Strafkosten P=0 assoziiert sind. Die euklidisch orientierte Prozedur wählt dagegen das blaue Branching aus. Der magenta-farbene Punkt kennzeichnet das ganzzahligen Optimum, und man sieht, daß wir uns durch die getroffene Wahl besser an diesen Punkt heranarbeiten, als hätten wir die rotmarkierten Branchingschnitte eingeführt.

Generalisierte obere Schranken

Im Algorithmus von Dakin werden Branchingschnitte der Form $x \leq b$ und $x \geq a$ eingeführt. In manchen Fällen jedoch scheint die Wahl dieser Schnitte unglücklich zu sein. Nehmen wir an, es existieren verallgemeinerte obere Schranken der Form $\sum_{j \in Q} x_j = b \geq 0$ für Variablen $x_j \in N$. Dann impliziert dies $x_j \leq b \quad \forall j$, was die Bezeichnung dieser Art von Restriktion erklärt. Für die weitere Erklärung gelte der Einfachheit halber $b = 1$. Dann bringt uns im Algorithmus die neue Restriktion $x_l = 0$ wahrscheinlich kaum weiter, während die Restriktion $x_l = 1$ sehr rigide ist, da sie schon $x_j = 0 \quad \forall j \neq l$ impliziert. Das eingeführte Branching ist also sehr ungleich. In dieser Situation könnten eventuell andere Schnitte $\sum_{j \in P} x_j = 1$ und $\sum_{j \in Q \setminus P} x_j = 1$ für ein $P \subseteq M$ besser sein.

Für einen Algorithmus zur Lösung allgemeiner (gemischt-)ganzzahliger Optimierungsprobleme ergeben sich dadurch allerdings gewisse Schwierigkeiten, da die Struktur der Eingabe nicht im vorhinein bekannt ist und er so solche generalisierte obere Schranken selbst erkennen müßte. Ein Automatismus würde dann aber alle Restriktio-

nen mit ausschließlich nicht-negativen Koeffizienten und Variablen als zu dieser Form gehörig erkennen, und es erscheint fraglich, ob es sehr hilfreich ist, für all diese Restriktionen Branchingschnitte der beschriebenen Form in Erwägung zu ziehen. Außerdem bliebe dann immer noch die Frage offen, wie man mit vertretbarem Aufwand eine „gute“ Teilmenge P von Q bestimmen kann, die eine merkliche Verbesserung zum kanonischen Branching bewirkt. Daher macht diese Form der Branchingschnitte in erster Linie dann einen Sinn, wenn für ein Problem einer bekannten Struktur ein Algorithmus gesucht wird und diese Fragen im vorhinein beantwortet werden können. Darüberhinaus hat dieses Branching den Nachteil, das Tableau zu vergrößern, so daß schließlich auf diese Möglichkeit der Generierung von Branchingschnitten in der neuen Library verzichtet wurde.

Schlupfvariablen als Ausgangspunkt für Branchingschnitte

In der einschlägigen Literatur findet sich häufig der Hinweis, daß für Schlupfvariablen zu einer Restriktion mit ausschließlich ganzzahligen Variablen nicht selbst Ganzzahligkeit gefordert werden muß, da sich diese aus der Ganzzahligkeit der vorkommenden Variablen schon ergibt. Das ist auch richtig, aber für den Autor ist es eine offene Frage, ob nicht durch die Hinzunahme von Branchingschnitten, die von einer Schlupfvariable herrühren, eine Leistungssteigerung erreicht werden könnte (siehe Abbildung 6).

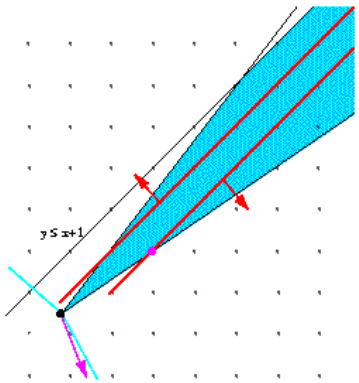


Abbildung 20: Mögliche Zielfunktionsvektoren

Allerdings bezahlt man diese einfache Möglichkeit, andere als koordinatenebenenpa-rallele Schnitte einzuführen, durch einen Mehraufwand bei der Bestimmung der Branchingvariablen, da diese ja nun auch Schlupfvariable sein darf. Hier wären Tests nötig, um die Frage zu klären, ob die Verwendung auch solcher Schnittebenen sich auszahlt - insbesondere auch für (gemischt-)ganzzahlige Optimierungsaufgaben mit einer vorgegebenen Struktur. Jedenfalls sollte die Möglichkeit, auch Schlupfvariablen zur Generierung von Branchingschnitten zu verwenden, nicht von vornherein verworfen werden.

Ausgehend vom mit dem schwarzen Punkt markierten relaxierten Optimum kommen nun drei verschiedene Branchingvariablen in Frage, nämlich die benutzerdefinierten Variablen x und y, sowie die Schlupfvariable zur Restriktion y x+1. Letztere führt zu den rot eingezeichneten Branchingschnitten, die für den magentafarben eingezeichneten Zielfunktionsvektor den größten Verlust in der Kostenfunktion bewirken.

Benutzung der Library - Eine Beispielsitzung

Bei dem verwendeten `linopt` handelt es sich um eine externe Library, die in einer der nächsten *MuPAD*-Versionen die bestehende `linopt`-Library ersetzen wird.

Zunächst werden einige gewöhnliche lineare Optimierungsaufgaben gelöst. Das Format sieht zunächst die Angabe der Restriktionen in Form einer Liste oder einer Menge gefolgt von der Zielfunktion vor. Optional folgt dann die Menge der nicht-negativen Variablen bzw. der Option `NonNegative` anstelle der Menge mit allen vorkommenden Variablen.

```
>> B := [[se1<=5,se2<=10,se3<=5,e1a<=3,e1b<=3,e2e1<=7,e2b<=4,e2e3<=9,
e3b<=3,e3d<=7,db<=1,dv3<=8,ba<=2,bc<=8,av1<=8,ac<=5,cd<=6,cv1<=1,
cv2<=2,cv3<=4,v3v2<=7,v3dest<=8,v1v2<=2,v1dest<=2,v2dest<=10,
se1+e2e1=e1a+e1b,se2=e2e1+e2e3+e2b,se3+e2e3=e3b+e3d,
e1b+e2b+e3b+db=ba+bc,e1a+ba=ac+av1,e3d+cd=db+dv3,
ac+bc=cd+cv3+cv2+cv1,av1+cv1=v1dest+v1v2,cv3+dv3=v3dest+v3v2,
cv2+v1v2+v3v2=v2dest],se1+se2+se3,NonNegative]:
>> z := time(): linopt::maximize(B): time()-z; %2;

1620

[Optimal, {e2e3 = 5, v2dest = 8, v3dest = 8, cv1 = 0, ac = 1,
   ba = 2, db = 0, av1 = 4, v1v2 = 2, cv2 = 2, se3 = 3,
   se1 = 5, cd = 3, v3v2 = 4, cv3 = 4, e1a = 3, bc = 8,
   e1b = 3, e2b = 4, e3b = 3, se2 = 10, dv3 = 8, e2e1 = 1,
   v1dest = 2, e3d = 5}, 18]
```

Als Ergebnis erhält man eine Liste mit dem Status der Lösung, `Optimal`, `Empty` oder `Unbounded`, dann eine zulässige Variablenbelegung, die im Falle des Status „Empty“ auch leer sein kann, respektive im Falle des Status „Unbounded“ noch von einer nicht-negativen Variablen phi abhängt. Zuletzt wird noch der erreichte Zielfunktionswert angegeben. Sofern minimiert (maximiert) wurde und der Status der Lösung `Empty` ist, ist dieser Wert `infinity` (bzw. `-infinity`). Im Falle eines unbeschränkten Problemes hängt die erreichte Zielfunktion noch von der nicht-negativen Variable phi ab.

```
>> X:= [[s1<=4,s2<=5,s3<=10,s4<=7,s5<=8,x16<=5,x17<=4,x21<=4,x27<=5,x28<=4,
x32<=7,x38<=1,x34<=3,x48<=4,x49<=5,x45<=3,x59<=4,x510<=5,
x67<=4,x6d<=4,x78<=7,x7d<=5,x8d<=10,x98<=3,x9d<=7,x109<=3,x10d<=8,
s1+x21=x16+x17,s2+x32=x21+x27+x28,s3=x32+x38+x34,
s4+x34=x48+x49+x45, s5+x45=x59+x510,x16=x67+x6d,
x17+x27+x67=x7d+x78, x28+x38+x48+x78+x98=x8d,
x49+x59+x109=x98+x9d,x510=x109+x10d], s1+s2+s3+s4+s5, NonNegative]:
>> z := time():linopt::maximize(X):time()-z;%2;

1950

[Optimal, {s1 = 4, x9d = 7, s2 = 5, x8d = 10, s4 = 6, s3 = 8,
   x34 = 0, s5 = 8, x21 = 4, x45 = 1, x48 = 0, x16 = 4,
   x109 = 0, x17 = 4, x38 = 1, x67 = 0, x32 = 7, x27 = 4,
   x510 = 5, x10d = 5, x28 = 4, x6d = 4, x78 = 3, x59 = 4,
   x49 = 5, x7d = 5, x98 = 2}, 31]

>> K := [[2*x+2*y>=4,-2*x+4*y>=-2,-2*x+y>=-8,-2*x+y<=-2,y<=6],
x+y,NonNegative]:
```

Zur Veranschaulichung zweidimensionaler Aufgaben kann der zulässige Bereich auch automatisch gezeichnet werden. Erwartet wird ein Problem in zwei Variablen und eine Liste von ihnen, wobei die Reihenfolge festlegt, welche Variable für die horizontale und welche für die vertikale Achse steht.

```
>> plot2d(linopt::plot_data(K,[x,y])):
```

Zur Berechnung aller zulässigen Ecken eines gegebenen (nicht notwendig zweidimensionalen!) Problems, steht die Prozedur `linopt::corners` zur Verfügung. Auch sie erwartet neben dem Problem noch eine Liste mit den vorkommenden Variablen, um die Ecken in dieser Reihenfolge ausgeben zu können. Optional kann dann noch `All` folgen; die Funktion liefert dann alle Ecken des Problems, also auch die nicht zulässigen.

```
>> linopt::corners(K,[x,y]);
```

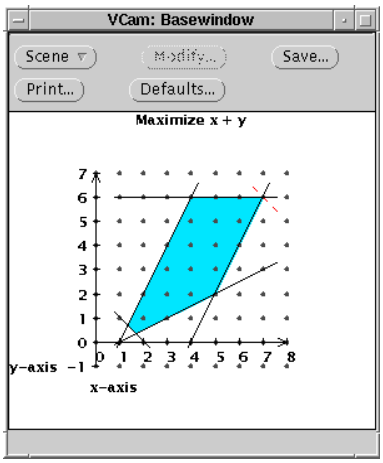


Abbildung 21: Grafische Darstellung des Problems `K`

```
[{[4, 6], [4/3, 2/3], [5/3, 1/3], [7, 6], [5, 2]}, 13, [7, 6]]

>> linopt::corners(K,[x,y],All)[1];

{[-4, 6], [4, 6], [4/3, 2/3], [5/3, 1/3], [7, 6], [13, 6],
   [10/3, -4/3], [0, 0], [1, 0], [2, 0], [0, 2], [0, -2],
   [0, -1/2], [4, 0], [0, 6], [5, 2], [0, -8]}
```

Mit der *MuPAD*-Funktion `setuserinfo` können Berechnungsschritte transparent gemacht werden. Interessant sind besonders die Entwicklung der Zielfunktion und die Sequenz der angesprungenen Ecken.

```
>> setuserinfo(linopt,8):linopt::maximize(K);
 Entering linopt::maximize
 Entering Tableau Constructor
 Entering Tableau::simplex
 Entering Phase I
 Entering Tableau::pivot1
 Result after this step   {x = 0, y = 0}
 Objective Function Value 0
 Entering Tableau::pivot1
 Result after this step   {x = 1, y = 0}
 Objective Function Value -1
 Entering Tableau::pivot1
 Result after this step   {y = 1/3, x = 5/3}
 Objective Function Value -2
 Phase I finished
 Entering Phase II
 Entering Tableau::pivot2
 Result after this step   {y = 2, x = 5}
 Objective Function Value -7
 Entering Tableau::pivot2
 Result after this step   {y = 6, x = 7}
 Objective Function Value -13
 Entering Tableau::pivot2
 Phase II finished

[Optimal, {y = 6, x = 7}, 13]
```

Die Option `DualPrices` von `linopt::maximize` (`linopt::minimize`) erzwingt die Ausgabe der Schat-

tenpreise eines Problemes. Neben der optimalen Lösung wird zusätzlich eine Liste der Restriktionen zusammen mit dem assoziierten Schattenpreis geliefert.

```
>> setuserinfo(linopt,0):linopt::maximize(K,DualPrices);

[Optimal, {y = 6, x = 7}, 13], {[0 <= x, 0], [0 <= y, 0],
   [y - 2 x <= -2, 0], [4 <= 2 x + 2 y, 0],
   [-2 <= 4 y - 2 x, 0], [y <= 6, 3/2], [-8 <= y - 2 x, 1/2]}

>> C := [[x1+x2<=5,x1>=3,x1<=x3],-x1+x2-x3,NonNegative]:
>> linopt::maximize(C);

            [Optimal, {x1 = 3, x2 = 2, x3 = 3}, -4]
```

Ein Beispiel für ein unbeschränktes Problem:

```
>> linopt::minimize(C);

[Unbounded, {x3 = PHI1 + 5, x2 = 0, x1 = 5}, - PHI1 - 10]
F := [[8*y-2*x<=21,6*x+12*y<=63,x<=6],10*y-x,NonNegative]:
>> plot2d(linopt::plot_data(F,[x,y])):
```

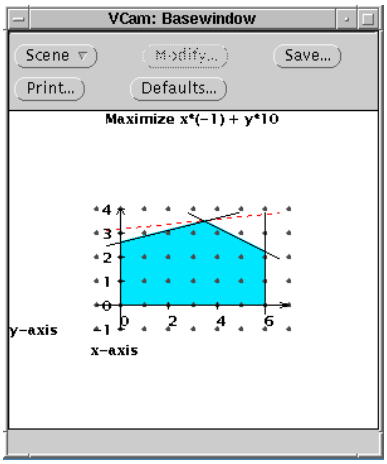


Abbildung 22: Grafische Darstellung des Problems `F`

```
>> linopt::maximize(F);

[Unbounded, {x = 7/2, y = 7/2}, 63/2]
```

Um für bestimmte Variablen Ganzzahligkeit zu erzwingen, wird an die Liste der Liste mit den Restriktionen, der Zielfunktion und der Menge der nicht-negativen Variablen noch eine Menge mit den Integervariablen angehängt. Letztere kann auch durch die Option `All` ersetzt werden, die dann für die Menge mit allen vorkommenden Variablen steht.

```
>> linopt::maximize(append(F,All));

[Optimal, {x = 2, y = 3}, 28]

>> Q := [{x1<=1,x2<=1,x3<=1,x4<=1,x5<=1,x6<=1,x7<=1,x8<=1,x9<=1,
          774*x1+76*x2+ 22*x3+42*x4+ 21*x5+760*x6+818*x7+62*x8+785*x9 <= 1500,
```

```
          67*x1+27*x2+794*x3+53*x4+234*x5+ 32*x6+797*x7+97*x8+435*x9 <= 1500},
          77*x1+6*x2+3*x3+6*x4+33*x5+13*x6+110*x7+21*x8+47*x9, NonNegative]:
>> z := time(): linopt::maximize(Q): time()-z; %2;

500

[Optimal, {x2 = 0, x4 = 1, x6 = 0, x5 = 1, x3 = 209587/ 613082,
   x7 = 1, x9 = 0, x8 = 1, x1 = 217620/306541}, 138366181/613082]

>> z := time():linopt::maximize(append(Q,All)):time()-z;%2;
1970

[Optimal, {x1 = 0, x3 = 0, x2 = 1, x4 = 1, x6 = 0, x5 = 1,
x7 = 1, x9 = 0, x8 = 1}, 176]

>> W:= [[x1+x2+x3+x4+x5<=4, x1+x2+x3+x4-x5<=3,x1+x2+x3-x4+x5<=3,
        x1+x2-x3+x4+x5<=3,x1-x2+x3+x4+x5<=3,-x1+x2+x3+x4+x5<=3,
        x1+x2+x3-x4-x5<=2,x1+x2-x3+x4-x5<=2,x1-x2+x3+x4-x5<=2,
        -x1+x2+x3+x4-x5<=2,x1+x2-x3-x4+x5<=2, x1-x2+x3-x4+x5<=2,
        -x1+x2+x3-x4+x5<=2,x1-x2-x3+x4+x5<=2,-x1+x2-x3+x4+x5<=2,
        -x1-x2+x3+x4+x5<=2, x1+x2-x3-x4-x5<=1,x1-x2+x3-x4-x5<=1,
        x1-x2-x3+x4-x5<=1,x1-x2-x3-x4+x5<=1,-x1+x2+x3-x4-x5<=1,
        -x1+x2-x3+x4-x5<=1,-x1+x2-x3-x4+x5<=1,-x1-x2+x3+x4-x5<=1,
        -x1-x2+x3-x4+x5<=1,-x1-x2-x3+x4+x5<=1, x1-x2-x3-x4-x5<=0,
        -x1+x2-x3-x4-x5<=0,-x1-x2+x3-x4-x5<=1,-x1-x2-x3+x4-x5<=0,
        -x1-x2-x3-x4+x5<=0, -x1-x2-x3-x4-x5<=-1],x1+x2+x3+x4+x5,NonNegative]:

>> z := time():linopt::maximize(W):time()-z;%2;

1880

[Optimal, {x1 = 2, x2 = 1/2, x3 = 1/2, x4 = 1/2, x5 = 1/2}, 4]

>> z := time():linopt::maximize(append(W,All)):time()-z;%2;

7450

    [Optimal, {x1 = 0, x2 = 0, x4 = 0, x5 = 0, x3 = 1}, 1]
```

Ausblick

Die in diesem Artikel beschriebenen Kriterien zur Bestimmung der Branchingschnitte sind nur ein Ausschnitt aus den zur Erstellung einer Bibliothek zum Lösen (gemischt-)ganzzahliger Optimierungsaufgaben notwendigen Überlegungen. Die neue Library `linopt` enthält einen verbesserten Simplex-Algorithmus mit impliziter Behandlung von Gleichungen sowie unbeschränkter und beidseitig beschränkter Variablen, eine Lehrversion mit transparenten Tableaus und der Möglichkeit zur Steuerung der Berechnung durch Vorgabe von Pivotelementen, sie nutzt Teightening-Verfahren und liegt darüberhinaus auch in einer parallelen Version vor. In der genannten Diplomarbeit [1] werden außerdem Besonderheiten bei der Implementierung eines (gemischt-)ganzzahligen Optimierers für ein Computeralgebrasystem beschrieben.

Literatur

[1] Sellmann, Meinolf. *Erstellung einer Library zur Bestimmung (gemischt-)ganzzahliger Lösungen li-*

nearer Optimierungsaufgaben für das Computeralgebrasystem MuPAD. Diplomarbeit an der Universität/Gesamthochschule Paderborn. Fachbereich 17, Mathematik und Informatik. Arbeitsgruppe Fuchssteiner, voraussichtlich 1997.

[2] Nemhauser, George L. *Integer and Combinatorial Optimization*. 1988.

[3] Salkin, Harvey M. *Foundations of Integer Programming*. 1989.

[4] Papadimitriou, Christos H. *Combinatorial Optimization*. 1982.

Computer Algebra and Differential Equations — An Overview

Werner M. Seiler
Institut für Algorithmen und Kognitive Systeme
Universität Karlsruhe, 76128 Karlsruhe, Germany
werner.seiler@informatik.uni-karlsruhe.de
http://iaks-www.ira.uka.de/iaks-calmet/werner/werner.html

We present an informal overview of a number of approaches to differential equations popular in computer algebra. This includes symmetry and completion theory, local analysis, differential ideal and Galois theory, dynamical systems and numerical analysis. A large bibliography is provided.

Introduction

Differential equations represent one of the largest fields within mathematics. Besides being an interesting subject of their own right one can hardly overestimate their importance for applications. They appear in natural and engineering sciences and increasingly often in economics and social sciences. Whenever a continuous process is modeled mathematically, chances are high that differential equations are used.

Thus it is not surprising that differential equations also play an important role in computer algebra and most general purpose computer algebra systems provide some kind of `solve` command. Many casual users believe that designing and improving such procedures is a central problem in computer algebra. But the real situation is somewhat different. Many computer algebra applications to differential equations work indirectly; they help to study and understand properties of the solution space.

The purpose of this article is to sketch in an informal way some of the main research directions in this field and to provide a starting point for more detailed studies by giving a large number of references. We omit all mathematical details (there is not a single formula in this article!) but describe briefly the central ideas. For the same reason we tried to cite introductory articles or books and not the historically first or the most "ground breaking" work.

The bibliography is of course far from being exhaustive. As a further source of references one should mention the survey [104] by Singer. It gives much more details, especially on the more algebraic approaches, and contains a large bibliography. The same holds for the more focused surveys by Hereman [50, 51] covering symmetry theory and related fields and the one by MacCallum [65] on the integration of ordinary differential equations. In addition there have been three conferences devoted exclusively to differential equations and computer algebra. Their proceedings [103, 112, 113] contain a number of useful introductory or review articles on more specialized topics.

We are taking a rather broad view and consider more or less any constructive approach to differential equations as "computer algebra". This also implies that we do not pay special attention to implementations. Among the many different approaches to differential equations which fall under this broad definition of computer algebra one can distinguish certain directions which have found most attention (at least measured in the number of articles devoted to them). We concentrate in this article on the following eight topics: (i) *solving* differential equations, (ii) *local analysis*, (iii) *symmetry analysis*, (iv) *completion*, (v) *differential ideal theory*, (vi) *differential Galois theory*, (vii) *dynamical systems theory*, and (viii) the relation between *numerical analysis* and computer algebra.

A comparison of the impact made by symmetry analysis and by differential Galois theory, respectively, demonstrates the importance of computer algebra tools. The latter one is a hardly known theory studied by a few pure mathematicians. The former one remained in the same state for many decades following Lie's original work. One reason was definitely the tedious determination of the symmetry algebra. As soon as computer algebra systems

emerged, the first packages to set up at least the determining equations were written. Since then Lie methods belong to the standard tools for treating differential equations.

Solving Differential Equations

Most computer algebra systems can solve some differential equations.[1] They mainly apply some standard techniques like those in Zwillinger's handbook [127] or try some "pattern matching" in a list of solved equations like Kamke [58]. Heuristics often extend the applicability of these techniques, for example by finding a transformation such that a given equation can be handled by the implemented methods.

Although this approach solves more differential equations than one might expect (see e. g. the recent review by Postel and Zimmermann [86][2]), it has some drawbacks. A major one is that no information is obtained, if the computer algebra system does *not* return a solution. It could be that the given differential equation has indeed no solution (or at least none in closed form) or that simply the heuristics were not able to determine a suitable transformation.

For that reason researchers in computer algebra are more interested in *decision algorithms*. These either yield a solution in a specific class of functions or decide that no such solution exists. However, so far only for linear ordinary differential equations such algorithms are known. There it is possible to decide with the help of differential Galois theory whether or not Liouvillian solutions exist.

There exists a number of reasons for this perhaps disappointing situation. Computability theory yields principal limits to what can be solved. For example if one restricts to *computable functions* some classical existence theorems for differential equations fail [1, 87]. More precisely, one has constructed examples of differential equations where one can show that solutions exists but that it is not possible to compute them. Some further (positive and negative) results in this direction can be found in [23].

Ideally a solution algorithm should return the *general solution*. But for nonlinear equations it is surprisingly difficult even just to define this term. A rigorous resolution of this problem based on differential ideal theory was only recently presented [53]. Intuitively one would expect that the general solution depends on some arbitrary parameters (constants or functions) and that every solution of the differential equation can be obtained by a suitable specialization of these. This works fine for linear equations where the solution space has the structure of a vector space. But many nonlinear equations possess in addition *singular integrals* not contained in the general solution.

Similarly, defining the term "closed form solution" is notoriously difficult. Is a solution in terms of, say, Bessel functions in closed form or not? Up to now no generally accepted definition has emerged. The basic idea behind "closed form" is that of *finite constructibility* out of a set of "elementary functions". Note that this is an algebraic and not an analytic property! On the practical side one must see that even if a solution in closed form can be computed it may take very long and the result may be completely useless, as it is too large. Especially, if the main goal is to obtain an impression of the behavior of the solution, it is usually much more efficient to resort to numerical methods.

In any case one can state that a notable solution theory exists only for ordinary differential equations (see e. g. the survey [65]), mainly based on differential Galois theory. But the algorithms to compute the general solution suffer from a very high complexity and are in practice often rather useless, especially for higher order equations. One way out is to incorporate heuristics as mentioned above.

Another possibility that also addresses the problem of useless output is to aim for "simple" solutions [7]. Popular variants are polynomial, rational [3] or exponential [11, 81] solutions. Because of their simple structure it is often possible to determine such solutions, if they exist, rather fast. But one should note that the classical methods for their computation are not always useful for computer algebra. It is still an active field of research to design effective algorithms being able to handle larger examples.

For partial differential equations the situation is much worse. In the last century mathematicians designed some solution methods (see e. g. the survey [118]). However, most of them are meanwhile almost forgotten; at least they are no longer found in textbooks on differential equations. It could be quite interesting to revive some of them for

[1] *MuPAD* provides the command `odesolve` for this task.

[2] An updated version can be found at `http://www.mupad.de/BIB/poszim96.html`.

use in computer algebra systems. There exist a few implementations of standard techniques like characteristics, separation of variables or integral transforms, but they can usually handle only rather simple equations. Often they just reduce to a system of (nonlinear) ordinary differential equations and the question is whether this can be solved. One can argue whether it really makes sense to speak of the general solution of a system of partial differential equations. For example one definition of a harmonic function is that it solves the Laplace equation (or more generally all holomorphic functions are solutions of the Cauchy-Riemann equation). Thus one might prefer to say that the Laplace equation defines a class of functions. In some simple cases like the wave equation one can give an explicit parameterization of this class in terms of some arbitrary functions which one may call the general solution. But usually no such parameterization exists. In order to get a really well-defined problem one must prescribe some initial or boundary conditions. In most applications such conditions arise automatically anyway.

Local Analysis

If it is not possible to construct a closed-form solution, one may go for an approximate solution describing the behavior of the solution in the neighborhood of a given point.[3] At *ordinary points* a Taylor series suffices; at *singular points* more general expansions must be used. In the case of linear differential equations singular points are only possible at singularities of the coefficients. Therefore one speaks of *fixed* singularities. Using the *Newton polygon* of the associated differential operator they can be further classified into *regular* and *irregular* ones [25].

Local analysis is a complex theory, even if one studies only real equations. Especially, if one wants to determine the radius of convergence of a formal solution, one must consider complex singularities. In the neighborhood of regular singular point one can represent the solution in form of a *Frobenius series*, a polynomial in $\log x$ with Taylor series coefficients multiplied by a factor $(x - x_0)^\alpha$ where α is a complex number. At irregular singular points the solution has usually an essential singularity and varies so rapidly that it makes no sense to construct an approximation; instead one tries to capture the asymptotic behavior which requires the addition of an exponential part.

There exist various algorithms for the construction of approximate or asymptotic solutions, partly dating back to Frobenius. Some are discussed together with implementations in [82, 115]. A main problem in the concrete application is that one cannot use an approximation of the location of the singularities. Thus one must not only solve polynomial equations but in general work with algebraic numbers which is quite expensive in any computer algebra system. However, with a careful analysis of the algorithms one can significantly reduce the necessary amount of computations with algebraic numbers.

Recent work concerns an extension to first order systems [6, 81]. In principle, one can transform any system into a single equation of higher order, e. g. using *cyclic vectors*. But this approach is rather inefficient, especially in higher dimensions. Hence one is interested in dealing directly with systems. *Moser's algorithm* performs here the classification into regular and irregular singularities; a rational version of it avoiding the use of algebraic extensions was presented by Barkatou [5].

For nonlinear differential equations the situation becomes much more complicated as *spontaneous* or *movable* singularities may occur, i. e. their location depends on initial or boundary data. One important direction is the *Painlevé theory* [54]. It was introduced by Painlevé while searching for new special functions (there still exists a strong connection between the local analysis of ordinary differential equations and special function theory). If all singularities are poles, no branch points appear in the (general) solution and it is single valued. A differential equation without movable branch points is said to possess the *Painlevé property* or to be integrable in the sense of Painlevé.

In general, it is not possible to check algorithmically whether or not a given differential equation has the Painlevé property. But there exist methods to check at least some necessary conditions; such methods are usually called *Painlevé test* [21]. One usually tries to construct a Laurent series around the singularity. Essentially, the test is passed, if this expansion has sufficiently many *resonances* or *Fuchsian indices* (free coefficients) to represent the general solution and if these occur at non-negative powers. In the case of negative resonances a perturbation approach [22] yields further information. Some references concerning implementations can be found in [95].

[3] Especially in the linear case local solutions can also be very useful for the construction of closed-form solutions.

Weiss *et al.* [122] generalized the Painlevé theory to partial differential equations where a whole singularity manifold must be considered. This extension is much used in the theory of integrable systems, as the Painlevé test represents an important indicator for integrability and can be performed comparatively easily. The *Painlevé conjecture* states that every ordinary differential equation obtained by symmetry reduction of an integrable system is of Painlevé type; only weakened versions of it have been proven [2, 71]. Truncated series expansions are useful for constructing Bäcklund transformations, Lax pairs and much more [121]. There also exist relations to non-classical symmetry reductions [30].

Symmetry Analysis

Symmetry analysis [9, 77, 111] has made the strongest impact on computer algebra applications to differential equations. The most general definition of a symmetry is that of a transformation that maps solutions into solutions. Depending on the kind of transformations considered one obtains different kinds of symmetries. One possible application of symmetries is the construction of (special) solutions. Other goals are classifications, a proof of complete integrability, separation ansätze, conservation laws and much more.

Symmetry analysis goes back to the seminal work of Lie. He developed the concept of Lie groups in his quest for a Galois theory for differential equations. As we will see later, not much has remained of this original motivation. Symmetry and Galois theory have developed in very different directions. Even the relation between the Lie symmetry and the Galois group of a differential equation is rather unclear.

The most popular form of symmetry analysis deals with *point symmetries*. They are generated by vector fields acting on the space of independent and dependent variables. These vector fields span the Lie algebra of the Lie group of symmetries. The decisive observation of Lie was that it often suffices to work with the vector fields (or infinitesimal symmetries) instead of the symmetries themselves. This leads effectively to a linearization of the problem.

The symmetry generators arise as the solutions of a linear system of partial differential equations, the *determining system*. For ordinary differential equations it is unfortunately sometimes as difficult to solve this system as to solve the original one. This holds especially for first order equations where the original equation is just the characteristic equation of the determining equation. For partial differential equations the determining system is typically very over-determined and contains often some trivial equations allowing in many cases a rather straightforward solution.

For ordinary differential equations the existence of a sufficiently large, *solvable* symmetry algebra implies that its general solution can be constructed by quadratures only, as each symmetry allows us to reduce the order of the equation by one. In the case of partial differential equations symmetry reductions yield only special solutions, namely those being invariant under the symmetry group. Here each symmetry allows us to reduce the number of independent variables by one.

However, at intermediate steps of the reduction again linear partial differential equations must be solved. For in order to obtain the reduction, one must either perform a coordinate transformation such that the symmetry generator is rectified (so-called *canonical coordinates*) or the *differential invariants* of the symmetry must be determined. These are functions annihilated by the generator.

Thus the usefulness of Lie symmetries depends crucially on the ability to solve effectively all the arising linear partial differential equations. At first sight it might look, as if, especially for ordinary differential equations, we made the problem only worse. But in many cases of practical interest it turns out that is much simpler to solve these linear partial differential equations than the original equation.

There exist so many implementations of symmetry methods that it is rather difficult to keep an overview; we refer again to the surveys by Hereman [50, 51]. In almost any computer algebra system one can find a package for setting up the determining system. A few of the packages try furthermore some heuristics to solve it automatically. Again it is rather surprising how often this suffices to obtain the complete symmetry algebra. The symmetry package of MAPLE is somewhat unusual, as it uses the exterior systems approach of Harrison and Estabrook [47].

Although Lie point symmetries proved to be very useful in many applications, many differential equations of practical interest have no such symmetries. There are two basic approaches to generalize the theory. One can consider more general transformations; this leads to *generalized* or *Lie-Bäcklund symmetries* [4]. Alternatively,

one weakens the requirement that every solution is mapped into a solution; this yields the so-called *non-classical methods*. In both cases the explicit construction of the symmetries becomes considerably more difficult.

Generalized symmetries are especially of interest for integrable systems [32, 123]. The existence of a *recursion operator* or a *master symmetry* generating an infinite hierarchy of symmetries is a strong indication that the considered system is integrable. Reduction with respect to generalized symmetries is an important tool for the construction of *soliton solutions*. It is also possible to classify nonlinear partial differential equations using these symmetries [73]. Some *MuPAD* packages for symmetries of integrable systems are described in [33].

Non-classical reductions can be understood within the general scheme of augmenting a given differential equation with *differential constraints* [78]. This corresponds to requiring that only some solutions are mapped into solutions, therefore one hopes to find more symmetries (these are sometimes called *weak symmetries*). In this approach the emphasis lies less on group theory but on the theory of over-determined systems of partial differential equations and thus on questions of completion (cf. [100]).

The first non-classical method was developed by Bluman and Cole [8] and uses the invariant surface condition as constraint. Although this leads for many differential equations to new reductions, the drawback is that the determining system becomes nonlinear. The *direct method* of Clarkson and Kruskal [20] tries to reduce a given partial differential equation to a system of ordinary differential equations by constructing a good ansatz; it corresponds to a special case of the method of Bluman and Cole.

The main problem in the method of differential constraints is to find compatible constraints leading to non-trivial reductions. Besides using the invariant surface condition no systematic way has been discovered so far and thus it remains essentially a game of "try and error". For this reason differential constraints have not yet found much attention in applications.

Completion

Most textbooks on differential equations treat only *normal* systems (or systems in *Cauchy-Kowalevsky form*). For ordinary differential equations this implies that one assumes that the equations can be solved for the highest order derivatives. For partial differential equations one must furthermore assume the existence of a distinguished independent variable such that one can solve for its derivatives to obtain the Cauchy-Kowalevsky form. However, in many fields one encounters systems of differential equations which are not normal. A simple example are the determining systems appearing in symmetry analysis which are usually over-determined. Non-normal systems also occur naturally in differential geometry and in theoretical physics (gauge theories).

For a non-normal system it is a priori not clear whether it possesses any solutions. It may happen that the system is inconsistent. This can only be decided after the construction of all *integrability conditions*. These are differential equations satisfied by any solution of the system but which are nevertheless *algebraically* independent of it. While it makes no problem to construct one integrability condition (typically this requires only a cross-derivative), it is not so easy to decide when *all* have been found, as in principle an infinite number of conditions must be checked.

The process of finding all integrability conditions is called *completion* of the differential equation. It results in a *formally integrable system*, as after completion it is straightforward to construct order by order a formal power series solution. Under additional assumptions it is sometimes possible to show the convergence of the series. This leads for analytic equations to existence and uniqueness theorems like the Cartan-Kähler theorem (the well-known Cauchy-Kowalevsky theorem is a special case of it). For non-analytic equations solvability is a much more complicated question due to Lewy type effects [63].

The first systematic approach to the problem of completion was probably provided by the Janet-Riquier theory [55] with the introduction of *passive* systems. Their definition is based on a *ranking* of the derivatives which decides in what order the integrability conditions are constructed. The completion can be done completely automatically only for quasi-linear systems (if all arising integrability conditions are also quasi-linear), as it must be possible to solve for the leading derivative. In this case the resulting passive system is sometimes call a *standard form* [91].

In geometric theories the notion of a passive system is replaced by *involution*. It combines a geometric definition of formal integrability with an algebraic criterion for the termination of the completion. As an intrinsic concept involution requires no coordinate dependent ingredients like a ranking. Involution analysis based on the Cartan-

Kähler theory [14] for exterior systems is discussed from an algorithmic point of view in [48, 49]. A completion algorithm for the jet bundle formalism based on the formal theory of Pommaret [84] was presented in [97].

Completion algorithms are very useful in the symmetry analysis of differential equations. Once a system is either passive or involutive, one can make statements about the size of the solution space [91, 98]. Thus it is possible to compute the size of the symmetry group without explicitly solving the determining system or to determine the loss of generality in a symmetry reduction [99]. One can even compute the abstract structure of the symmetry algebra without solving the determining system [64, 92].

These concepts are closely related to Gröbner bases in commutative algebra. This holds especially for the Janet-Riquier theory where rankings play a similar role as in the definition of a Gröbner basis. Therefore one sometimes find the term *differential Gröbner basis* for a passive system. Integrability conditions arising from cross-derivatives may be considered as "differential S-polynomials". But these analogies acquire a precise meaning only in the context of differential algebra.

There is a one-to-one correspondence between linear systems of partial differential equations in one dependent variable and polynomial ideals. This has lead in commutative algebra to the new concept of an *involutive basis* of an ideal [42]. It is computed using algorithms coming from the completion theory of differential equations, but it is an ordinary (though not reduced) Gröbner basis. In some cases the new algorithms are considerably faster than the classical Buchberger algorithm.

Differential Ideal Theory

Like the differential Galois theory discussed in the next section, differential ideal theory belongs to the field of *differential algebra*. It can be informally described as an attempt "to write differential in front of everything in algebra". Thus it deals with differential rings, differential fields etc. Of course, this requires an algebraic definition of differentiation. In differential algebra any mapping that is linear with respect to addition and satisfies the Leibniz or product rule is called a *derivation*. A differential ring is a commutative ring together with one (or more) derivation.

Differential polynomials arise by adjunction of differential indeterminates to a differential ring. But the ring of differential polynomials is not Noetherian. Adjoining a differential indeterminate corresponds to adjoining infinitely many algebraic indeterminates, as one must introduce all its derivatives as additional, algebraically independent variables. Thus Hilbert's Basis Theorem does not apply.

A *differential ideal* is an ideal which is in addition closed under the derivation of the differential ring. Many of the basic ideas in differential ideal theory can be traced back to Ritt [93]; the most advanced book is still the one by Kolchin [60]. Like in the purely algebraic theory one would like to introduce something like a Gröbner basis. As the ring of differential polynomials is not Noetherian, algorithms along the lines of the Buchberger algorithm do not terminate in general [18]. This is related to the fact that the *ideal membership problem* is undecidable for arbitrary differential ideals [35]. However, this result is more of theoretical interest, as for finitely generated ideals the decidability is still an open question. In any case one must say that so far no generally accepted definition of a differential Gröbner basis has emerged.

There exist two basic strategies to circumvent this principal problem. One can either restrict to special classes of ideals where a proof of termination is possible or one weakens the properties expected of a differential Gröbner basis. The completion algorithm of the Janet-Riquier theory can be considered as a simple example for the first strategy. An example for the second one are the bases introduced by Mansfield [70]. They are computed with pseudo-reductions and have thus weaker properties than their algebraic counterpart. Especially, it may happen that one leaves the ideal.

Recently, Boulier *et al.* [10] presented a *Rosenfeld-Gröbner algorithm* which computes a representation for the radical ideal of a finitely generated differential ideal in the following form. The radical is written as a finite intersection of saturations ideals; these are radical differential ideals defined by a system of differential polynomial equations and inequalities. This representation allows for an easy algorithmic test of radical ideal membership and for computing formal power series solutions.

Open problems are to obtain a minimal decomposition, i. e. to use only a minimal number of saturation ideals,

and to find bases for these ideals (avoiding the inequalities). These questions are closely related to the inclusion problem for differential ideals which in turn can be seen as the problem of determining the relation between the singular and the general solutions of a differential equation. The principal obstacle in the construction of the bases is a very typical one in differential algebra. A theorem of Ritt asserts that by taking sufficiently many derivatives of the equations one can always get a basis but no bound for the number of derivatives needed is known.

Differential algebra is applied in *automatic theorem proving* in differential geometry [126]. This is similar to the use of algebraic ideal theory in theorem proving in elementary geometry. For this kind of applications *characteristic sets* seem to be more useful than Gröbner bases. A nice example for the possibilities here is the automatic derivation of Newton's law of gravity from the three Kepler laws [125].

Besides ideals of differential polynomials there has also been some work on ideals of linear differential operators or ideals of the Weil algebra [34]. However, here one is dealing with non-commutative rings. One could also consider the Cartan-Kähler theory as a kind of differential ideal theory, as it represents differential equations by closed ideals of differential forms.

Differential Galois Theory

Already Lie was looking for a differential analog of the (algebraic) Galois theory, when he introduced Lie groups. What is nowadays usually called differential Galois theory [66, 105] has however no connection to Lie symmetry theory. The latter one uses continuous transformation groups and can be applied to any differential equations. But as discussed above it is not completely algorithmic. The former one is based on linear algebraic groups. It considers exclusively linear ordinary differential equations and culminates in various algorithms for explicitly computing Liouvillian solutions.

Determining the solutions of linear differential equations is a very classical topic and many famous mathematicians like Liouville, Fuchs, Klein or Jordan studied it in the last century and their results are still very important for the design of algorithms. Differential Galois theory was essentially founded by Picard and Vessiot and given its modern form by Kolchin [60]. Pommaret [85] developed an alternative theory following more closely Lie's ideas and using the formal theory.

Liouvillian functions comprise essentially all expressions one can “easily write down”. Allowed operations are the usual arithmetic ones, roots, exponentials, logarithms, integrals and algebraic functions. A more formal definition uses a tower of simple extensions of the field of rational functions. An important point is that for any Liouvillian function one needs only a finite number of extensions, thus it is algorithmically constructible. Most expressions one would call “closed-form” are in fact Liouvillian.

Most solution algorithms are based on the seminal work of Singer [102]. He showed that the logarithmic derivative of any Liouvillian solution is algebraic and determined an a priori bound for the degree of the minimal polynomial, namely the Jordan bound for the index of an Abelian normal subgroup of a finite linear group. In principle, this suffices to determine all Liouvillian solutions, but the bound grows rapidly with the order of the equations leading thus to a very high complexity of the algorithm.

Using the representation theory of finite groups Ulmer [114] could significantly improve the bound given by Singer, so that at least the treatment of equations up to third order seems feasible, but there does not yet exist an implementation. Group theory yields also a number of other interesting results like criteria for the existence (and number) of algebraic solutions (the solutions which are most expensive to determine belong to this class) and gives the basic case distinctions in the solution algorithms.

The original work of Singer covered only equations with rational coefficients. Later, he extended it to Liouvillian coefficients [12, 106]. For second order equations Kovacic [57, 28] developed independently a solution algorithm. Only much later it could be shown that the classification underlying this algorithm can also be derived within the Singer theory [108]. The Kovacic algorithm has been implemented in several computer algebra systems.

An alternative approach based on the *invariant ring* of the differential Galois group was presented by Fakler [31] following ideas going back to Fuchs (see also the work of Singer and Ulmer [108] and van Hoeij and Weil [116], respectively). For second order equations there exists an isomorphism between the invariant ring and the rational solutions of some symmetric power of the differential equation. This isomorphism allows one to derive explicit

solution formulae and thus a rather efficient algorithm.[4]
The determination of the *differential Galois group* for a given equation is rather difficult. Some progress has recently been made for second and third order equations [107] where it could be reduced to the problem of finding solutions of some associated linear differential equations in the coefficient field and of factoring such equations. If there was an easy way to compute the group directly, one could probably design more efficient solution algorithms. But currently it is the other way round: the solution algorithms help finding the Galois group.
There has also been some work on the *inverse problem* of differential Galois theory. Here a linear algebraic group is given and the task is to determine a differential equation that has it as differential Galois group. One can prove that such a differential equation always exists [74]. Ramis [89] showed that it is often possible to reduce the inverse problem to the direct problem and then even give an explicit solution.
All the theory mentioned here works only for *irreducible* equations. Thus the (efficient) *factorization* of linear differential operators is an important problem in differential Galois theory. A solution of this problem based on the Newton polygon was recently presented by van Hoeij [115]. Factorization (although only of polynomials) is also an issue in differential ideal theory.
Differential Galois theory also gives algorithms for the construction of (Liouvillian) *first integrals* [69, 88, 120]. These can be used to construct explicit solutions [68]. Other applications appear in the theory of completely integrable systems. Ziglin has given an algebraic characterization of such systems based on their monodromy group. His criterion for integrability can be rephrased in terms of certain properties of the differential Galois group [19, 75].

Dynamical Systems

Applications in dynamical systems theory are not in the main stream of computer algebra. Conversely, numerical computations play a much more prominent role in dynamical systems theory than symbolic ones. Nevertheless, the use of computer algebra systems is becoming more and more popular in this field, especially for perturbation analysis [90]. The mainly used feature is the ability to expand functions into series and to manipulate these.
But also classical algebraic problems are of great importance in the study of dynamical systems. For example, before a fixed point can be analyzed it must be determined. This requires the solution of a nonlinear system of algebraic equations. If the vector field is rational, this can be done with Gröbner bases. Often the vector field depends on some parameters. At certain values of these parameters, the properties of the vector field may change, i. e. a *bifurcation* occurs. The determination of these values is a fundamental problem in dynamical systems theory.
Of special interest are here *equivariant systems*, i. e. systems for which the defining vector field is invariant under the action of a symmetry group [44, 45]. In this case one can use linear representation theory and polynomial invariant theory for determining the fundamental invariants and equivariants [38, 124]. Using normal forms they enable the local bifurcation analysis, i. e. the typical bifurcation diagram in the neighborhood of a critical point can be derived.
A typical problem in computer algebra is the determination of *normal forms*. For dynamical systems such normal forms have already been introduced by Poincaré, Birkhoff, Gustavson and many others, often in the context of celestial mechanics [13, 24]. They form the basis for the solution of many problems in dynamical systems theory like for example stability or bifurcation analysis. The main idea behind normal forms is to study the system in the neighborhood of a *fixed point* (or *equilibrium*) and to try to remove by a coordinate transformation as many nonlinear terms from the differential equation as possible. According to the Hartman-Grobman theorem all such terms can be eliminated near a *hyperbolic fixed point* where the Jacobian has no zero or purely imaginary eigenvalues. Thus at a hyperbolic point linear stability theory is sufficient.
Around other types of equilibria like centers the analysis is more involved. This concerns especially Hamiltonian systems where stable fixed points can never be hyperbolic. If there are *resonances* between the eigenvalues of the Jacobian, the normal form is necessarily non-linear. In order to determine a normal form one makes a power series ansatz for the coordinate transformation and determines the coefficients of the ansatz by requiring that besides

[4] A *MuPAD* implementation of this approach (and some related code) can be down-loaded from Fakler's WWW page with the URL `http://iaks-www.ira.uka.de/home/fakler/index.html`.

the resonances all non-linear terms of the differential equations up to a certain order disappear. The resulting transformed differential equations are normal forms.

An algorithm for computing normal forms of a given system that is suitable for implementation in a computer algebra system was presented by Walcher [119]. It is closely related to *Lie transforms* [72]. This technique has its origin in Hamiltonian mechanics where it yields a canonical transformation. However, it can be extended to general dynamical systems. In contrast to this Birkhoff normal form Gatermann and Lauterbach [40] use normal forms from singularity theory in order to study bifurcation phenomena. For equivariant systems they automatically classify them using Gröbner bases.

Another application is the determination of *center manifolds* [17], a special form of invariant manifolds. If a system possesses a center manifold, it often suffices to study the behavior of the system on this manifold. For example, if the zero solution of the reduced system is stable, then solutions of the original system for initial data sufficiently close to the center manifold will approach this manifold exponentially fast. Thus the reduced system completely describes the asymptotic behavior of such solutions.

The main point in using center manifold theory is that it yields a reduction of the dimension and thus typically a considerable simplification of the analysis. Often it is possible to reduce an infinite-dimensional problem to a finite-dimensional one. There are two important computational steps in the application of the theory. First we need an approximation for the center manifold, then we must compute the reduced system. As in normal form theory, this is done step by step with a power series ansatz. Laure and Demay [61] showed for the Couette-Taylor problem how computer algebra and numerical analysis can interact to solve a complicated bifurcation problem for an infinite-dimensional problem using a reduction to a finite-dimensional center manifold.

A more theoretical application of computer algebra concerns *Hilbert's 16th problem* of bounding the number of limit cycles in a planar polynomial system. For quadratic systems a lot of results are known [96]; however already the cubic case becomes very complicated. An important subproblem is the *center problem*, namely to distinguish between a focus and a center. The derivation of sufficient and especially of necessary conditions for a center can be very involved and is sometimes hardly feasible without computer algebra [80]. In a recent study of cubic systems [29] a CRAY-J90 had to be used.

Numerical Analysis

It was already mentioned above that the capabilities of computer algebra systems to explicitly solve differential equations are limited. This holds especially for partial differential equations. Therefore numerical methods have lost nothing of their importance. Symbolic and numerical computations can interact in many ways and most computer algebra systems provide some numerical facilities.

The oldest and simplest approach consists of *interfacing* a computer algebra system and a numerical library. Typically the interaction is one-way: the computer algebra system is used to derive the differential equations (e. g. the equations of motion of a complicated physical system); the interface generates code in the language of the numerical library (perhaps including some optimization steps); finally, the differential equations are solved with some methods from the numerical library.

To some extend most of the common computer algebra systems can do this, as they all provide commands to convert an expression into C or FORTRAN. However, it is rather cumbersome to automatically generate whole programs that way. For such purposes one better uses a specialized package like GENTRAN [41] which is part of the REDUCE distribution. Another problem is the optimization of the generated code which is usually necessary. In REDUCE the package SCOPE [117] can do this.

MuPAD provides a very efficient form of interfacing: *dynamical modules* [110]. These are developed in a language like C or C++ and can be linked to *MuPAD* dynamically at run-time. Compared with approaches based on interprocess communication this leads to much less overhead. As the module has direct access to the internal data of the *MuPAD* session, much less data must be communicated. The procedures implemented in a dynamical module can be called within *MuPAD* like any other function. Provided a convenient interface exists this allows in principle to work within *MuPAD* interactively with a numerical library.[5]

[5] A prototypical implementation of an interface to some functions of the NAG C-library is discussed on the *MuPAD* web site at the URL

Computer algebra systems can also help to *select* an appropriate method from a numerical library. Modern libraries have reached such a level of sophistication that for many users it is impossible to fully exploit their potential. They provide many different routines for the same task and their working can be further tuned by many input parameters whose meaning remains a secret for non-experts. A computer algebra system can analyze the given differential equation (e. g. estimate its stiffness) and then choose an appropriate method and determine reasonable values for its parameters. An example for this approach is the AXIOM package ANNA developed by Dupée [27].

Goldman *et al.* [43] go considerably further in their application of computer algebra in numerical analysis by using it as a software engineering tool. They automatically *generate* the full FORTRAN code for numerically solving the Navier-Stokes equations. Their argument is that such programs are so long and complicated that their maintenance and adaption (new boundary conditions, different discretizations etc) is rather difficult and error-prone. They use instead a number of input files that contain all the relevant information about the problem in a format that is comparatively easy to read and let the computer algebra system then generate the source code.

One can also use computer algebra to *derive* numerical schemes. The Butcher theory of Runge-Kutta methods is here a typical example. For higher order methods the order conditions become rather large and complicated. Computer algebra packages have been developed that derive and solve them (using Gröbner bases) [46, 109].[6] For partial differential equations the construction of higher-order discretizations or finite elements can be rather involved and is sometimes only feasible with the help of a computer algebra system [76]. Computer algebra can also assist in proving the stability of finite difference schemes [36].

Another topic where computer algebra plays a certain role in numerical analysis are *differential algebraic equations*. The *index* of such a system comprising differential and algebraic equations measures in a certain sense, how far it is away from a pure differential equation [15]. This gives an indication of the difficulties one must expect in a numerical integration. The determination of the index is essentially equivalent to the completion procedures described above [62, 83, 101], as it can be defined as the number of steps needed for the completion. However, in practice numerical analysts often prefer the use of *automatic differentiation* to computer algebra [16].

None of the applications described so far represents really what one would call a *hybrid* algorithm combining symbolic and numerical elements, i. e. where computer algebra is an integral part of the solution process and not only used to determine either the problem or the numerical method for its solution. However, we are not aware of any such algorithm for differential equations, although some ansätze based on symmetry theory have been developed. Dorotnitsyn [26] showed how one may construct finite difference schemes inheriting the symmetries of a differential equation. Such schemes should very well preserve the associated conservation laws, but so far no numerical tests have been published.

In contrast, for solving algebraic equations several hybrid algorithms have been designed. One of them deals with nonlinear systems possessing symmetries [39], as they e. g. arise in equivariant systems. In the symbolic part it uses the linear representation theory of finite groups to transform the problem into an optimal form for the numerical part. This includes for example a block diagonalization of the Jacobian. The numerical algorithm is complicated due to the underlying group theory. Gatermann [37] showed how the numerical computations can be automated by first computing the necessary group theoretic data which is summarized in a bifurcation graph.

Conclusions and Outlook

The application of computer algebra to differential equations is a vast (and not very well defined) field. We could only briefly indicate some of the main research directions and had to omit many others. For example, one can extend the idea of transforming differential equations far beyond simple heuristics and is then lead to the *equivalence problem of Cartan* [59, 79]. Within the algebraic approaches we ignored the theory of $\mathcal{D}$-modules [67] which is of considerable importance in control theory.

The fields we have touched on are in rather different states. Some of them like symmetry theory are meanwhile fairly mature with the fundamentals well understood and they provide standard techniques for tackling differential

`http://www.mupad.de/PAPERS/MODULES/NAGC/index.html`

[6] Actually, the problem of automatically generating the order conditions for Runge-Kutta methods was mentioned by Jenks [56] in the problem section of the SIGSAM Bulletin already in 1976.

equations implemented in many computer algebra systems. Others are still in an early stage of their development and essential questions are open. Such fields are usually known only to some experts and only prototypical implementations of algorithms exist.

One common feature shared by most of the fields is the complexity of the algorithms. If we take the various completion methods as example, it is obvious from their close relation to Gröbner bases that their complexity is at least as bad as that of the Buchberger algorithm. Although Gröbner bases solve *in principle* many problems in commutative algebra, it is well-known that one often fails to get a basis in reasonable time. One possible way out is the stronger use of heuristics and techniques from Artificial Intelligence, although this is an unpleasant thought for many pure mathematicians.

Some readers might be surprised that we discussed applications in numerical analysis as broadly as more traditional topics like symmetry theory. But we believe that in the future this direction will be among the most important ones. Despite all the successes of Lie symmetries, differential Galois theory etc. one must clearly see that these theories are of hardly any value for many of the problems an engineer, say, typically faces. A popular benchmark problem for numerical methods for differential algebraic equations comes from vehicle dynamics and models with five links a wheel suspension [52]. Its equations of motion must be generated by computer and consist of 7000 lines of FORTRAN code. It appears hardly realistic to solve such a system with Lie symmetries (if it possesses any!) or any other analytic technique.

This does not imply that there is no point in further studying symbolic methods, not at all! Toy models that can be solved analytically are important for obtaining a deeper understanding of underlying structures. One may hope that such understanding may lead to more efficient numerical algorithms for such large problems. And again we want to stress that the application of computer algebra to differential equations is not restricted to solving them!

Most of the current numerical methods for ordinary differential equations do not take into account any special properties of the equation (except its stiffness). It is a rather new trend in numerical analysis to try to use such properties for the design of better algorithms. One prominent example of this trend are *symplectic integrators* [94] which are superior to most conventional methods in the long term integration of Hamiltonian systems, as they preserve many *qualitative* features of such systems. One might expect that computer algebra will play an important role in this emerging field.

The combination of symbolic and numerical computation will become much more important in the future. In the form of simple interfaces it happens already now in many places and hopefully we can soon add powerful hybrid methods. For most *users* of computer algebra systems (this is a very different community than the participants of computer algebra conferences!) such possibilities are of much greater importance than many of the fancy algorithms developed by theorists.

Acknowledgments

I thank F. Postel for inviting me to write this article. J. Calmet, W. Fakler, K. Gatermann, V. Gerdt, E. Hubert and E. Pflügel read parts of various draft versions and gave valuable comments. This work was supported by Deutsche Forschungsgemeinschaft.

References

[1] O. Aberth. The failure in computable analysis of a classical existence theorem for differential equations. *Proc. Amer. Math. Soc.*, 30:151–156, 1971.

[2] M.J.D Ablowitz, A. Ramani, and H. Segur. Nonlinear evolution equations and ordinary differential equations of Painlevé type. *Lett. Nuovo Cim.*, 23:333–338, 1978.

[3] S.A. Abramov and K.Yu. Kvansenko. Fast algorithms to search for the rational solutions of linear differential equations with polynomial coefficients. In S.M. Watt, editor, *Proc. ISSAC '91*, pages 267–270. ACM Press, New York, 1991.

[4] R.L. Anderson and N.H. Ibragimov. *Lie-Bäcklund Transformations in Applications*. SIAM, Philadelphia, 1979.

[5] M.A. Barkatou. A rational version of Moser's algorithm. In A.H.M. Levelt, editor, *Proc. ISSAC '95*, pages 297–302. ACM Press, New York, 1995.

[6] M.A. Barkatou. An algorithm to compute the exponential part of a formal fundamental matrix solution of a linear differential system. *Appl. Alg. Eng. Commun. Comput.*, 8:1–23, 1997.

[7] O. Becken. Algorithmen zum Lösen einfacher Differentialgleichungen. *Rostocker Informatik Berichte*, 17:1–24, 1995.

[8] G.W. Bluman and J.D. Cole. The general similarity solution of the heat equation. *J. Math. Mech.*, 18:1025–1042, 1969.

[9] G.W. Bluman and S. Kumei. *Symmetries and Differential Equations*. Applied Mathematical Sciences 81. Springer-Verlag, New York, 1989.

[10] F. Boulier, D. Lazard, F. Ollivier, and M. Petitot. Representation for the radical of a finitely generated differential ideal. In A.H.M. Levelt, editor, *Proc. ISSAC '95*, pages 158–166. ACM Press, New York, 1995.

[11] M. Bronstein. Linear ordinary differential equations: breaking through the order 2 barrier. In B.M. Trager and D. Lazard, editors, *Proc. ISSAC '92*, pages 42–48. ACM Press, New York, 1992.

[12] M. Bronstein. On solutions of linear ordinary differential equations in their coefficient field. *J. Symb. Comp.*, 13:413–439, 1992.

[13] A.D. Bruno. *Local Methods in Nonlinear Differential Equations*. Springer-Verlag, Berlin, 1989.

[14] R.L. Bryant, S.S. Chern, R.B. Gardner, H.L. Goldschmidt, and P.A. Griffiths. *Exterior Differential Systems*. Mathematical Sciences Research Institute Publications 18. Springer-Verlag, New York, 1991.

[15] S.L. Campbell and C.W. Gear. The index of general nonlinear DAEs. *Numer. Math.*, 72:173–196, 1995.

[16] S.L. Campbell and R. Hollenbeck. Automatic differentiation and implicit differential equations. In M. Berz, C. Bischof, G. Corliss, and A. Griewank, editors, *Proc. Second Int. Workshop Computational Differentiation*, pages 215–227. SIAM, Philadelphia, 1996.

[17] J. Carr. *Applications of Centre Manifold Theory*. Applied Mathematical Sciences 35. Springer-Verlag, New York, 1981.

[18] G. Carra Ferro. Gröbner bases and differential algebra. In L. Huguet and A. Poli, editors, *Proc. AAECC-5*, Lecture Notes in Computer Science 350, pages 129–140. Springer-Verlag, Berlin, 1987.

[19] R.C. Churchill and D.L. Rod. On the determination of Ziglin monodromy groups. *SIAM J. Math. Anal.*, 22:1790–1802, 1991.

[20] P.A. Clarkson and M.D. Kruskal. New similarity reductions of the Boussinesq equation. *J. Math. Phys.*, 30:2201–2212, 1989.

[21] R. Conte. Singularities of differential equations and integrability. In D. Benest and C. Froeschlé, editors, *Introduction to Methods of Complex Analysis and Geometry for Classical Mechanics and Non-Linear Waves*, pages 49–143. Editions Frontieres, Gif-sur-Yvette, 1994.

[22] R. Conte, A.P. Fordy, and A. Pickering. A perturbative Painlevé approach to nonlinear differential equations. *Physica D*, 69:33–58, 1993.

[23] J. Denef and L. Lipshitz. Power series solutions of algebraic differential equations. *Math. Ann.*, 267:213–238, 1984.

[24] J. Della Dora and L. Stolovitch. Normal forms of differential systems. In E. Tournier, editor, *Computer Algebra and Differential Equations*, London Mathematical Society Lecture Note Series 193, pages 143–184. Cambridge University Press, 1994.

[25] J. Della Dora and E. Tournier. Formal solutions of differential equations in the neighbourhood of singular points. In P.S. Wang, editor, *Proc. SYMSAC*, pages 25–29. ACM Press, New York, 1981.

[26] V.A. Dorodnitsyn. Finite difference models entirely inheriting symmetry of original differential equations. In N.H. Ibragimov, M. Torrisi, and A. Valenti, editors, *Modern Group Analysis: Advanced Analytical and Computational Methods in Mathematical Physics*, pages 191–201. Kluwer, Dordrecht, 1993.

[27] B.J. Dupée and J.H. Davenport. An intelligent interface to numerical routines. In J. Calmet and C. Limongelli, editors, *Proc. DISCO '96*, Lecture Notes in Computer Science 1128, pages 252–262. Springer, Heidelberg, 1996.

[28] A. Duval and M. Loday-Richaud. Kovacic's algorithm and its application to some families of special functions. *Appl. Alg. Eng. Commun. Comput.*, 3:211–246, 1992.

[29] V.F. Edneral. Computer evaluation of cyclicity in planar cubic system. In W.W. Küchlin, editor, *Proc. ISSAC '97*, pages 305–309. ACM Press, New York, 1997.

[30] P.G. Estévez. Non-classical symmetries and the singular manifold method: the Burgers and the Burgers-Huxley equations. *J. Phys. A*, 27:2113–2127, 1994.

[31] W. Fakler. On second order homogeneous linear ordinary differential equations with Liouvillian solutions. *Theor. Comp. Sci.*, to appear.

[32] A.S. Fokas. Symmetries and integrability. *Stud. Appl. Math.*, 77:253–299, 1987.

[33] B. Fuchssteiner, S. Ivanov, and W. Wiwianka. Algorithmic determination of infinite dimensional symmetry groups for integrable systems in 1+1 dimensions. *Math. Comp. Model.*, 25:91–100, 1997.

[34] A. Galligo. Some algorithmic questions on ideals of differential operators. In B.F. Caviness, editor, *Proc. EUROCAL '85, vol. 2*, Lecture Notes in Computer Science 204, pages 413–421. Springer-Verlag, Berlin, 1985.

[35] G. Gallo, B. Mishra, and F. Ollivier. Some constructions in rings of differential polynomials. In H.F. Mattson, T. Mora, and T.R.N. Rao, editors, *Proc. AAECC-9*, Lecture Notes in Computer Science 539, pages 171–182. Springer-Verlag, Heidelberg, 1991.

[36] V.G. Ganzha and R. Liska. Application of the Reduce computer algebra system to stability analysis of difference schemes. In E. Kaltofen and S.M. Watt, editors, *Computers and Mathematics*, pages 119–129. Springer, New York, 1989.

[37] K. Gatermann. Computation of bifurcation graphs. In E. Allgower, K. Georg, and R. Miranda, editors, *Exploiting Symmetry in Applied and Numerical Analysis*, AMS Lectures in Applied Mathematics 29, pages 187–201. AMS, Providence, 1993.

[38] K. Gatermann. Semi-invariants, equivariants and algorithms. *Appl. Alg. Eng. Commun. Comput.*, 7:105–124, 1996.

[39] K. Gatermann and A. Hohmann. Symbolic exploitation of symmetry in numerical pathfollowing. *Impact Comp. Sci. Eng.*, 3:330–365, 1991.

[40] K. Gatermann and R. Lauterbach. Automatic classification of normal forms. *Nonlin. Anal.*, to appear.

[41] B.L. Gates. A numerical code generation facility for REDUCE. In B.W. Char, editor, *Proc. SYMSAC '86*, pages 94–99. ACM Press, New York, 1986.

[42] V.P. Gerdt. Gröbner bases and involutive methods for algebraic and differential equations. In J. Fleischer, J. Grabmeier, F.W. Hehl, and W. Küchlin, editors, *Computer Algebra in Science and Engineering*, pages 117–137. World Scientific, Singapore, 1995.

[43] V.V. Goldman, J.A. van Hulzen, A.E. Mynett, A.S. Posthuma, and H.J. van Zuylen. The application of computer algebra for the discretization and coding of the Navier-Stokes equations. In A.M. Cohen, L. van Gastel, and S. Verduyn Lionel, editors, *Computer Algebra in Industry 2*, pages 131–150. John Wiley, New York, 1995.

[44] M. Golubitsky and D.G. Schaeffer. *Singularities and Groups in Bifurcation Theory I.* Applied Mathematical Sciences 51. Springer-Verlag, New York, 1985.

[45] M. Golubitsky, D.G. Schaeffer, and I. Stuart. *Singularities and Groups in Bifurcation Theory II.* Applied Mathematical Sciences 69. Springer-Verlag, New York, 1988.

[46] D. Gruntz. Symbolic computation of explicit Runge-Kutta formulas. In W. Gander and J. Hrebffcek, editors, *Solving Problems in Scientific Computing Using Maple and MATLAB*, chapter 19, pages 267–283. Springer-Verlag, Berlin, 1995.

[47] B.K. Harrison and F.B. Estabrook. Geometric approach to invariance groups and solutions of partial differential systems. *J. Math. Phys.*, 12:653–666, 1971.

[48] D.H. Hartley. Involution analysis for non-linear exterior differential systems. *Math. Comp. Model.*, 25:51–62, 1997.

[49] D.H. Hartley and R.W. Tucker. A constructive implementation of the Cartan-Kähler theory of exterior differential systems. *J. Symb. Comp.*, 12:655–667, 1991.

[50] W. Hereman. Review of symbolic software for the computation of Lie symmetries of differential equations. *Euromath Bull.*, 2:45–82, 1994.

[51] W. Hereman. Symbolic software for Lie symmetry analysis. In N.H. Ibragimov, editor, *CRC Handbook of Lie Group Analysis of Differential Equations, Volume 3: New Trends in Theoretical Development and Computational Methods*, chapter 13. CRC Press, Boca Raton, Florida, 1995.

[52] M. Hiller and S. Frik. Road vehicle benchmark 2: Five link suspension. In W. Kortüm, S. Sharp, and A. de Pater, editors, *Application of Multibody Computer Codes to Vehicle System Dynamics, Progress Report to the 12th IAVSD Symposium, Lyon 1991*, pages 198–203, 1992.

[53] E. Hubert. The general solution of an ordinary differential equation. In Y.N. Lakshman, editor, *Proc. ISSAC '96*, pages 189–195. ACM Press, New York, 1996.

[54] E.L. Ince. *Ordinary Differential Equations*. Dover, New York, 1956.

[55] M. Janet. Sur les Systemes d'Équations aux Dérivées Partielles. *J. Math. Pure Appl.*, 3:65–151, 1920.

[56] R.J. Jenks. Problem #11: Generation of Runge-Kutta equations. *SIGSAM Bull.*, 10(1):6–7, 1976.

[57] J.J. Kovacic. An algorithm for solving second order linear homogeneous differential equations. *J. Symb. Comp.*, 2:3–43, 1986.

[58] E. Kamke. *Differentialgleichungen: Lösungsmethoden und Lösungen*, volume 1. Geest & Portig, Leipzig, 1967.

[59] N. Kamran. The equivalence problem of Elie Cartan, differential equations and computer algebra. In E. Tournier, editor, *Computer Algebra and Differential Equations*, pages 87–114. Academic Press, London, 1988.

[60] E.R. Kolchin. *Differential Algebra and Algebraic Groups*. Academic Press, New York, 1973.

[61] P. Laure and Y. Demay. Symbolic computation and equation on the center manifold: Application to the Couette-Taylor problem. *Comp. Fluids*, 16:229–238, 1988.

[62] G. Le Vey. Differential algebraic equations: A new look at the index. Rapport de Recherche 2239, INRIA Rennes, 1994.

[63] H. Lewy. An example of a smooth linear partial differential equation without solution. *Ann. Math.*, 66:155–158, 1957.

[64] I.G. Lisle, G.J. Reid, and A. Boulton. Algorithmic determination of structure of infinite Lie pseudogroups of symmetries of PDEs. In A.H.M. Levelt, editor, *Proc. ISSAC '95*, pages 1–6. ACM Press, New York, 1995.

[65] M.A.H. MacCallum. Using computer algebra to solve ordinary differential equations. In A.M. Cohen, L. van Gastel, and S.M. Verduyn Lunel, editors, *Computer Algebra in Industry 2*, pages 19–41. John Wiley, New York, 1995.

[66] A.R. Magid. *Lectures on Differential Galois Theory*. University Lecture Series 7. AMS, Providence, 1994.

[67] B. Malgrange. Motivation and introduction to the theory of d-modules. In E. Tournier, editor, *Computer Algebra and Differential Equations*, London Mathematical Society Lecture Note Series 193, pages 3–20. Cambridge University Press, 1994.

[68] Y.-K. Man. Computing closed form solutions of first order ODEs using the Prelle-Singer procedure. *J. Symb. Comp.*, 16:423–443, 1993.

[69] Y.-K. Man and M.A.H. MacCallum. A rational approach to the Prelle-Singer algorithm. *J. Symb. Comp.*, 24:31–43, 1997.

[70] E. Mansfield. *Differential Gröbner Bases*. PhD thesis, Macquarie University, Sydney, 1991.

[71] J.B. McLeod and P.J. Olver. The connection between partial differential equations soluble by inverse scattering and ordinary differential equations of Painlevé type. *SIAM J. Math. Anal.*, 14:488–506, 1983.

[72] K.R. Meyer. Lie transform tutorial II. In K.R. Meyer and D.S. Schmidt, editors, *Computer Aided Proofs in Analysis*, IMA Volumes in Mathematics and its Applications 28, pages 190–210. Springer-Verlag, New York, 1991.

[73] A.V. Mikhailov, A.B. Shabat, and R.I. Yamilov. The symmetry approach to the classification of non-linear equations. Complete lists of integrable systems. *Russ. Math. Surv.*, 42:3–53, 1987.

[74] C. Mitschi and M.F. Singer. The inverse problem in differential Galois theory. In B.L.J. Braaksma, G.K. Immink, and M. van der Put, editors, *Proc. The Stokes Phenomenon and Hilbert's 16th Problem*, pages 185–196. World Scientific, New York, 1996.

[75] J.J. Morales and C. Simo. Picard-Vessiot theory and Ziglin's theorem. *J. Diff. Eqs.*, 107:140–162, 1994.

[76] E.H. Mund. Computer algebra and finite element methods in engineering. In A.M. Cohen, L. van Gastel, and S. Verduyn Lionel, editors, *Computer Algebra in Industry 2*, pages 81–100. John Wiley, New York, 1995.

[77] P.J. Olver. *Applications of Lie Groups to Differential Equations*. Graduate Texts in Mathematics 107. Springer-Verlag, New York, 1986.

[78] P.J. Olver. Direct reduction and differential constraints. *Proc. Roy. Soc. Ser. A*, 444:509–523, 1994.

[79] P.J. Olver. *Equivalence, Invariants, and Symmetry*. Cambridge University Press, 1995.

[80] J.M. Pearson, N.G. Lloyd, and C.J. Christopher. Algorithmic derivation of centre conditions. *SIAM Rev.*, 38:619–636, 1996.

[81] E. Pflügel. An algorithm for computing exponential solutions of first order linear differential systems. In W.W. Küchlin, editor, *Proc. ISSAC '97*, pages 164–171. ACM Press, New York, 1997.

[82] E. Pflügel. On the latest version of DESIR. *Theor. Comp. Sci.*, to appear.

[83] O.-P. Piirilä and J. Tuomela. Differential-algebraic systems and formal integrability. Research Report A326, Helsinki University of Technology, Institute of Mathematics, 1993.

[84] J.F. Pommaret. *Systems of Partial Differential Equations and Lie Pseudogroups*. Gordon & Breach, London, 1978.

[85] J.F. Pommaret. *Differential Galois Theory*. Gordon & Breach, London, 1983.

[86] F. Postel and P. Zimmermann. A review of the ODE solvers of Maple, Mathematica, Macsyma and MuPAD. In A. Carriére and L.R. Oudin, editors, *Proc. 5th Rhine Workshop on Computer Algebra*, pages 2.1–2.10. Report PR 801/96, ISL, Saint-Louis (France), 1996.

[87] M.B. Pour-el and J.I. Richards. A computable ordinary differential equation which possesses no computable solution. *Ann. Math. Logic*, 17:61–90, 1979.

[88] M.J. Prelle and M.F. Singer. Elementary first integrals of differential equations. *Trans. Amer. Math. Soc.*, 139:167–189, 1969.

[89] J.P. Ramis. About the solution of some inverse problems in differential Galois theory by Hamburger equations. In K.D. Elworthy and L. Markus, editors, *Differential Equations, Dynamical Systems, and Control Science. A Festschrift in Honor of Lawrence Markus*, Lecture Notes in Pure and Applied Mathematics 152, pages 277–299. Marcel Dekker, New York, 1994.

[90] R.H. Rand and D. Armbruster. *Perturbation Methods, Bifurcation Theory and Computer Algebra*. Applied Mathematical Sciences 65. Springer-Verlag, New York, 1987.

[91] G.J. Reid. Algorithms for reducing a system of PDEs to standard form, determining the dimension of its solution space and calculating its Taylor series solution. *Eur. J. Appl. Math.*, 2:293–318, 1991.

[92] G.J. Reid, I.G. Lisle, A. Boulton, and A.D. Wittkopf. Algorithmic determination of commutation relations for Lie symmetry algebras of PDEs. In B.M. Trager and D. Lazard, editors, *Proc. ISSAC '92*, pages 63–68. ACM Press, New York, 1992.

[93] J.F. Ritt. *Differential Algebra*. Dover, New York, 1966. (Original: AMS Colloquium Publications, Vol. XXXIII, 1950).

[94] J.M. Sanz-Serna and M.P. Calvo. *Numerical Hamiltonian Problems*. Applied Mathematics and Mathematical Computation 7. Chapman & Hall, London, 1994.

[95] C. Scheen. Implementation of the Painlevé test for ordinary differential equations. *Theor. Comp. Sci.*, to appear.

[96] D. Schlomiuk, J. Guckenheimer, and R. Rand. Integrability of plane quadratic vector fields. *Expo. Math.*, 8:3–25, 1990.

[97] J. Schü, W.M. Seiler, and J. Calmet. Algorithmic methods for Lie pseudogroups. In N. Ibragimov, M. Torrisi, and A. Valenti, editors, *Proc. Modern Group Analysis: Advanced Analytical and Computational Methods in Mathematical Physics*, pages 337–344. Kluwer, Dordrecht, 1993.

[98] W.M. Seiler. On the arbitrariness of the general solution of an involutive partial differential equation. *J. Math. Phys.*, 35:486–498, 1994.

[99] W.M. Seiler. Arbitrariness of the general solution and symmetries. *Acta Appl. Math.*, 41:311–322, 1995.

[100] W.M. Seiler. Involution and symmetry reductions. *Math. Comp. Model.*, 25:63–73, 1997.

[101] W.M. Seiler. Numerical analysis of constrained Hamiltonian systems and the formal theory of differential equations. *Math. Comp. Simul.*, to appear, 1997.

[102] M.F. Singer. Liouvillian solutions of n-th order homogeneous linear differential equations. *Am. J. Math.*, 103:661–682, 1981.

[103] M.F. Singer, editor. *Differential Equations and Computer Algebra (CADE-90)*, Computational Mathematics and Applications, London, 1990. Academic Press.

[104] M.F. Singer. Formal solutions of differential equations. *J. Symb. Comp.*, 10:59–94, 1990.

[105] M.F. Singer. An outline of differential Galois theory. In E. Tournier, editor, *Computer Algebra and Differential Equations*. Academic Press, New York, 1990.

[106] M.F. Singer. Liouvillian solutions of linear differential equations with Liouvillian coefficients. *J. Symb. Comp.*, 11:251–273, 1991.

[107] M.F. Singer and F. Ulmer. Galois groups of second and third order linear differential equations. *J. Symb. Comp.*, 16:9–36, 1993.

[108] M.F. Singer and F. Ulmer. Liouvillian and algebraic solutions of second and third order linear differential equations. *J. Symb. Comp.*, 16:37–73, 1993.

[109] M. Sofroniou. Symbolic derivation of Runge-Kutta methods. *J. Symb. Comp.*, pages 265–296, 1994.

[110] A. Sorgatz. *Dynamische Module*. MuPAD Reports. Teubner, Stuttgart, 1996.

[111] H. Stephani. *Differential Equations: Their Solution Using Symmetries*. Cambridge University Press, Cambridge, 1989.

[112] E. Tournier, editor. *Computer Algebra and Differential Equations (CADE-88)*, Computational Mathematics and Applications, London, 1988. Academic Press.

[113] E. Tournier, editor. *Computer Algebra and Differential Equations (CADE-92)*, London Mathematical Society Lecture Note Series 193. Cambridge University Press, 1992.

[114] F. Ulmer. On Liouvillian solutions of linear differential equations. *Appl. Alg. Eng. Commun. Comput.*, 2:171–194, 1992.

[115] M. van Hoeij. Formal solutions and factorization of differential operators with power series coefficients. *J. Symb. Comp.*, 24:1–30, 1997.

[116] M. van Hoeij and J.A. Weil. An algorithm for computing invariants of differential Galois groups. *J. Pure Appl. Alg.*, 117&118:353–379, 1997.

[117] J.A. van Hulzen, B.J.A. Hulshof, B.L. Gates, and M.C. van Heerwaarden. A code optimization package for REDUCE. In G.H. Gonnet, editor, *Proc. ISSAC '89*, pages 163–170. ACM Press, New York, 1989.

[118] E. von Weber. Partielle Differentialgleichungen. In *Enzyklopädie der mathematischen Wissenschaften, Vol. II, Part 1.1*, chapter A5, pages 294–399. 1900.

[119] S. Walcher. On transformation into normal form. *J. Math. Anal. Appl.*, 180:617–632, 1993.

[120] J.A. Weil. First integrals and Darboux polynomials of homogeneous linear differential systems. In G. Cohen, M. Giusti, and T. Mora, editors, *Proc. AAECC-11*, Lecture Notes in Computer Science 948, pages 469–484. Springer-Verlag, Heidelberg, 1995.

[121] J. Weiss. The Painlevé property for partial differential equations II: Bäcklund transformations, Lax pairs, and the Schwarzian derivative. *J. Math. Phys.*, 24:1405–1413, 1983.

[122] J. Weiss, M. Tabor, and G. Carnevale. The Painlevé property for partial differential equations. *J. Math. Phys.*, 24:522–526, 1983.

[123] W. Wiwianka and B. Fuchssteiner. Algorithms to detect complete integrability in 1+1 dimensions. In S. Carillo and O. Ragnisco, editors, *Research Reports in Physics – Nonlinear Dynamics*, pages 131–135. Springer-Verlag, Berlin, 1990.

[124] P.A. Worfolk. Zeros of equivariant vector fields: Algorithms for an invariant approach. *J. Symb. Comp.*, 17:487–511, 1994.

[125] Wu W.-T. Automatic derivation of Newton's gravitational law from Kepler's laws. *Academica Sinica Mathematics-Mechanization Research Preprints*, 1:53, 1987.

[126] Wu W.-T. A constructive theory of differential algebraic geometry based on works of J.F. Ritt with particular applications to mechanical theorem proving of differential geometries. In G. Chaohao, M. Berger, and R.L. Bryant, editors, *Differential Geometry and Differential Equations*, Lecture Notes in Mathematics 1255, pages 173–189, Shanghai, 1987. Springer-Verlag, Berlin 1987.

[127] D. Zwillinger. *Handbook of Differential Equations*. Academic Press, San Diego, 1992.

Algorithms for Solving Linear Ordinary Differential Equations

Winfried Fakler
IAKS, Universität Karlsruhe, fakler@ira.uka.de
http://iaks-www.ira.uka.de/home/fakler/

This article presents the new domain of linear ordinary differential operators and shows how it works in a few examples. Furthermore, in a very informal way the algebraic point of view dealing with ordinary differential equations will be introduced. Using such tools allows to develop general algorithms for solving linear equations.

Introduction

In designing algorithms for solving differential equations one necessarily has to work at the (algebraic) structure of these equations. On one hand this leads to a classification of special types of differential equations. But it never can be complete. On the other hand this will give a classification like linear and nonlinear, ordinary and partial differential equations.
A further decision is, what kind of solutions one would like to determine. For example, one can search for power series solutions, formal or closed-form (symbolic) solutions. This article treats symbolically constructable solutions of linear ordinary differential equations.
Let

$$L(y) = y^{(n)} + a_{n-1}y^{(n-1)} + \ \ldots\ + a_1 y' + a_0 y = 0 \qquad (a_i \in k)$$

be a nth order homogeneous linear differential equation over the coefficient field k, where k is e.g. the field of rational functions. It is well-known, that a nth order linear differential equation has exactly n linearly independent solutions which form a vector space.
Constructable solutions can be grouped together to classes of solutions.

Rational solutions.

Rational functions are solutions lying in k. For that class J. Liouville already gave an algorithm in 1833, but only when k is the rational function field. More general versions are presented by Singer (1991) and Bronstein (1992).

Algebraic solutions.

Algebraic solutions are functions, which lie in an algebraic extension of k, i.e. they satisfy an irreducible polynomial over k. An example for this class is

$$\sqrt[3]{1-\sqrt{x}}.$$

Many renowned mathematicians like Pépin, Fuchs, Klein, Jordan were searching for an algorithm for algebraic solutions. Today, there exists an algorithm from Singer, but it is far from being satisfactory.

Liouvillian solutions.

Liouvillian solutions are functions, which can be constructed from the rational functions by successively substituting nested algebraic functions, integrals and exponential of integrals. An example for such a construction is

$$x \xrightarrow{\sqrt{}} \sqrt{x} \xrightarrow{e^{\int}} \exp\left[\int \sqrt{x}\right].$$

This class of functions includes also functions like the trigonometric functions and logarithms. In principle it consists of nearly all closed-form functions. An important exception are the Bessel functions (and other special functions).

Exponential solutions.

Exponential solutions are functions, whose logarithmic derivatives lie in k. If y is a solution, then y'/y is its logarithmic derivative. An example is

$$y = \exp(x^3),$$

since $y'/y = 3x^2$ lies in k. For this article, exponential functions forms the most important subclass of liouvillian functions. Without an algorithm for them, it is not possible to give an algorithm for liouvillian solutions. Fortunately, there are algorithms for exponential solutions. The very first one stems from Beke in 1894.

Dividing functions into one of these classes is not always unique, e.g. for the function $y = \sqrt{x}$, it is possible to attach it either to the algebraic functions or to the exponential, since $y'/y = 1/2x \in k$. Furthermore, it can be hard to decide for a given function whether it is liouvillian and how one could find the simplest construction.

To every linear ordinary differential equation

$$L(y) = y^{(n)} + a_{n-1}y^{(n-1)} + \ \ldots\ + a_1 y' + a_0 y = 0$$

one can associate a linear operator

$$L(D)[y] = (D^n + a_{n-1}D^{n-1} + \ \ldots\ + a_1 D + a_0)[y] = 0$$

in a unique way. Here, D^i is only another notation for the i-th derivation of y. One calls such an operator a linear ordinary differential operator. The mathematical structure of linear ordinary differential operators is a ring. This structure and its conversion in *MuPAD* is described in the next section.

The LODO Domain Constructor

The ring of linear ordinary differential operators $k[D]$ is presented in *MuPAD* as the domain constructor `Linear-OrdinaryDifferentialOperator`. From the mathematical point of view linear differential operators generate a left skew polynomial ring of derivation type. The elements of such a ring are called skew polynomials or Ore polynomials. For Ore polynomials the usual polynomial addition holds. Only the multiplication is different. It is declared as an extension of the rule for $a \in k$

$$Da = aD + a'$$

to arbitrary Ore polynomials. Multiplication of Ore polynomials is in fact operator composition. Therefore, $k[D]$ is not a commutative ring. This means, there is a left and right division. Indeed, there exists an extended Euclidean algorithm and it is possible to determine for any two nontrivial elements a smallest nontrivial common left multiple. This is the so-called *Ore condition*, i.e. skew polynomials are left Ore rings. Linear ordinary differential operators are even left and right Ore rings.

Starting from this mathematical background the category constructor `UnivariateSkewPolynomialCat` was generated, in which already all operations for univariate skew polynomials independent of the representation are implemented. For a true representation the following domains hierachy was created:

`BaseDomain`
↓
`PolynomialExplicit`
↓
`UnivariatePolynomial`
↓
`UnivariateSkewPolynomial`
↓
`LinearOrdinaryDifferentialOperator`

In this way, based on polynomials it was possible to implement the new domains in a reasonable time. Simply the polynomial multiplication needs to be overloaded by the new noncommutative multiplication and all the resulting left and right operations had to be implemented. Altogether it is an example for the advantages of the domains constructor concept. Based on the domain `UnivariateSkewPolynomial`, it would be possible to generate beside the domain `LinearOrdinaryDifferentialOperator` e.g. a domain for linear ordinary difference operators without need for implementing all operations once more.
To create a LODO domain one has to choose a variable for the operator, e.g. 'Df`, a variable with respect to which one wants to differentiate, and optionally a coefficient field or ring of characteristic zero from the domains package. Note, in *MuPAD* 'D` is not a variable, since it is predefined as an operator. A LODO domain can be created for example with the call

```
>> EF := Dom::ExpressionField(normal):
>> lodo := Dom::LinearOrdinaryDifferentialOperator(Df,x,EF);
```

There are more possiblities for generating differential operators, e.g. one can generate the same operator by a vector

```
>> lodo([x+1, sin(x), 1]);
```

or by a differential equation

```
>> lodo(diff(y(x),x,x)+sin(x)*diff(y(x),x)+(x + 1)*y(x),y(x));
```

and of course by an operator polynomial

```
>> A:=lodo(Df^2+sin(x)*Df+x+1);
```

$$Df^2 + \sin(x)\ Df + (x + 1)$$

```
>> B:=lodo(Df+x);
```

$$Df + x$$

The product of the two operators one gets with

```
>> P:=A*B;
```

$$Df^3 + (x + \sin(x))\ Df^2 + (x + x\ \sin(x) + 3)\ Df + (x + \sin(x) + x^2)$$

That this product is really an operator composition one can test with

```
>> expand((A*B)(y(x),Unsimplified) - A(B(y(x),Unsimplified),Unsimplified));
```

$$0$$

One can also see from

```
>> lodo(Df)*lodo(x), lodo(x)*lodo(Df);
```

$$x\ Df + 1,\ x\ Df$$

that the defined multiplication satisfies the rule above and that it is noncommutative. The option 'Unsimplified` is necessary, since the function `func_call` of the LODO domain is primarily intended for the zero test of solutions. Naturally it is possible to evaluate operators, but there are internal manipulations which leave the solution space unchanged, however it will change the resulting expression. By the given option this will be suppressed. It is also possible for example to compute a right division.

```
>> t:=P::rightDivide(P,A);

              table(
                remainder = (- cos(x) + 2 ) Df + (sin(x) - 1),
                quotient = Df + x
              )


>> t[quotient]*A+t[remainder];

      3                   2                                              2
    Df  + (x + sin(x)) Df  + (x + x sin(x) + 3) Df + (x + sin(x) + x )
```

In short, a LODO domain contains among other things the operations left/right{Divide, Quotient, Remainder, Gcd, Lcm, ExtendedEuclid} and allows to determine the adjoint operator,

```
>> P::adjoint(P);

    3                   2
- Df  + (x + sin(x)) Df  + (- x + 2 cos(x) - x sin(x) - 1 ) Df +

                            2
(x - sin(x) - x cos(x) + x  - 1)
```

symmetric powers (`symmetricPower`) of an operator and currently limited factoring and computing zeros or solutions of operators with rational function coefficients. For demonstrating that, here an example:

```
>> L:=lodo(Df^4+(2*x-1)/(2*x*(x-1))*Df^3+(143*x-147)/(784*x^2*(x-1))*Df^2\
&>   +(-18*x+21)/(32*x^3*(x-1))*Df+(2349*x-2940)/(3136*x^4*(x-1)));


  4   /    2 x - 1    \   3   /    143 x - 147     \   2
Df  + | ------------- | Df  + | ------------------ | Df  +
      |             2 |       |         2        3 |
      \ - 2 x + 2 x   /       \ - 784 x  + 784 x   /

/   - 18 x + 21    \          2349 x - 2940
| ---------------- | Df + --------------------
|       3       4 |                4         5
\ - 32 x  + 32 x   /       - 3136 x  + 3136 x

>> Factor(L);

/   2   /    2 x - 1    \               1          \ /       1  \
| Df  + | ------------- | Df - ------------------ | | Df + --- |
|       |             2 |                      2  | \       4 x /
\       \ - 2 x + 2 x   /       - 196 x + 196 x    /

/       1  \
| Df - --- |
\      4 x /
```

An operator which decomposes into factors is called *reducible*, if it is not reducible, it is called *irreducible*. Currently, the function `Factor` can only find left and right factors of degree 1, which means that a decomposition

in irreducible factors is guaranteed only for operators up to third degree. Nevertheless it is possible to find decompositions of higer degree operators. The situation for computing liouvillian zeros is quite similar. Finding *all* liouvillian zeros can currently only be guaranteed for second degree operators and reducible operators of third degree. But one can also find liouvillian zeros of higher degree operators.

```
>> sols:=L::liouvillianZeros(L):
>> map(sols,combine@simplify@expand@eval);

{  1/4     3/4     3/4    /  1/4    / acosh(2 x - 1) \     \
{ x   , 2 x   , 2 x    int| x    exp| -------------- |, x | -
{                         \         \       14       /    /

      1/4    /  3/4    / acosh(2 x - 1) \     \
   2 x    int| x    exp| -------------- |, x |,
             \         \       14       /    /

      3/4    /  1/4    /   acosh(2 x - 1) \     \
   2 x    int| x    exp| - -------------- |, x | -
             \         \         14       /    /

      1/4    /  3/4    /   acosh(2 x - 1) \     \ }
   2 x    int| x    exp| - -------------- |, x | }
             \         \         14       /    / }

>> map(%,L);

                                    {0}
```

The last call shows that the determined functions are in fact zeros of the operator.
For readers who want to know more about Ore polynomials and linear ordinary differential operators we refer to Bronstein and Petkovsek [2] and Ore [6]. The currently most efficient method for factoring differential operators is described in van Hoeij [11]. Information about the domains package one can find in the online help system of *MuPAD*.

An Algebraic Algorithm

Algorithms computing liouvillian solutions of second order linear differential equations have been developed at the end of the last century. The first complete and implemented algorithm for second order equations stems from Kovacic in 1977, see [5]. A considerably simplification and much more efficient variant of this algorithm was recently presented from Ulmer and Weil [10] and already in 1981 Singer gave an algorithm computing liouvillian solutions for equations of arbitrary order. Unfortunately this algorithm has such an enormous complexity, that it is even not implemented for second order equations. A new development by Singer and Ulmer [8] could close this gap.
All modern solution procedures are in principle based on differential Galois theory. Because of the vector space structure of a solution set in this theory one associate a linear group to every linear differential equation. From this group, it is now possible to draw conclusions about the solutions. For example it is known, that if a linear differential equation has a liouvillian solution y, then it has also a liouvillian solution of which the logarithmic derivative y'/y is algebraic of bounded degree. On the basis of this Singer theorem, one can reduce the problem of finding liouvillian solutions to the problem of finding a minimal polynomial, thus an algebraic equation. Hence, it is an algebraic problem. The considerably higher efficiency of the Ulmer-Weil algorithm compared with Kovacic's algorithm comes from the different calculation of the coefficients of the minimal polynomial. In the Ulmer-Weil algorithm they are simply determined via recursion.

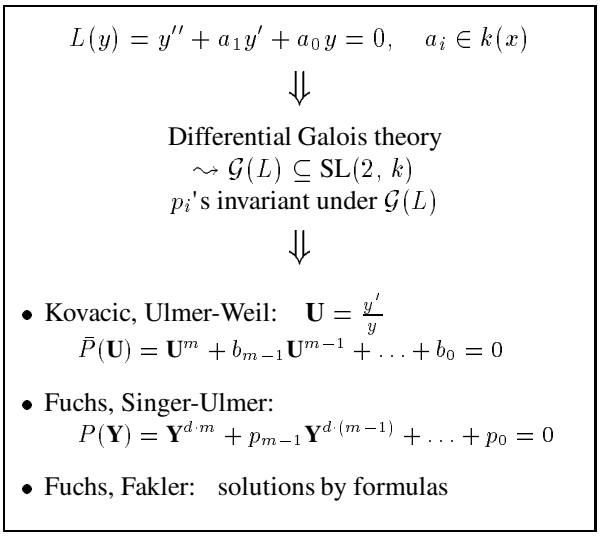


Figure 23: Principle of algebraic algorithms.

Even, when the Galois group of an irreducible differential equation is finite, one can determine a minimal polynomial of a solution. But for this the Galois group must be explicitely known. For finite primitive groups, L. Fuchs determined in the years from 1875 to 1878 such a method for second order equations. In Singer und Ulmer [7] this method was extended to higher order equations. Minimal polynomials for all finite groups are given in Fakler [3]. For the reader who wants to know more about differential Galois theory we refer to Kaplansky [4] and to the references in the given articles. In figure 23 one can find an overview about the methods for second order differential equations.

In the following we outline the method given in Fakler [3] for second order differential equations and we show how one can use it together with a factorization to compute liouvillian solutions for many higher order equations. Let

$$L(y) = y'' + a_1 y' + a_0 y = 0, \quad a_i \in k(x)$$

be a differential equation with rational function coefficients over a field k of charakteristic 0. This will be satisfied e.g. for $k = \mathbb{Q}$. Further, let $\{y_1, y_2\}$ be a fundamental set of solutions for $L(y) = 0$ and $\mathcal{G}(L)$ its Galois group. One can imagine it as a (2×2) matrix group, i.e. $\mathcal{G}(L)$ is a subgroup of the general linear group $\mathrm{GL}(2, k)$.

For using the powerful tools from Galois theory one has to take care that the Galois group of the equation is unimodular, which means that the determinants of all matrices are 1. Then the Galois group is a subgroup of the special linear group $\mathrm{SL}(2, k)$. This is necessary, since only these groups are all known. However, it is not a problem because e.g. with the transformation $y = z \cdot \exp(-\frac{\int a_1}{2})$ we can guarantee it. For that reason, we assume from now on that the Galois group is unimodular.

One distinguishes, if the Galois group is reducible or imprimitive (which means here, that non-zero elements are only in the main diagonal or only outside the main diagonal) or if it is isomorph to the tetrahedral or octahedral or icosahedral group.

That the Galois group of a second order equation is reducible is equivalent to the existence of an exponential solution.

For distinguishing the remaining cases the so-called symmetric powers of $L(y) = 0$ are used. Symmetric powers $L^{\circledS m}(y) = 0$ are from $L(y) = 0$ constructable differential equations with the property that all mth power products of solutions of $L(y) = 0$ are solutions of it. The second symmetric power $L^{\circledS 2}(y) = 0$ e.g. has the solution space $\{y_1^2, y_1 y_2, y_2^2\}$. Rational solutions of symmetric powers $L^{\circledS m}(y) = 0$ correspond to homogeneous (polynomial) invariants of degree m of $\mathcal{G}(L)$. Since the invariants of all the Galois groups are known, it is possible to distinguish

the following cases for irreducible second order equations: If $L^{\circledS m}(y) = 0$ has a nontrivial rational solution for

- $m = 4$: $\mathcal{G}(L)$ is imprimitive
- $m = 6$, $(m \neq 4)$: $\mathcal{G}(L) \cong$ tetrahedral group
- $m = 8$, $(m \neq 4, 6)$: $\mathcal{G}(L) \cong$ octahedral group
- $m = 12$, $(m \neq 4, 6, 8)$: $\mathcal{G}(L) \cong$ icosahedral group
- $(m \neq 4, 6, 8, 12)$: $\mathcal{G}(L) \cong \mathrm{SL}(2, k)$.

More, one gets from that a beautiful criterion for when an irreducible second order differential equation with unimodular Galois group has liouvillian solutions namely if and only if its twelveth symmetric power has a nontrivial rational solution.
In the imprimitive case the fourth symmetric power $L^{\circledS 4}(y) = 0$ has a rational solution r (in one case, there are two rational solutions, see figure 24). If W denotes the Wronskian of $L(y) = 0$, which is, by the way, computable from $W = \exp(\int a_1)$, then

$$y_1 = \sqrt[4]{r}\, e^{-\frac{C}{2}\int \frac{W}{\sqrt{r}}} \quad \text{and} \quad y_2 = \sqrt[4]{r}\, e^{\frac{C}{2}\int \frac{W}{\sqrt{r}}}$$

are the desired solutions. The remaining constant C can be computed with the determining equation

$$\frac{4r''r - 3(r')^2}{16r^2} + \frac{W^2}{4r}C^2 - \frac{r'}{4r}a_1 + a_0 = 0.$$

Unfortunately, for the rest of the cases there are no direct solution formulae. However, it remains possible to compute the minimal polynomial of a solution via solution formulae. One recalls, in the Ulmer-Weil algorithm the coefficients of the minimal polynomial of the logarithmic derivative of a solution are determined by recursion. Here the coefficients are computed by determining a constant. With the tetrahedral group we demonstrate how this procedure will work. For the octahedral and icosahedral group the process is similar.
In a precomputation one determines a minimal polynomial decomposed into invariants for the tetrahedral group:

$$\mathbf{Y}^{24} + 10I_2\mathbf{Y}^{16} + 5I_3\mathbf{Y}^{12} - 15I_2^2\mathbf{Y}^8 - I_2I_3\mathbf{Y}^4 + I_1^4.$$

To the invariant $\bar{I}_1 = 4I_1$ corresponds a rational solution r of the sixth symmetric power $L^{\circledS 6}(y) = 0$. Since solutions of differential equations can only be unique up to a constant, one has to determine a constant c such that $\bar{I}_1 = c \cdot r$. To do this, one can use the syzygy between the invariants from which one gets the determining equation

$$\left(25J(r, H(r))^2 + 64H(r)^3\right)c^2 + 10^6 \cdot 108r^4 = 0,$$

where

$$H(f) = \frac{m-1}{W^2}\left[\left(\frac{f'}{f}\right)^2 + m\left(\frac{f'}{f}\right)' + ma_1\left(\frac{f'}{f}\right) + m^2a_0\right]f^2$$

is the Hessian – here with $m = 6$ – and

$$J(f, g) = \frac{mfg' - nf'g}{W}$$

the Jacobian with $m = 6$ and $n = 8$. The invariant I_2 one then gets from the Hessian ($m = 6$) by

$$I_2 = \frac{1}{400} \cdot c^2 \cdot H(r),$$

while invariant I_3 can be computed with the Jacobian by ($m = 6$, $n = 8$)

$$I_3 = \frac{1}{3200} \cdot c^3 \cdot J(r, H(r)).$$

In figure 24 is a summary of the just represented method. As an application, we now solve the famous example of

Input: $L(y) = 0$ with $\mathcal{G}(L) \subseteq \mathrm{SL}(2, \bar{\mathbb{Q}})$
Output: set of liouvillian solutions or minimal polynomial of a solution

1. $L(y) = 0$ is reducible?
   compute an exponential and a liouvillian solution

2. $L^{\circledS 4}(y) = 0$ has a nontrivial rational solution r?

   (a) only one? $y_{1,2} = \sqrt[4]{r}\,\exp\left[\pm\frac{C}{2}\int\frac{W}{\sqrt{r}}\right]$

   (b) two? $r := c_1 r_1 + c_2 r_2$, then (a)

3. $L^{\circledS m}(y) = 0$, $m \in \{6, 8, 12\}$ has a nontrivial rational solution r?

   - $m = 6$
     $\left(25J(r,H(r))^2 + 64H(r)^3\right)c^2 + 10^6 \cdot 108r^4 = 0 \quad (n = 8)$
     $I_1 = \frac{1}{4}\cdot c\cdot r, \quad I_2 = \frac{1}{400}\cdot c^2\cdot H(r), \quad I_3 = \frac{1}{3200}\cdot c^3\cdot J(r,H(r))$
     $P(\mathbf{Y}) = \mathbf{Y}^{24} + 10I_2\mathbf{Y}^{16} + 5I_3\mathbf{Y}^{12} - 15I_2^2\mathbf{Y}^8 - I_2I_3\mathbf{Y}^4 + I_1^4$
   - $m = 8$
     $\left(49J(r,H(r))^2 + 144H(r)^3\right)c - 118013952r^3H(r) = 0 \quad (n = 12)$
     $I_1 = -\frac{1}{16}\cdot c\cdot r, \quad I_2 = \frac{1}{150528}\cdot c^2\cdot H(r)$
     $P(\mathbf{Y}) = \mathbf{Y}^{48} + 20I_1\mathbf{Y}^{40} + 70{I_1}^2\mathbf{Y}^{32} + \left(2702{I_2}^2 + 100{I_1}^3\right)\mathbf{Y}^{24} +$
     $\left(-1060I_1{I_2}^2 + 65{I_1}^4\right)\mathbf{Y}^{16} + \left(78{I_1}^2{I_2}^2 + 16{I_1}^5\right)\mathbf{Y}^8 + {I_2}^4$
   - $m = 12$
     $\left(121J(r,H(r))^2 + 400H(r)^3\right)c + 708624400\cdot 1728r^5 = 0 \quad (n = 20)$
     $I_1 = \frac{1}{125}\cdot c\cdot r, \quad I_2 = \frac{1}{121\cdot 34375}\cdot c^2\cdot H(r), \quad I_3 = \frac{11}{2420\cdot 3125}\cdot c^3\cdot J(r,H(r))$
     $P(\mathbf{Y}) = \mathbf{Y}^{120} + 20570I_2\mathbf{Y}^{100} + 91I_3\mathbf{Y}^{90} - 86135665{I_2}^2\mathbf{Y}^{80} - 78254I_2I_3\mathbf{Y}^{70} +$
     $\left(14993701690{I_2}^3 + 11137761250{I_1}^5\right)\mathbf{Y}^{60} + 897941{I_2}^2I_3\mathbf{Y}^{50} +$
     $\left(-11602919295{I_2}^4 + 273542733750{I_1}^5I_2\right)\mathbf{Y}^{40} + \left(-151734{I_2}^3 - 6953000{I_1}^5\right)I_3\mathbf{Y}^{30} +$
     $\left(503123324{I_2}^5 - 7854563750{I_1}^5{I_2}^2\right)\mathbf{Y}^{20} + \left(1331{I_2}^4 + 500{I_1}^5I_2\right)I_3\mathbf{Y}^{10} + 3125{I_1}^{10}$

4. $L(y) = 0$ has no liouvillian solution.

Figure 24: Sketch of the second order algorithm.

Kovacic.

Example.

The second order differential equation

$$L(y) = y'' + \left(\frac{3}{16x^2} + \frac{2}{2(x-1)^2} - \frac{3}{16x(x-1)}\right) y = 0$$

possess no exponential solution and its fourth symmetric power $L^{\circledS 4}(y) = 0$ has no nontrivial rational solution. But then $L^{\circledS 6}(y) = 0$ possesses the rational solution $r = x^2(x-1)^2$. This means that the Galois group $\mathcal{G}(L)$ is isomorphic to the tetrahedral group.

For $W = 1$, the Hessian and Jacobian are computed with

$$H(r) = \frac{25}{4}x^2(x-1)^3 \qquad \text{and} \qquad J(r, H(r)) = -\frac{25}{2}x^3(x-1)^4(x-2).$$

From that, one gets the determining equation

$$\begin{aligned}\left(c^2+27648\right)x^{16} + \left(-8c^2-221184\right)x^{15} + \left(28c^2+774144\right)x^{14} + \\ \left(-56c^2-1548288\right)x^{13} + \left(70c^2+1935360\right)x^{12} + \left(-56c^2-1548288\right)x^{11} + \\ \left(28c^2+774144\right)x^{10} + \left(-8c^2-221184\right)x^9 + \left(c^2+27648\right)x^8 &= 0,\end{aligned}$$

respectively e.g. for the regular point $x_0 = 2$ the equation

$$c^2 + 27648 = 0.$$

Hence, $c = \pm 96\sqrt{-3}$. Substituting

$$\begin{aligned} I_1 &= \frac{1}{4} \cdot c \cdot r = 24\sqrt{-3}\, x^2(x-1)^2, \\ I_2 &= \frac{1}{400} \cdot c^2 H(r) = 432x^2(x-1)^3, \\ I_3 &= \frac{1}{3200} \cdot c^3 J(r, H(r)) = 10368\sqrt{-3}\, x^3(x-1)^4(x-2) \end{aligned}$$

in the minimal polynomial decomposed into invariants, we obtain the desired minimal polynomial of a solution:

$$\begin{aligned} P(\mathbf{Y}) = \mathbf{Y}^{24} &- 4320x^2(x-1)^3\mathbf{Y}^{16} + 51840\sqrt{-3}\, x^3(x-1)^4(x-2)\mathbf{Y}^{12} - 2799360x^4(x-1)^6\mathbf{Y}^8 \\ &+ 4478976\sqrt{-3}\, x^5(x-1)^7(x-2)\mathbf{Y}^4 + 2985984x^8(x-1)^8. \end{aligned}$$ ◇

Solutions of the minimal polynomial $P(\mathbf{Y}) = 0$ are also solutions of the differential equation $L(y) = 0$. From the Galois theory one knows, that for solvable groups the solutions can be represented in radicals (nested root expressions). Therefore, in the cases of the tetrahedral or octahedral group it would be possible to compute radical expressions of the solutions. But such a calculation has an enormous complexity and there is no implementation in any computer algebra system for performing this operation known to the author. However, in the icosahedral case it is only possible to represent a solution implicit as solution of a minimal polynomial.

With this algorithm one can even find solutions of higher order reducible equations. For this, one has to factorize the associated operator and beginning from the right to solve the factors. With the method of variation of parameters which was introduced by Lagrange one can construct solutions of the equation from solutions of the factors. An example for that is already given in the last section. Unfortunately, decomposing into factors is not unique. To determine in this way as many as possible (all) liouvillian solutions one can compute from a given factorization with the algorithm from Tsarev [9] all the other possible factorizations. By a complete implementation of factoring and e.g. by an implementation of the algorithm from Singer and Ulmer [8], one then could find all liouvillian solutions.

I thank Frank Postel for his invitation to present my implementations and results from [3] here, Eckhard Pflügel for implementing the algorithm for exponential solutions, Paul Zimmermann for including my algorithms into the ODE solver and for many helpful discussions, Ralf “tonner” Hillebrand for including some of my special wishes into the *MuPAD* system and for his help in solving problems and last but not least Werner Seiler for proofreading this article.

References

[1] Barkatou, M. (1997). *An Efficient Algorithm for Computing Rational Solutions of Systems of Linear Differential Equations*. Preprint.

[2] Bronstein, M., Petkovsek, M. (1996). *An introduction to pseudo-linear algebra*. Theor. Comp. Science **157**, No. 1.

[3] Fakler, W. (1997). *On second order homogeneous linear differential equations with Liouvillian solutions.* To appear in Theor. Comp. Science **187**.

[4] Kaplansky, I. (1957). *Introduction to differential algebra.* Paris: Hermann.

[5] Kovacic, J. (1986). *An algorithm for solving second order linear homogeneous differential equations.* J. Symb. Comp. **2**, 3-43.

[6] Ore, O. (1933). *Theory of non-commutative polynomials.* Ann. of Math. **34**, 480-508.

[7] Singer, M.F., Ulmer, F. (1993). *Liouvillian and Algebraic Solutions of Second and Third Order Linear Differential Equations.* J. Symb. Comp. **16**, 37-73.

[8] Singer, M.F., Ulmer, F. (1996). *Linear Differential Equations and Products of Linear Forms.*. Preprint. To appear in J. Pure and Applied Algebra.

[9] Tsarev, S.P. (1996). *An Algorithm for Complete Enumeration of All Factorizations of a Linear Ordinary Differential Operator*. In: Proceedings of ISSAC' 96, 226-231.

[10] Ulmer, F., Weil, J.A. (1996). *Note on Kovacic's Algorithm.* J. Symb. Comp. **22**, 179-200.

[11] van Hoeij, M. (1996). *Factorization of Linear Differential Operators*. PhD thesis, University of Nijemegen.

[12] Zwillinger, D. (1992). *Handbook of differential equations*. 2nd ed. San Diego: Academic Press.

Polynomfaktorisierung in *MuPAD*

Stefan Wehmeier
Universität–Gesamthochschule Paderborn, stefanw@math.uni-paderborn.de

Schon seit Version 1.2 sind Methoden zur Faktorisierung von Polynomen ein wichtiger Bestandteil der mathematischen Bibliothek von MuPAD . Ähnlich wie bei `gcd` *und* `int`*, verbirgt sich hinter der Funktion* `factor` *ein umfangreiches Library–Paket mit Namen* `faclib`*. Seit der ersten Version dieses Pakets (siehe [1]) ist der Funktionsumfang weiter gestiegen. Der vorliegende Artikel gibt einen kurzen Überblick über den erreichten Stand.*

Einführung

Sei R ein faktorieller Ring. Eine *Faktorisierung* eines Polynoms $f \in R[x_1, \ldots x_n]$ ist definiert als eine Liste

$$[r, f_1, \alpha_1, \ldots, f_k, \alpha_k],$$

wo $r \in R$ ein Inhalt von f, $f_1, \ldots f_k \in R[x_1, \ldots, x_n]$ irreduzible nichtkonstante Polynome und $\alpha_1, \ldots \alpha_k$ positive ganze Zahlen sind, so daß die f_i paarweise teilerfremd sind und

$$f = r \prod_{i=1}^{k} f_i^{\alpha_i}$$

gilt. Die Faktorisierung eines Polynoms f wird durch den Funktionsaufruf `factor(f)` berechnet.
Die *quadratfreie Faktorisierung* ist ebenso definiert, nur daß die f_i nicht irreduzibel, sondern nur quadratfrei sein müssen und zusätzlich gefordert wird, daß die α_i paarweise verschieden sind. Man erhält sie mittels `sqrfree(f)`.
In der ursprünglichen, von Jian Yuan erstellten Fassung der Library `faclib` wurde hauptsächlich der Fall $R = \mathbb{Z}$ betrachtet und mit den in [2] angegebenen Methoden behandelt. Die jetzige Fassung enthält Ergänzungen von Klaus Drescher, Paul Zimmermann und dem Autor.
Polynomfaktorisierung ist sicherlich — verglichen etwa mit manchen Aufgabenstellungen in der Numerik — ein weniger praxisrelevantes Gebiet. Dementsprechend wird es viele Anwender mit Interesse für algorithmische Mathematik geben, die mehr der Weg als das Ziel interessiert. `faclib` bietet daher auch die Möglichkeit, den Rechenweg mittels `setuserinfo(faclib,..)` genau zu verfolgen; außerdem gibt sind für manche Probleme mehrere, alternative Verfahren implementiert. Der interessierte Benutzer kann auf diese Weise selbst eine Wahl treffen, indem er die Funktionen von `faclib` direkt und nicht mittels `factor` aufruft.

Polynome in einer Variablen

Ganzzahlige und rationale Polynome

Für univariate Polynome über $\mathbb{Z}$ bzw. $\mathbb{Q}$ bietet *MuPAD* neben dem Verfahren aus [2] mit exponentieller worst–case Laufzeit auch eine Implementation des Algorithmus von Lenstra, Lenstra und Lovasz (`faclib:lll`), das trotz polynomiell beschränkter Laufzeit in der Praxis allerdings gegenüber dem erstgenannten wesentlich langsamer ist.

Polynome über algebraischen Körpererweiterungen

Univariate Polynome über (separablen) algebraischen Körpererweiterungen `Dom::AlgebraicExtension(F,...)` werden mittels Faktorisierung der Norm faktorisiert. Voraussetzung ist natürlich, daß die Faktorisierung von Polynomen über F möglich ist.

Polynome über endlichen Körpern

Während Faktorisierung modulo einer Primzahl ein wichtiges Hilfsmittel für die Faktorisierung über $\mathbb{Z}$ ist, wurden andere endliche Körper als $\mathbb{Z}_p$ erst später implementiert (als `Dom::GaloisField`). In beiden Fällen wird für Polynome in einer Variable der Algorithmus von Cantor–Zassenhaus verwendet, siehe [3].

Polynome über Z_n

MuPAD findet in manchen Spezialfällen eine Faktorisierung eines Polynoms über $\mathbb{Z}_n$, selbst wenn n nicht prim ist.

Multivariate Polynome

Gerade die Verbesserung der Funktionalität in diesem Bereich war ein Schwerpunkt der Arbeit der Nachfolger von Jian Yuan. Hierbei wurde der Algorithmus aus [2] für multivariate Polynome über $\mathbb{Z}$ beibehalten. Hinzu kamen mehrere generische Algorithmen für beliebige Körper, d.h. *MuPAD* ist nun in der Lage, Polynome in beliebig vielen Variablen über all den Körpern zu faktorisieren, über denen dies für univariate Polynome möglich ist.

Vorhandene Algorithmen

Abgesehen von Kronecker's Trick (implementiert als `faclib::kronecker_factor`) beruhen alle Verfahren auf dem Prinzip des ideal–adischen Hensel Lifting, siehe [5], oder dem verwandten Prinzip der Approximation an eine Nullstelle (siehe etwa [6], beruhend auf [7]). Manche der Algorithmen sind nur für bivariate Polynome implementiert. Im einzelnen existieren:

- Lineares (`faclib::multifactor`) bzw. quadratisches (`faclib::quadfactor`) Liften mit Rekombination der Faktoren durch Ausprobieren aller Möglichkeiten
- Lineares Liften, verbunden mit Bestimmung eines „kürzesten Vektors" (Algorithmus aus [5]) (`faclib::bifactor`)
- Finden *eines* Faktors durch lineare bzw. quadratische Nullstellenapproximation (Algorithmus aus [8]), verbunden mit der Lösung eines Gleichungssystems (`faclib::findfactor` ohne bzw. mit Option `Quadratic`).

Laufzeitverhalten

Beschränkt man sich auf bivariate Polynome, die in jeder Variable den Grad n haben, so hat `findfactor` eine asymptotische Laufzeit von $O(n^7)$ und `bifactor` eine von $O(n^{11})$ Körperoperationen, während `multifactor` und `quadfactor` im schlimmsten Fall eine exponentielle Laufzeit haben (die Zeit für die als erster Schritt notwendige Bestimmung der modularen Faktoren ist hier nicht eingerechnet und hängt vom Koeffizientenkörper ab) . In der Praxis kommt dieser schlimmste Fall selten vor, und die Laufzeiten verhalten sich genau umgekehrt. Aber selbst für „ungünstige" Beispiele von dem in [9] beschriebenen Typ wirkt sich das exponentielle Verhalten erst bei sehr großen Eingaben spürbar aus.
Im Vergleich zu anderen Computeralgebrasystemen liegen die Stärken von *MuPAD* im Bereich von dichten Polynomen (d.h. Polynomen mit vielen Termen) und endlichen Körpern mit großer Charakteristik.
Eine detaillierte Analyse der Algorithmen mit Korrektheitsbeweis, Laufzeitabschätzungen und experimentell gemessenen Zeiten (für den Fall von Polynomen in zwei Variablen über endlichen Körpern) findet man in [10].

Einbindung von Modulen

Eine ganz neue Perspektive auch für die Faktorisierung von Polynomen ergibt sich durch die Möglichkeit, dynamische Module in *MuPAD* einzubinden. So bietet das Modul `magnum` eine gegenüber der *MuPAD*–Library um Größenordnungen schnellere Faktorisierung von univariaten Polynomen über $\mathbb{Z}_p$, wo $p < 2^{16}$. Das an der Universität Kaiserslautern entwickelte Computeralgebrasystem *Singular* kann ebenfalls unter *MuPAD* mittels einer als Modul implementierten Schnittstelle angesprochen werden und bietet schnelle Faktorisierungsalgorithmen für $\mathbb{Q}$, algebraische Zahlkörper sowie endliche Körper mit Charakteristik $p \leq 32003$.
Diese Module bedeuten gerade deswegen eine ideale Ergänzung zu den in der Library vorhandenen Funktionen, weil deren Schwachpunkte (algebraische Zahlkörper, endliche Körper mit sehr kleiner Charakteristik) abgedeckt werden, während die in den Modulen fehlenden Fälle (vor allem endliche Körper mit großer Charakteristik) eine Stärke von `faclib` bilden.

Literatur

[1] Jian Yuan, *Factorization in MuPAD*. In: MathPAD, 4(2):46–47, Dezember 1994.

[2] Keith Geddes, Stephen Czapor und George Labahn, *Algorithms for computer algebra*, Kluwer Academic Publishers, 1992.

[3] David Cantor und Hans Zassenhaus, *A new algorithm for factoring polynomials over finite fields*. In: Mathematics of computation, 36(154):587–592, April 1981.

[4] A.K. Lenstra, H.W. Lenstra und L.Lovasz, *Factoring polynomials with rational coefficients*. In: Math. Ann. (261):515–534, 1982.

[5] Joachim von zur Gathen, *Hensel and Newton methods in valuation rings*. In: Mathematics of Computation, 42(166):637–661, 1984.

[6] Erich Kaltofen, *A Polynomial-Time Reduction from Bivariate to Univariate Integral Polynomial Factorization*. In: Proc. 23rd Sympos. on Foundations of Computer Science: 57–64, 1982.

[7] Arjen K. Lenstra, *Polynomial factorization by root approximation*. In: Proc. of the EUROSAM, 272–276, 1984.

[8] Joachim von zur Gathen und Erich Kaltofen, *Factorization of Multivariate Polynomials over Finite Fields*. In: Mathematics of Computation, 45(171):251–261, 1985.

[9] Erich Kaltofen, David Musser und David Saunders, *A generalized class of polynomials that are hard to factor*. In: SIAM Journal on Computing, 12(3):473–483, 1983.

[10] Stefan Wehmeier, *Bivariate Polynomials Over Finite Fields*. Diplomarbeit, 1997. Für Surfer erreichbar unter `http://www.math.uni-paderborn/user/stefanw/Papers/diplom.dvi`.

`property` – algebraische Eigenschaften in MuPAD

Jürgen Billing
Universität–GH Paderborn, bij@uni-paderborn.de

In diesem Artikel wird eine neue Funktionalität des Computeralgebra-Systems MuPAD *vorgestellt. Es wird die Möglichkeit beschrieben, die Evaluierung mit algebraischen Eigenschaften zu beeinflussen.*

Grenzen der Evaluierung

Ausdrücke können oft nicht vereinfacht werden, wenn den Bezeichnern der Ausdrücke keine Werte zugewiesen werden. Dadurch können verschiedene mathematische Probleme nicht dargestellt werden:

$$\text{sign}(a) = \begin{cases} 1, & \text{falls} \quad a > 0 \\ 0, & \text{falls} \quad a = 0 \\ -1, & \text{falls} \quad a < 0 \end{cases},$$

$$(-1)^n = \begin{cases} 1, & \text{falls} \quad n\ gerade \\ -1, & \text{falls} \quad n\ ungerade \\ (-1)^n & \text{sonst} \end{cases},$$

$$\sqrt{x \cdot y} = \sqrt{|\ x\ |}\,\sqrt{|\ y\ |}, \qquad \text{wenn} \quad (x \cdot y) > 0$$

.
In diesen Fällen wird die Möglichkeit vermißt, einem Bezeichner Eigenschaften zuweisen zu können, die den Wertebereich des Bezeichners einschränken, ohne ihn auf einen konkreten Wert festzulegen.

Evaluierung von Eigenschaften

Ziele

Ein Evaluierungsmechanismus für Eigenschaften von Bezeichnern soll dem Benutzer von MuPAD die Möglichkeit geben, auf die symbolische Evaluierung Einfluß zu nehmen.
Die Struktur der Eigenschaften soll eine eindeutige Spezifizierung für die Erstellung neuer Eigenschaften und deren Einbettung in bestehende Eigenschaftsdomains ermöglichen.
Außerdem sollen vorhandene Regeln erweiterbar sein.

Organisation von Eigenschaften

Algebraische Eigenschaften können nach ihrer Bedeutung und ihrer Darstellung zu Klassen zusammengefaßt werden.

Beispiel:	Zahlenbereiche	Relationen
	Intervalle	Körpererweiterungen
	Funktionstypen	Eigenschaften von Matrizen
	Teilbarkeit	

Für die Implementierung solcher Gruppierungen bietet sich in MuPAD die Definition eines eigenen *Domains* an.

Domain von Eigenschaften

Ein Domain von Eigenschaften hat als Domainelemente algebraische Eigenschaften. Die Methoden des Domains repräsentieren die Verknüpfungen von Eigenschaften dieses Domains und realisieren die Ableitungen von Eigenschaften.

Beispiel:
Das Domain `NRange` repräsentiert die gebräuchlichen Zahlenbereiche. Elemente des Domains sind z.B.
`NRange("PosInt"),NRange("Integer"),NRange("Positive"),`
`NRange("NonNegative"),NRange("real")` etc.
Die Verknüpfung von Eigenschaften wird durch die Methoden `NRange::_plus`, `NRange::_mult` usw. ermöglicht.
`NRange("NonNegative") + NRange("PosInt")` $\Longrightarrow$ `NRange("Positive")`
`NRange("Positive") * NRange("Integer")` $\Longrightarrow$ `NRange("RealNum")`
Ableitungen (Folgerungen) zwischen Zahlenbereichen berechnet die Methode `NRange::is`:
`NRange::is(NRange("Integer"), NRange("RealNum"))` $\Longrightarrow$ `TRUE`
`NRange::is(NRange("RealNum"), NRange("Positive"))` $\Longrightarrow$ `UNKNOWN`

Die Verknüpfungen `_and` und `_or` haben eine andere Bedeutung als die Operationen `_plus`, `_subtract`, `_negate`, `_mult`, `_divide`, `_invert`, `_power`.
`_and` (`_or`) repräsentiert die Schnittmenge (Vereinigung) von Bezeichnern mit diesen Eigenschaften.

`NRange("Positive") and NRange("Integer") = NRange("PosInt")`
$\Longleftrightarrow \quad \{x \mid x \in \mathbb{R}_{\geq 0}\} \cap \{x \mid x \in \mathbb{N}\}$

Durch die Methoden `_plus`, `_subtract` etc. werden folgende Regeln ausgedrückt:

Der Bezeichner `x` hat die Eigenschaft `X(k)` (des Domains `X`), `y` die Eigenschaft `X(l)`. Der Ausdruck `x + y` habe dann die Eigenschaft $\texttt{X}(\widehat{\texttt{kl}})$. Die Methode `X::_plus` muß also bei Aufruf `X::_plus(X(k), X(l))` das Ergebnis $\texttt{X}(\widehat{\texttt{kl}})$ zurückgeben.

Vorteile der Klassifizierung

- *Übersichtlichkeit*
  Durch die Zugehörigkeit zweier Eigenschaften zu einem Domain besteht sofort ein inhaltlicher Zusammenhang zwischen diesen Eigenschaften (gleiche Darstellung, existierende Verknüpfungen durch die Methoden des Domains, existierende Ableitung durch die Methode `"is"` des Domains etc.)

  Die gesamten Regeln zur Verknüpfung aller Eigenschaften eines Domains (mit einem bestimmten Operator) können durch eine Methode dieses Domains dargestellt werden. Dadurch können sehr viele Regeln eingespart werden. (Das Domain `NRange` mit `18` Zahlenbereichen müßte nur für die Verknüpfung `_plus` aller Bereiche `324` Regeln beinhalten.)

- *beliebige Schachtelung von Eigenschaften*
  Eigenschaften können Teil einer neuen Eigenschaft sein (z.B. Intervalle von Zahlenbereichen). Damit können bestehende Strukturen und Regeln genutzt werden, deren Korrektheit und Abgeschlossenheit bekannt ist.

- *einfache Einbettung neuer Eigenschaften*
  Neue Eigenschaften können leicht in das bestehende System integriert werden. Dazu muß ein Domain konstruiert werden, das diese Eigenschaften als Elemente enthält. Dieses Domain regelt mit seinen Methoden die Verknüpfungen und Ableitungen der neuen Eigenschaften. Die Interaktion mit anderen Eigenschaftsdomains wird durch eine spezifizierte Schnittstelle geregelt.

- *einfaches Entfernen von Eigenschaften*
  Eigenschaften, die nicht benötigt werden, können entfernt werden, ohne daß die Evaluierung aller anderen Eigenschaften beeinträchtigt wird (natürlich nur, wenn zwischen diesen Domains keine Beziehungen bestehen).

- *einfaches Ersetzen von Eigenschaften*
  Soll ein Eigenschaftsdomain erweitert oder korrigiert werden, wirken sich diese Änderungen nur auf dieses Domain aus. Die Eigenschaften und Regeln anderer Domains bleiben unberührt.

Der nächste Abschnitt behandelt die Evaluierung von Eigenschaften verschiedener Domains.

Verbindung zwischen Eigenschaftsdomains

Die Verknüpfung von Eigenschaften verschiedener Domains sollte nicht Aufgabe der domaineigenenen Methoden sein (da diese Methoden für die Verknüpfung von Elementen „ihres“ Domains zuständig sind).
Wenn eine Verknüpfung von Eigenschaften verschiedener Domains sinnvoll ist, dann besteht auch die Möglichkeit der *Konvertierung* der Eigenschaften in ein gemeinsames Domain.

Beispiel:
Eigenschaft `(>0)` und Eigenschaft `NRange("Integer")` ergibt `NRange("PosInt")`

Die letzte Verknüpfung wird realisiert, indem die Eigenschaft `(>0)` (des Domains der Relationen) in die Eigenschaft `NRange("Positive")` des Domains der Zahlenbereiche (`NRange`) konvertiert wurde. Die Verknüpfung kann dann die Methode `_and` von `NRange` vornehmen.
Jedes Domain hat dazu für jedes andere Eigenschaftsdomain eine Methode, die seine Elemente in eine Eigenschaft (äquivalenter Bedeutung) des „Ziel“–Domains überführt. Eine solche Konvertierungs–Methode können auch Domains besitzen, die keine Eigenschaften als Elemente haben.

Beispiel:
Das Domain `IV` hat als Eigenschaften Intervalle von Zahlenbereichen. Eine Konvertierung eines Zahlenbereichs in ein Intervall wird durch die Methode `NRange::IV` berechnet:
`NRange::IV(NRange("RealNum"))` $\Longrightarrow$ `IV(-infinity, infinity, NRange("RealNum"))` (offenes Intervall von $-\infty$ bis ∞ über den reellen Zahlen)
`NRange::IV(NRange("NonNegInt"))` $\Longrightarrow$ `IV([0], infinity, NRange("Integer"))` (rechts offenes Intervall von 0 bis ∞ über den ganzen Zahlen)
Eine Konvertierung eines Intervalls in einen Zahlenbereich wird durch die Methode `IV::NRange` berechnet:
`IV::NRange(IV(-infinity, infinity, NRange("real")))` $\Longrightarrow$ `NRange("real")`

Auch das Domain `DOM_INT` der ganzen Zahlen kann eine Methode `"NRange"` besitzen, die für jede Zahl (vom Typ `DOM_INT`) die Zugehörigkeit zu einem Zahlenbereich berechnet:
`DOM_INT::NRange(3)` $\Longrightarrow$ `NRange("PosInt")`
`DOM_INT::NRange(-3)` $\Longrightarrow$ `NRange("NegInt")`

Durch diese Konvertierung können die vorhandenen Regeln der Domains auch für die Verknüpfung von Eigenschaften verschiedener Domains verwendet werden.
Ist eine solche Verknüpfung nicht möglich, kann der Ausdruck der Verknüpfung als Eigenschaft angesehen und zur Evaluierung benutzt werden.
Außer der Konvertierung stehen noch zwei weitere Möglichkeiten zur Verfügung, die Evaluierung von Eigenschaften zu beeinflussen: *Regeln* und *Entscheidungsfunktionen*.

Regeln und Entscheidungsfunktionen

Eine *Regel* beschreibt die Verknüpfung von genau zwei Eigenschaften (beliebiger Domains).

Beispiel:
```
NRange("Real")^2 = NRange("NonNegative")
NRange("Negative")^2 = NRange("Positive")
```

```
NRange("Integer")^2 = NRange("NonNegInt")
NRange("NegInt")^2 = NRange("PosInt")
...
```

Diese Regeln beschreiben das Potenzieren einer Zahl aus einem bestimmten Bereich mit 2. Alle definierten Regeln werden *vor* den domaineigenen Methoden abgefragt. Deshalb können Regeln dazu verwendet werden, fehlende oder spezielle Verknüpfungen auszudrücken.
Eine weitere Möglichkeit bieten *Entscheidungsfunktionen*. Diese Funktionen drücken mehrere Regeln in einer Prozedur aus und können abhängig von den übergebenen Parametern beliebige Sachverhalte darstellen.

Beispiel:

```
proc(operator, op_1, op_2)
begin
  if operand = _power then
    if is(op_2, Even) = TRUE and is(op_2 > 0) = TRUE then
      return(table(NRange("RealNum")  = NRange("NonNegative"),
                   NRange("Negative") = NRange("Positive"),
                   NRange("Integer")  = NRange("NonNegInt"),
                   NRange("NegInt")   = NRange("PosInt")
                   # ... weitere Regeln #
                  )[op_1])
    end_if
  end_if;
  return(FAIL)
end_proc
```

Die Funktion beschreibt das Quadrieren einer Zahl aus einem bestimmten Bereich mit einer beliebigen positiven geraden Zahl.

Benutzung der Bibliothek `property`

Die Bibliothek `property` befindet sich derzeit noch in der Entwicklung und wird später in die bestehenden MuPAD-Bibliotheken integriert. Sie stellt verschiedene Funktionen zur Verfügung, die im folgenden kurz beschrieben werden.

Benutzer-Schnittstelle

- Setzen von Eigenschaften
  - `assume(ident, prop)`
    `ident` bekommt Eigenschaft `prop`, bestehende Eigenschaften werden gelöscht
  - `assume(ident, prop <, _Op>)`
    bestehende Eigenschaften von `ident` und `prop` werden mit Operator `_Op` verknüpft und `ident` zugewiesen
  - `assume(rel <, _Op>)`
    Setzen einer Relation `rel`, bestehende Eigenschaften der beteiligten Bezeichner werden mit neuer Eigenschaft verknüpft
- Abfragen von Eigenschaften
  - `properties(ident)`
    Eigenschaften von `ident` werden zurückgegeben

- `properties(expr <, Type>)`
  Eigenschaften von `expr` werden in eine Eigenschaft des Domains `Type` konvertiert und zurückgegeben

- Löschen von Eigenschaften
  - `unassume(ident <, prop>)`
    Eigenschaften (`prop` oder alle) werden gelöscht

- Ableiten von Eigenschaften
  - `is(ident, prop)`
    die Ableitung der Eigenschaften von `ident` nach `prop` wird überprüft
  - `is(expr, prop)`
  - `is(rel)`

Die Bibliothek `property` bietet außerdem Funktionen an, um Regeln und Entscheidungsfunktionen zu setzen:

- Setzen einer Regel oder Funktion
  - `property::setrule(<domain,> prop_1 _op prop_2 = prop_res)`
    im Domain `domain` wird die Regel „aus `prop_1 _op prop_2` folgt `prop_res`“ gespeichert
  - `property::setrule(<domain,> func, func_name <, func_remark>)`
    im Domain `domain` wird die Funktion `func` mit Namen `func_name` (und Kurzbeschreibung) gespeichert

- Abfragen einer Regel oder Funktion
  - `property::getrule(<domain,> prop_1 _op prop_2)`
    im Domain `domain` wird nach einer Regel für `prop_1 _op prop_2` gefragt

- Löschen einer Regel oder Funktion
  - `property::delrule(domain)`
    Löschen aller Regeln und Funktionen
  - `property::delrule(<domain,> prop_1 _op prop_2)`
    Löschen einer Regel
  - `property::delrule(<domain,> func_name)`
    Löschen einer Funktion

Das Laden und Speichern benutzerdefinierter Regeln und Entscheidungsfunktionen wird durch folgende Funktionen unterstützt:

- Laden von Regeln und Funktionen
  - `property::loadrule(domain <, string> <, New>)`
    Regeln und Funktionen von `domain` werden geladen

- Speichern von Regeln und Funktionen
  - `property::saverule(domain <, string> <, New>)`
    Regeln und Funktionen von `domain` werden gespeichert

Eigenschaften

Verschiedene Eigenschaften werden als Domain zur Verfügung gestellt.

- `NRange` – Domain der Zahlenbereiche
- `Rel` – Domain der Relationen
- `IV` – Domain der Intervalle
- `Divs` – Domain für Teilbarkeitseigenschaften
- `Prop` – (Beispiel-) Domain für beliebige Eigenschaften

Die Eigenschaften sind als Methoden und Elemente des Domains `Type` organisiert.

Beispiel:
`Type::RealNum` – Zahlenbereich „reelle Zahlen“
`Type::Integer` – Zahlenbereich „ganze Zahlen“
`Type::Even` – ganze gerade Zahl
`Type::Odd` – ganze ungerade Zahl
`Type::Divs(n)` – Teilbarkeit „teilbar durch n“
`Type::IV([-1], [1], Type::RealNum)` – geschlossenes Intervall $[-1, 1]$ über den reellen Zahlen
`...`

Regelwerke

Zur einfachen Implementierung neuer Regeln (ohne die Erstellung eines neuen Eigenschaftsdomains) können Regeln und Entscheidungsfunktionen in einer Datei gesammelt und mit den Funktionen `property::loadrule` (und `property::saverule`) geladen (und gespeichert) werden.
Die Regeln (und Funktionen) müssen in dem gleichen Format wie beim Aufruf der Funktion `property::setrule` angegeben und mit einem Doppelpunkt `:` abgeschlossen werden.
Zur Darstellung neuer Eigenschaften kann (z.B. zu Testzwecken) das vordefinierte Domain `property::Prop` benutzt werden. Als Argument kann jeder beliebige MuPAD-Ausdruck angegeben werden. Verknüpfungen können nur durch Regeln und Entscheidungsfunktionen ausgedrückt werden. Als einzige Ableitung steht der Vergleich der Operanden zur Verfügung.

Beispiele

```
>> assume(a, Type::NonNegative): assume(b, Type::PosInt):
>> is(a + b, Type::Positive);

                TRUE

>> is(a - b, Type::NonNegative);

                UNKNOWN

>> setprop(c, Type::Divs(10)): is(2*c, Type::Divs(4));

                TRUE

>> is(c, Type::Divs(4));

                FALSE
```

Benutzung des Domains `property::Prop`:

```
>> export(property, Prop):
>> assume(x, Prop("blue")): assume(y, Prop("yellow")):
>> properties(x + y);

                Prop("blue") + Prop("yellow")

>> setrule(Prop, Prop("blue") + Prop("yellow") = Prop("white")):
>> setrule(Prop, Prop("blue") - Prop("yellow") = Prop("black")):
>> properties(x + y);

                Prop("white")

>> is(x - y, Prop("black"));

                TRUE
```

Erstellung eigener Eigenschaften und Ableitungen

Rahmen für ein Eigenschaftsdomain

Ein Domain von Eigenschaften muß einige Bedingungen erfüllen:

Einträge
- Eintrag `TRUE` unter dem Index `"prop"` zur Identifizierung als Eigenschaftsdomain
- seinen Namen als Zeichenkette unter dem Index `"type"`
- eine leere Liste (`[]`) unter dem Index `"ax"` (zur Speicherung von Regeln und Funktionen)

Methoden
- eine Methode `new` zur parameterabhängigen Erzeugung der Eigenschaften des Eigenschaftsdomains
- eine Methode `is` zur Ableitung von Eigenschaften (dieses Domains)
- je eine Methode für die Operationen `_and`, `_or` und die Negierung einer Eigenschaft `_not` (wenn dies sinnvoll ist)
- für jeden Standard-Operator (`+-*/^`) die Verknüpfung zweier Eigenschaften des Domains
- eine Methode für jedes Eigenschaftsdomain, wenn die Eigenschaften in Eigenschaften dieses Domains konvertiert werden können
- Methode `print` zur benutzerfreundlichen Ausgabe der Eigenschaften

Beispiel-Domain

Zur Demonstration soll die Eigenschaft eines Bezeichners, *gerade* oder *ungerade* zu sein, als eigenständiges Domain definiert werden. (Dies ist ein Spezialfall der Eigenschaft *Teiler eines Bezeichners*.)
Zunächst ist das Eigenschaftsdomain zu spezifizieren.

Name: `EO` (`Even` – `Odd`)

Eigenschaften: `Even` – *gerade*, `Odd` – *ungerade*

Methoden:
- `new(x)` erzeugt eine Eigenschaft; is `x` gerade, wird die Eigenschaft `Even` erzeugt, sonst `Odd`
- `is(x, y)` berechnet eine Ableitung „folgt aus `x` die Eigenschaft `y`“
- `_and` UND–Verknüpfung; `_or` ODER–Verknüpfung

- _not Negierung
- _plus, _subtract, _negate, _mult, _divide, _invert, _power – Standard-Verknüpfungen
- Divs – Konvertierung
- print – Ausgabe

Beispiel-Implementierung:

```
EO:= domain("EO"):
EO::info:= "Domain of `Even' and `Odd'":
EO::type:= "EO":
EO::prop:= TRUE:
EO::ax:= []:

EO::new:=
  proc()
  begin
    new(EO, args() mod 2)
  end_proc:

EO::_and:=
  proc()
  begin
    if op(args(1)) <> op(args(2)) then
      property::Null()
    else
      args(1)
    end_if
  end_proc:

EO::_plus:=
  proc()
  begin
    EO(op(args(1)) + op(args(2)))
  end_proc:

EO::_subtract:=
  proc()
  begin
    EO(op(args(1)) + op(args(2)))
  end_proc:

EO::_negate:=
  proc()
  begin
    args()
  end_proc:

EO::_power:=
  proc()

EO::is:=
  proc()
  begin
    if domtype(args(1)) <> EO
       or domtype(args(2)) <> EO then
      UNKNOWN
    else
      bool(op(args(1)) = op(args(2)))
    end_if
  end_proc:

EO::_or:=
  proc()
  begin
    property::Null()
  end_proc:

EO::_mult:=
  proc()
  begin
    EO(op(args(1)) * op(args(2)))
  end_proc:

EO::_divide:=
  proc()
  begin
    property::Null()
  end_proc:

EO::_invert:=
  proc()
  begin
    property::Null()
  end_proc:
```

```
  begin
    args(1)
  end_proc:

EO::Divs:=
  proc()
  begin
    if op(args() = 0) then
      Divs(2)
    else
      not(Divs(2))
    end_if
  end_proc:
```

```
EO::print:=
  proc()
  begin
    if op(args()) = 0 then
      "Even"
    else
      "Odd"
    end_if
  end_proc:
```

Zur Verknüpfung von Standard-Typen bekommt das Domain `DOM_INT` die Methode `EO`:

```
sysassign(DOM_INT::EO, proc()
                       begin
                         EO(args() mod 2)
                       end_proc):
```

Beispiel-Anwendung:

```
>> assume(x, EO(0));

                         Even

>> assume(y, EO(1));

                         Odd

>> is(x, EO(0));

                         TRUE

>> is(x * y, EO(0));

                         TRUE

>> is(x + y, EO(0));

                         FALSE

>> is(2*(x + y), EO(0));

                         TRUE
```

Möglichkeiten der Nutzung algebraischer Eigenschaften

Zur Nutzung algebraischer Eigenschaften sind verschiedene Möglichkeiten vorstellbar.
Viele Systemfunktionen (`int`, `limit`, `sum`...) nutzen bereits zusätzliche Informationen über die Eigenschaften von Bezeichnern. Durch die Zuweisung von Eigenschaften wie `assume(x > 0)` oder `assume(x, Type::RealNum)` können so differenziertere Lösungen gefunden werden.
Mit einer globalen Zuweisung von Eigenschaften an alle Bezeichner könnten Berechnungen wesentlich vereinfacht werden. (Beispiel: Berechnung im Bereich der reellen statt der komplexen Zahlen.)

Weitere Möglichkeiten

Wenn die Frage nach einer Eigenschaft eines Bezeichners den Wert `UNKNOWN` ergibt, wird die Evaluierung ohne diese Information fortgesetzt.
An dieser Stelle ist bei einer interaktiven Berechnung eine Frage an den Benutzer denkbar.

```
>> x*(-1)^n;

 > is(n, Even)  (y/n)?> n
 > is(n, Odd)  (y/n)?> y

        -x
```

Der Benutzer wird direkt in den Evaluierungsprozeß eingebunden und kann Einfluß auf das Ergebnis nehmen und gleichzeitig mehr Verständnis für die ablaufenden Algorithmen gewinnen.
Wesentlich aufwendiger ist in diesem Fall das Berechnen mehrerer Lösungen abhängig von möglichen Werten der beteiligten Bezeichner.

```
>> x*(-1)^n;

         /
         |  x,          if is(n, Even)
         |
       <   -x,          if is(n, Odd)
         |
         |  x*(-1)^n    else
         \
```

Dem Benutzer könnten dann Ausgaben mit umfangreichen Informationen zur Verfügung gestellt werden, wie hier an einem Beispiel gezeigt wird.

Ein tutorielle Lernsoftware zur Algebra der Sekundarstufe I

Frank Postel
Universität–Gesamthochschule Paderborn, frankp@uni-paderborn.de

Computer–Algebra–Systeme waren anfangs nur auf Großrechnern einsetzbar. Insbesondere durch die seit Mitte der 80er Jahre ständig wachsende Leistungsfähigkeit von PC haben sie auch in den Bereichen von Schule und Universitäten weite Verbreitung gefunden.
Mathematica, Maple V und Derive gelten als die wohl populärsten dieser Systeme in der Gegenwart[1], zu denen in letzter Zeit auch immer mehr das System MuPAD gezählt werden darf.
Dieser Artikel nennt Probleme des Einsatzes dieser Systeme im Mathematikunterricht der Sekundarstufe I, formuliert Anforderungen an eine geeignete Software und beschreibt das Konzept einer Software, die diesen Anforderungen gerecht zu werden versucht und sich darüberhinaus in Form eines flexiblen tutoriellen Systems präsentiert.
Die Software ist in MuPAD implementiert und entstand in Zusammenarbeit mit der Cornelsen Software GmbH.

Zu Anforderungen der Sekundarstufe I an Computer–Algebra–Systeme

Eine Analyse von Computer–Algebra–Systemen hinsichtlich ihrer Tauglichkeit zur Verwendung im Mathematikunterricht der Sekundarstufe I (hierbei insbesondere im Rahmen der Algebra) muß u.a. folgende Aspekte berücksichtigen:

1. Das Computer–Algebra–System muß mit rationalen Zahlen in reiner und gemischter Schreibweise rechnen können.

2. Es muß hinsichtlich der Division von Termen die beiden Zeichen „:" und „/" unterscheiden können. Beispiel: $\frac{4}{6} = \frac{4:2}{6:2} = \frac{2}{3}$.

3. Die Reihenfolge der Glieder von Termen und Untertermen darf nicht automatisch festgelegt werden, etwa durch eine vom Computer–Algebra–System vorgegebene interne Anordnung. Eine solche automatische Sortierung von Ausdrücken[2] kann sonst dazu führen, daß der Schüler einzelne Rechenschritte des Systems nicht oder nur schwer nachvollziehen kann.

4. Das Computer–Algebra–System muß folgende Aufgabentypen unterstützen:

[1] In den letzten Jahren hat sich Derive mehr in Richtung Lehrsoftware entwickelt, deren Verbreitung in Schulen insbesondere durch die vor kurzer Zeit vorgenommene Portierung dieser Software auf den Taschenrechner TI-92 von Texas Instruments Inc. stark zugenommen hat.

[2] Im folgenden repräsentiert ein *Ausdruck* einen Term, eine Gleichung oder ein lineares Gleichungssystem.

- Elementare Termumformung wie die Anwendung der Kommutativ- und Assoziativgesetze oder des Distributivgesetzes, Termumformungen wie beispielsweise das Herstellen eines Hauptnenners oder das Kürzen bei rationalen Termen.
- Lösen linearer Gleichungen (auch Bruchgleichungen) mit und ohne Parameter.
- Lösen linearer Ungleichungen (auch Bruchungleichungen) mit und ohne Parameter, dabei ggf. Berücksichtigung von Fallunterscheidungen.
- Lösen linearer Gleichungssysteme, wahlweise nach dem Gleichsetzungsverfahren, Einsetzungsverfahren und dem Additionsverfahren.
- Lösen quadratischer Gleichungen, Wurzelgleichungen, Exponentialgleichungen, logarithmischen Gleichungen und goniometrischen Gleichungen.
- Bestimmung von Lösungsmengen und ggf. von Definitionsbereichen.

5. Die einzelnen Berechnungsschritte des Computer–Algebra–Systems – sei es zum Umformen von Termen oder zum Lösen von Gleichungen – müssen den Anforderungen der Sekundarstufe I hinsichtlich des Rechnens in kleinen Schritten genügen. Beispiele sind:

$$\frac{8}{12} = \frac{8:4}{12:4} = \frac{2}{3}, \quad \frac{2\cdot 15}{5\cdot 4} = \frac{1\cdot 3}{1\cdot 2} = \frac{3}{2} \quad \text{(Kürzen auf dem Bruchstrich)},$$

$$2\left(x-(1+t)\right)+x \overset{\text{DG}}{=} 2(x-1-t)+x \overset{\text{DG}}{=} 2x-2-2t+x \overset{\text{KG}}{=} -2-2t+2x+x \overset{\text{V}}{=} -2-2t+3x\,.$$

Keines der zur Zeit auf dem Markt befindlichen Computer–Algebra–Systeme erfüllt auch nur Teile dieser Anforderungen hinreichend zufriedenstellend. Sie führen in der Regel eine automatische Zusammenfassung und Sortierung der Glieder von Termen durch, d.h. sie wenden automatisch – und nicht auf expliziten Wunsch des Benutzers – beispielsweise Assoziativ- und Kommutativgesetze sowie das Distributivgesetz an. Auf die Eingabe eines Ausdrucks hin, antworten die Computer–Algebra–Systeme in der Regel unmittelbar mit einem bereits vereinfachten, d.h. einem zur Eingabe lediglich mathematisch äquivalenten Ausdruck, der beispielsweise durch vorhergehendes (d.h. automatisches, implizites) Kürzen, Bilden des Hauptnenners oder Ausklammern gemeinsamer Faktoren aus der Eingabe entstanden ist.
Hinsichtlich der Behandlung umfangreicherer Aufgaben wie beispielsweise dem Lösen von Gleichungen oder Gleichungssystemen präsentieren Computer–Algebra–Systeme in der Regel den Weg zur Lösung in Form einer „black–box", führen mehrere Schritte gleichzeitig aus und geben keine oder nur spärliche Informationen über den Berechnungsablauf.
Computer–Algebra–Systeme können als Werkzeug zum Training bereits erlernter Fähigkeiten dienen und so eventuell die Möglichkeit schaffen, neue Aufgaben im Unterricht erschließen zu können. Wie jedoch können Computer–Algebra–Systeme den Schüler schon beim Erlernen *elementarer* Unterrichtseinheiten unterstützen?

Ein flexibles tutorielles System für die Algebra der Sekundarstufe I

G. Holland [1] beschreibt die Anforderungen an sogenannte intelligente tutorielle Systeme (IST). Sie sollen „für den von ihm unterstützten Unterrichtsgegenstand die Rolle eines menschlichen Tutors übernehmen". Dabei definieren sich die Anforderungen an ein solches System über ein hohes Maß an Flexibilität und Adaptivität. „Das System ist flexibel, wenn es der/dem Lernenden die Auswahl an Lernangeboten und Repräsentationsformen in angemessenem Umfang ermöglicht. Es ist darüberhinaus adaptiv, indem es Lernstil und Lernzustand der/des jeweiligen Lernenden berücksichtigt."
Im folgenden wird eine Software vorgestellt, welche die auf Seite 73 aufgelisteten Merkmale einer Software für den Einsatz im Mathematikunterricht der Sekundarstufe I erfüllt und darüberhinaus den Anspruch erhebt, den Anforderungen einer tutoriellen Software hinsichtlich Flexibilität zu erfüllen.
Der Schüler gibt dem System eine zu einem bestimmten Aufgabentyp – wie z.B. dem Lösen linearer Gleichungen – passende Aufgabe vor. Die Aufgabe kann im Rahmen des Aufgabentyps von dem Schüler frei gewählt werden (z.B. $2(x+1)-2=x+2$).

Der Schüler macht seinen ersten Berechnungsschritt, den er dem System vorschlägt, hier z.B. die Anwendung des Distributivgesetzes. Das System überprüft daraufhin, ob der Berechnungsschritt *sinnvoll* ist (s.u.) und antwortet – falls nicht – mit einem entsprechenden Hinweis. Ist das eingegebene Kommando vom System akzeptiert worden, so ist der Schüler aufgefordert, das Ergebnis der Berechnung hinzuschreiben – hier z.B. $2x + 2 \cdot 1 - 2 = x + 2$ oder auch $2x = x + 2$ – welches das System daraufhin auf Korrektheit überprüft.

An einer beliebigen Stelle eines Berechnungsablaufs kann der Schüler das System bitten, den nächsten sinnvollen Berechnungsschritt vorzuschlagen. Im obigen Beispiel würde das System mit dem Vorschlag antworten, auf beiden Seiten der Gleichung 2 zu addieren. Darüberhinaus kann das System – wieder auf ausdrücklichen Wunsch des Schülers – zu jeder Zeit das Ergebnis eines Berechnungsschrittes vorrechnen. Fragt beispielsweise der Schüler das System nach dem Ergebnis des von ihm vorgeschlagenen Berechnungsschrittes „Anwenden des Distributivgesetzes", so würde das System mit dem Resultat $2x + 2 - 2 = x + 2$ antworten. Auf den Berechnungsschritt „addiere 2 auf beiden Seiten" würde das System die Gleichung $2(x + 1) - 2 + 2 = x + 4$ (bzw. $2(x + 1) = x + 4$, siehe Seite 75) liefern.

Ein Berechnungsschritt wird hierbei als *sinnvoll* charakterisiert, wenn dieser – die syntaktische Korrektheit vorausgesetzt – hinsichtlich der Hinführung zur Lösung unmittelbar zu einem *vereinfachten* Ergebnis führt. Im obigen Beispiel sind u.a. „Anwenden des Kommutativgesetzes" oder „Addiere 5 auf beiden Seiten" keine sinnvollen Berechnungsschritte, obwohl sie zwar mathematisch korrekt sind aber nicht unmittelbar zu einem vereinfachten Zwischenresultat zur Berechnung der Lösung führen. Aber auch ein Berechnungsvorschlag vom Schüler wie „Addiere 1 auf beiden Seiten" würde vom System als nicht sinnvoll abgelehnt werden, obwohl man geneigt sein könnte, das entsprechende Zwischenresultat als vereinfacht zu betrachten. Hier besteht durchaus Diskussionsbedarf!

Auf Seite 76 wird versucht, dem Leser eine Charakterisierung von sinnvollen gegenüber nicht sinnvollen Berechnungsschritten zu liefern.

Das System ist also hinsichtlich seiner tutoriellen Eigenschaften in der Lage, an einer beliebigen Stelle eines vollständigen Lösungsweges für eine von dem Schüler frei vorgegebene und zu einem bestimmten Aufgabentyp passende Aufgabe,

1. einen von dem Schüler vorgegebenen Berechnungsschritt auf sinnvoll oder nicht sinnvoll zu überprüfen,
2. auf Wunsch des Schülers einen sinnvollen Berechnungsschritt vorzuschlagen,
3. ein von dem Schüler nach Anwendung eines konkreten Berechnungsschrittes angegebenes Resultat auf Korrektheit zu überprüfen,
4. auf Wunsch des Schülers das auf Anwendung eines konkreten Berechnungsschrittes folgende Resultat auszurechnen.

Die Software gestattet, eine gewisse Anzahl an Optionen einzustellen. Darunter fallen u.a. Einstellungen wie Anzahl der Nachkommastellen von Dezimalbrüchen, das zu verwendende Verfahren zum Lösen von linearen Gleichungssystemen (siehe Seite 73) oder die Einstellung zweier Benutzerebenen („basic" oder „advanced").

Mit der Auswahl einer bestimmten Benutzerebene wird festgelegt, wie feinschrittig bestimmte Berechnungen, die vom System ausgeführt werden, ablaufen sollen. So liefert das System auf Benutzerebene „basic" bei Addition von $\frac{1}{3}$ auf beiden Seiten der Gleichung $2(x + 1) - \frac{1}{3} = x + 2$ das Resultat $2(x + 1) - \frac{1}{3} + \frac{1}{3} = x + 2 + \frac{1}{3}$, während auf Benutzerebene „advanced" das Resultat $2(x + 1) = x + 2 + \frac{1}{3}$ geliefert werden würde. Es sei ausdrücklich betont, daß die Benutzerebene nur die Berechnungen des Systems beeinflußt. Der Schüler ist frei, mit beiden Lösungen zu antworten!

Die tutorielle Komponente der Software wirft eine Reihe von didaktischen und technischen Fragen auf. Soll beispielsweise zugelassen werden, daß der Schüler mehrere Befehle auf einmal auf einen Ausdruck anwendet? Die Software gestattet lediglich die Anwendung eines Befehles auf einen Term oder eine Gleichung. Wie restriktiv soll sich das System bei nicht sinnvollen Befehlen dem Schüler gegenüber verhalten? So ließe sich diskutieren, ob Befehle des Schülers akzeptiert werden sollen, die nicht zu einer direkten Vereinfachung des Ausdrucks führen. Hierbei könnte die Software den Schüler darauf hinweisen, daß er sich von einem „optimalen" Lösungsweg weggegeben hat[3].

[3] Die Vorgehensweise der hier beschriebenen Software wird exemplarisch weiter unten beschrieben.

Ein technisches Problem ist die Anwendung von Heuristiken, welche die Software verwendet, um beispielsweise entscheiden zu können, ob ein von dem Schüler vorgegebener Befehl sinnvoll ist oder das Ende einer Berechnung erreicht ist. Werden dabei Heuristiken falsch angewendet, so kann es entweder dazu führen, daß die Software nicht in der Lage ist, eine Berechnung zu Ende zu führen, oder aber einen schuluntypischen Weg zur Lösung verfolgt.

Zur Terminologie „sinnvoller Berechnungsschritte" – Beispiele

Der Autor verzichtet in dem vorliegenden Artikel, Beispiele von Berechnungen der Software aufzuzeigen. Dem Leser ist zum einen der schultypische Weg zur Lösung bei den auf Seite 73 genannten Aufgabentypen bekannt und wird – in seiner kritischen Haltung – eigenen Tests der Software vermutlich mehr Glauben als dem hier geschrieben Wort schenken. Desweiteren bietet der Artikel keine Möglichkeit, den interessanten Teil der Software der Überprüfung der zum einen vom Schüler gewählten Berechnungen zum Weg zur Lösung und zum anderen der vom Schüler berechneten Zwischenschritte, exemplarisch darzustellen.

Die Angaben in der folgende Tabelle versuchen, den Begriff der „sinnvollen Berechnung" exemplarisch zu verdeutlichen.[4] Dabei finden sich in der *ersten Spalte* konkrete Aufgaben, die dem Schüler vorliegen. Die *zweite Spalte* gibt an, welcher Berechnungsschritt vom System – bezogen auf die jeweilige Aufgabe – vorgeschlagen werden würde. Das zu einem konkreten Berechnungsschritt vom System berechnete Ergebnis findet sich in der *dritten Spalte*.[5]

In der *vierten Spalte* steht eine (nicht vollständige) Auswahl an sinnvollen Befehlen zur Verfügung, die vom Schüler – bezogen auf die jeweilige Aufgabe – eingegeben werden können. Zu diesen Befehlen mögliche Ergebnisse, die vom Schüler als Antwort genannt werden können und vom System als korrekt akzeptiert werden würden, finden sich in der *fünften* Spalte. Schließlich gibt die *sechste Spalte* eine (nicht vollständige) Auswahl an Befehlen an, die als nicht sinnvoll zurückgewiesen werden würden.

[4] Diese Angaben entsprechen dem gegenwärtigen Stand der Entwicklung (Änderungen vorbehalten).

[5] Dabei ist der Benutzerlevel „advanced" zugrundegelegt, siehe oben.

Aufgabe	System		Schüler		
		Resultat	sinnvoll	mögliche Resultate	nicht sinnvoll
$3\left(2x-\frac{1}{2}\right)=4-2(1-x)$	DG	$6x-\frac{3}{2}=4-2+2x$	DG	$6x-\frac{3}{2}=4-2+2x$ $6x-\frac{3}{2}=2+2x$ $3\cdot 2x-3\cdot\frac{1}{2}=4-2(1-x)$ $6x-3\cdot\frac{1}{2}=4-2(1-x)$	V, KG
		$3\left(2x-\frac{1}{2}\right)+2(1-x)=4$	$+2(1-x)$	$3\left(2x-\frac{1}{2}\right)+2(1-x)=4-2(1-x)+2(1-x)$ $3\left(2x-\frac{1}{2}\right)+2-2x=4$ $6x-\frac{3}{2}+2-2x=4$	
$6x-\frac{3}{2}=4-2+2x$	V	$6x-\frac{3}{2}=2+2x$	V	$6x-\frac{3}{2}=2+2x$ $4x-\frac{3}{2}=2$	$+x,\ -1$, DG, KG, Kürzen
		$-\frac{3}{2}=4-2+2x-6x$	$-6x$	$-\frac{3}{2}=4-2+2x-6x$ $-\frac{3}{2}=2-4x$ $6x-6x-\frac{3}{2}=4-2+2x-6x$	
		$6x-\frac{3}{2}+2=4+2x$	$+2$	$6x-\frac{3}{2}+2=4+2x$ $6x+\frac{1}{2}=4+2x$	
$4x=2+\frac{3}{2}$	HN	$4x=\frac{2\cdot 2}{2}+\frac{3}{2}$	HN	$4x=\frac{2\cdot 2}{2}+\frac{3}{2}$ $4x=\frac{7}{2}$ $x=\frac{7}{8}\quad(*)$	$-4x, :2, -2, -\frac{3}{2}$, V
		$x=\frac{2+\frac{3}{2}}{4}$	$:4$	$x=\frac{2+\frac{3}{2}}{4}$ $x=\frac{\frac{7}{2}}{4}\quad(**)$	

(*) Die Aufgabe ist zu Ende gerechnet; der Schüler muß dann die entsprechende Lösungsmenge (hier: $L=\{\frac{7}{8}\}$) hinschreiben.

(**) Die Aufgabe ist noch nicht beendet, das Resultat kann weiter vereinfacht werden.

Die technische Realisierung der Software

Die beschriebene Software läßt sich in zwei Teile gliedern. Der eine, mathematische Teil basiert auf dem Computer–Algebra–System *MuPAD* und wird in der *MuPAD* eigenen Programmiersprache implementiert. Die didaktische Spezifizierung mathematischer Berechnungen ist hierbei größtenteils von Cornelsen Software vorgegeben.

Die Bedienungsoberfläche zur Eingabe von Termen und Gleichungen stellt den zweiten Teil dar und präsentiert sich dem Benutzer in Form eines Editors für 2-dimensionale Formeln, der von Cornelsen Software unter Windows 95 entwickelt wird. Zweidimensional heißt in diesem Zusammenhang, daß die Formel in der gängigen mathematischen Notation eingegeben wird, z.B. $\frac{a+b}{c}$, und nicht „linear" wie beispielsweise in Programmiersprachen als

$(a + b)/c$.
Der folgende Ausschnitt aus einem Testprogramm veranschaulicht die Arbeitsweise des Formeleditors. Befehle werden über Menüpunkte ausgewählt, wobei ein Teil der Befehle auch über Tastatureingaben zur Verfügung stehen (hier: „V" für das Kommando „Vereinfache").
Der Editor stellt sogenannte *Templates* für bestimmte mathematische Strukturen zur Verfügung, hier zu sehen am Beispiel der Eingabe eines Bruches.

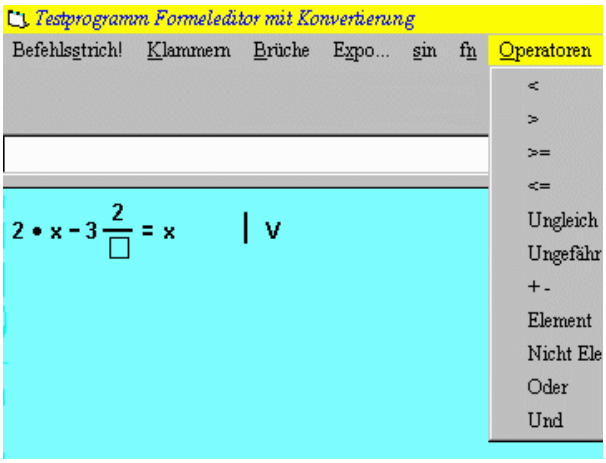


Abbildung 25: Ausschnitt aus einem Testprogramm zum Formeleditor

Die Kommunikation zwischen dem Formeleditor und *MuPAD* basiert auf COM unter Windows 95. Eine Formel im Formeleditor wird nach deren Eingabe in eine Zeichenkette gewandelt, die nach *MuPAD* gesendet wird. *MuPAD* übersetzt wiederum die erhaltene Zeichenkette in ein internes Datenformat, führt entsprechende Berechnungen aus und sendet das Ergebnis der Berechnungen – wieder in Form einer Zeichenkette – zum Formeleditor, der das Ergebnis als eine 2-dimensionale Formel grafisch darstellt.

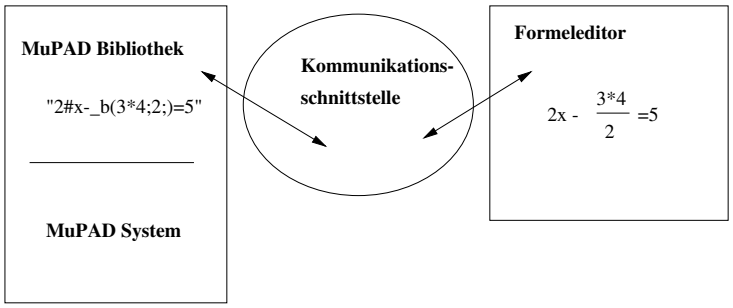


Abbildung 26: Die Komponenten der Software

Die Kommunikationsschnittstelle wurde von Klaus Drescher (UGH Paderborn, *MuPAD* Group) entwickelt.

Bemerkungen zu *MuPAD*

MuPAD stellt ein geeignetes Werkzeug zur Entwicklung des Systems dar. Es bietet dem Benutzer und Programmierer eine Vielzahl an mathematischen Funktionen, die auf modernen Algorithmen basieren, eine Reihe von Datentypen wie z.B. ganze Zahlen, rationale Zahlen, Zeichenketten oder Tabellen, sowie entsprechende Operationen auf diesen Datentypen.

Die für die Realisierung der tutoriellen Software entscheidende Eigenschaft von *MuPAD* ist jedoch die Möglichkeit, beliebige benutzerdefinierte Datentypen – sogenannte *Domains* – erzeugen zu können. Diese lassen sich in nahezu alle *MuPAD*-Funktionen integrieren (man spricht dann von sogenannten *generischen* Funktionen) und können für alle mathematischen Operationen (wie beispielsweise Addition, Multiplikation, Division, etc.) neu definiert werden. Hierdurch lassen sich die auf Seite 73 erwähnten, von einem Computer–Algebra–System i.d.R. implizit vorgenommenen Umformungen einer Benutzereingabe, nahezu vollständig verhindern bzw. kontrolliert ausprogrammieren.
Natürlich beherrscht *MuPAD* das Lösen von Gleichungen, Ungleichungen oder Gleichungssystemen, was in der hier beschriebenen Software zur Bestimmung der Wertegleichheit von Gleichungen oder Teilausdrücken benötigt wird. Um sinnvolle Berechnungsschritte vorschlagen oder auch um das Ende einer Berechnung erkennen zu können, werden mächtige Funktionen von *MuPAD* zum Umformen und Vereinfachen von Ausdrücken (z.B. `simplify`, `gcd` und `normal`) verwendet.

Ausblick

Mit dieser Software wird ein Computer–Algebra–System zur Verfügung stehen, das sich an den Anforderungen des Mathematikunterrichts der Sekundarstufe I orientiert.[6]
Das vorgestellte System reiht sich zudem nicht in die Sparte klassischer Lernsoftware ein, wo dem Schüler eine Sammlung (feststehender) Aufgaben und zugehörigen Lösungsansätzen zu einem konkreten Thema präsentiert wird. Die intelligente tutorielle Komponente dieser Software hingegen gibt dem Schüler eine praktisch unendliche Vielzahl an Übungsaufgaben an die Hand. Zu jeder gewählten Aufgabe kann er darüberhinaus – auf seinen expliziten Wunsch – das System nach einer schultypischen Musterlösung fragen.
Die Software befindet sich gegenwärtig noch in Entwicklung. Ein erster Teil dieser Software wird voraussichtlich noch im Herbst dieses Jahres auf dem Markt erscheinen. Er umfaßt lineare Termumformungen, Lösen linearer Gleichungen und Ungleichungen, Lösen linearer Bruchgleichungen und -ungleichungen (jeweils wahlweise mit und ohne Parameter) und das Lösen linearer Gleichungssysteme mit zwei und drei Gleichungen. Darüberhinaus werden sich grafische Ausgaben von Funktionsgraphen anfertigen lassen. Für die weitere Entwicklung des Systems ist das Lösen von quadratischen Gleichungen, Wurzelgleichungen, Exponentialgleichungen, Logarithmengleichungen und goniometrischen Gleichungen vorgesehen.
Es stellt sich die Frage, an welche Lerngruppe sich das System richtet. Realschule und/oder Gymnasium? Leistungsschwache oder auch leistungsstarke Schüler? Diese Frage wird sich nach Einführung der beschriebenen Software in Unterrichtseinheiten beantworten lassen, auf die man sicherlich gespannt sein darf.
Hinsichtlich der u.a. auf Seite 74 erwähnten didaktischen und technischen Fragestellungen wird die Software – so ist zu hoffen – kritische und fruchtbare Diskussionen über deren Einsatz und Konzeption hervorrufen. Der Autor erlaubt sich diesbezüglich den Artikel mit dem folgenden Zitat von G. Holland [1] zu beenden: „Die Verwendung des Computers als intelligenter Tutor wird in der deutschsprachigen Mathematikdidaktik hingegen vorerst nur ein geringes Interesse entgegengebracht. Ein wesentlicher Grund dürfte der große Aufwand sein, welcher mit Entwurf und Implementation derartiger Systeme verbunden ist. Dieser Aufwand läßt sich nur reduzieren, wenn man die Anforderungen an tutorielle Systeme nicht zu hoch schraubt. [...]".

Literatur

[1] G. Holland: *Schülermodellierung bei aufgabenorientierten intelligenten tutoriellen Systemen.* Computer-Mensch-Mathematik. Beiträge zum 6. Internationalen Symposium für Didaktik der Mathematik vom 23. bis 27.9.1990. Hölder-Pichler-Tempsky, Wien. B.G. Teubner, Stuttgart. 1990.

[6] Im Gegensatz dazu ist in der gegenwärtigen Unterrichtspraxis eher zu beobachten, daß sich der Unterricht – wenn Computer eingesetzt werden – an den vorhandenen Softwareprodukten orientieren muß.

MuPAD Pro: Das Computeralgebra-System für Windows 95 / NT

Oliver Kluge
SciFace Software GmbH & Co. KG, Technologiepark 12, 33100 Paderborn

Fortschritt durch Technologie bedeutet die Entlastung des Menschen von automatisierbarer Arbeit und bewirkt damit einen neuen Freiraum zu kreativer Tätigkeit. Moderne Technik und Naturwissenschaft beruht weitgehend auf der Anwendung kompliziert erscheinender Formeln und aufwendigen Rechenverfahren. So komplex dem Anwender die dahinterliegenden mathematischen Grundlagen auch erscheinen mögen, so basieren diese trotzdem zu 97% auf Routine und sich wiederholenden gedanklichen Konstruktionen. Diese können zur Schaffung von schöpferischer Freiheit automatisiert werden: Genau das verwirklicht Computeralgebra. Durch Einbeziehung von mathematischem und verfahrensmäßigem Expertenwissen schafft sie ein mächtiges Werkzeug für neue Erkenntnis und Anwendung in Technik und Naturwissenschaft.

Die von Prof. Dr. Fuchssteiner aufgezeigten Perspektiven durch Computeralgebra spiegeln sich in MuPAD Pro wider, dem modernen Computeralgebra-System, das sich sowohl an den Entwickler in der Industrie als auch an den Schüler im Mathematikunterricht richtet.

MuPAD Pro ist das erste in Deutschland zur Marktreife entwickelte Computeralgebra-System. Es wird in einer Kooperation der SciFace Software GmbH & Co. KG und der Universität-Gesamthochschule Paderborn weiterentwickelt und gepflegt. MuPAD Pro ist das ultimative Werkzeug für jede Art von Mathematik am Computer. Sein Ziel ist die Abschaffung aller Unannehmlichkeiten, die sich aus umständlichem und oft fehlerträchtigem Lösen von mathematischen Standardaufgaben und Umformen komplizierter Gleichungen ergeben.

MuPAD Pro kann auf verschiedene Arten benutzt werden:

- als ein alleskönnender Taschenrechner (Formel/Ausdruck eintippen und fertig)
- um in der MuPAD eigenen hochstrukturierten Programmiersprache zu programmieren.

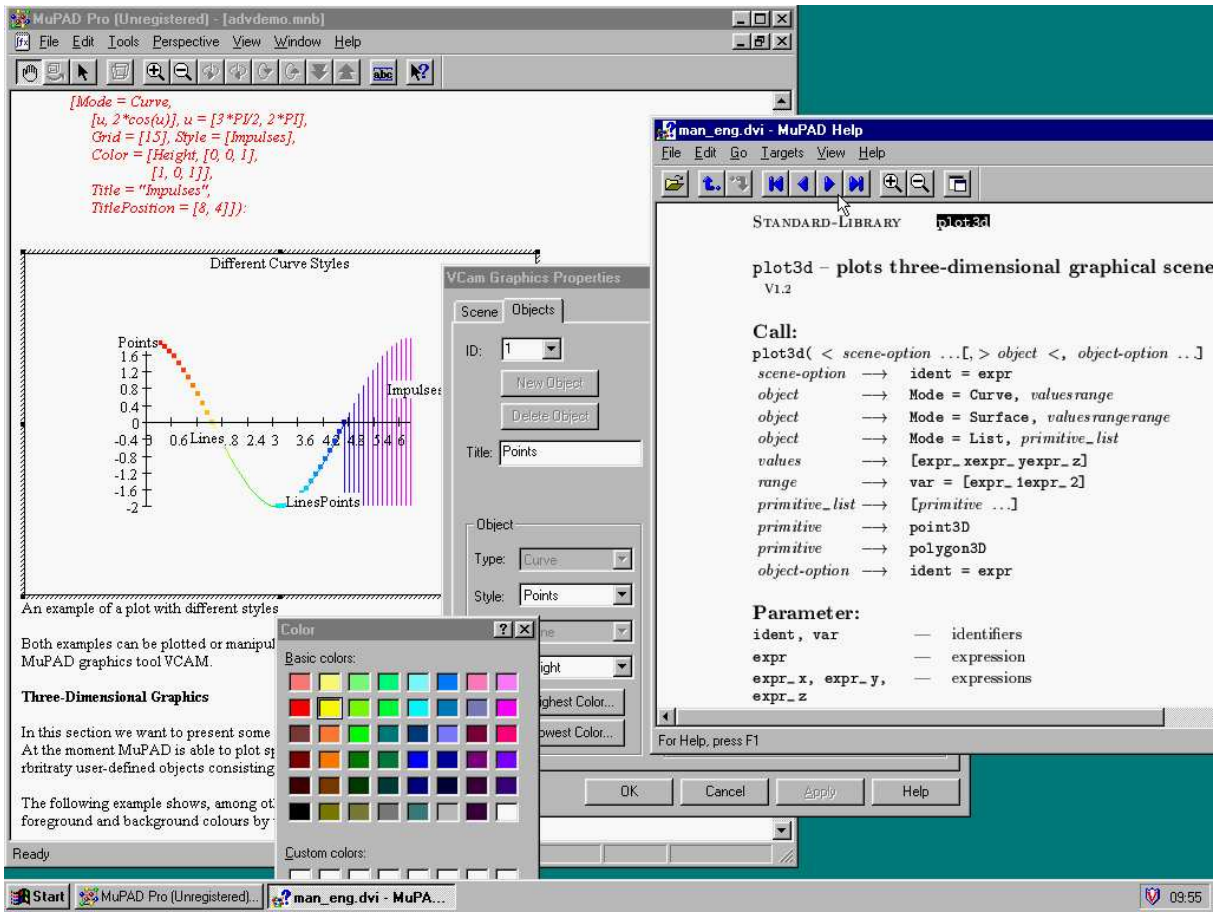
Bei beiden Varianten sind keine großen Mathematikkenntnisse vonnöten, nur ein problemorientiertes Verständnis wird vorausgesetzt.

MuPAD Pro kann Funktionen nicht nur berechnen, sondern auch visualisieren. MuPAD zeichnet mathematische Sachverhalte. Es erstellt grafische Szenen mit frei definierbaren Objekten, z.B. Kurvenverläufen, statistischen Daten, dreidimensionalen Körpern. Die Szenen sind frei dreh- und vergrößerbar. Alle Objekte können interaktiv, also ohne umständliche Kommandos manipuliert werden.

Durch sein Modulkonzept ist MuPAD Pro problemlos erweiterbar: Programme, die Nutzer in anderen Programmiersprachen geschrieben haben, können leicht an MuPAD Pro angebunden werden und Daten mit MuPAD Pro austauschen. So kann man durch die Flexibilität von MuPAD Pro auch fremde Spezialroutinen nutzen und muß nicht auf die eigenen Speziallösungen verzichten.

MuPAD Pro basiert auf der seit Jahren bewährten Kerntechnologie von MuPAD, die an der Universität Paderborn entwickelt wurde. MuPAD Pro zeichnet sich durch die völlig neu gestaltete Benutzeroberfläche aus. Das Rechenblatt, ein sogenanntes Notebook, vereint Rechnungen, Texte, Graphiken und externe Applikationen in einem Dokument. Das umfangreiche Hypertext-Hilfesystem kann an die Bedürfnisse des einzelnen Benutzers durch frei definierbare Hyperlinks, Anmerkungen und Bookmarks angepaßt werden.

MuPAD Pro: Das Computeralgebra-System für Windows 95 / NT



Nach der ersten öffentlichen Präsentation von MuPAD Pro auf der CeBIT '97 wird Ende Oktober die SciFace Software GmbH & Co. KG mit dem Produkt MuPAD-Pro für Windows 95/NT in der Version 1.4 starten. Der Einführungspreis für eine Einzellizenz wird mit 298 Mark deutlich unter dem ähnlicher Produkte mit weit geringerer Funktionalität liegen. Zudem bleibt man der Überzeugung treu, daß der Zugang zu Werkzeugen wie MuPAD jedem offen stehen sollte, und bietet daher im kommenden Jahr MuPAD-Light[1] an, eine reine Terminalversion von MuPAD-Pro für den wissenschaftlichen und nichtkommerziellen Bereich. Desweiteren sind auf MuPAD Pro basierende OEM-Produkte, wie interaktive Bücher, Schulmaterialien für die Sekundarstufe II und Lernsoftware für die Sekundarstufe I in Entwicklung.

Bei weiteren Fragen wenden Sie sich bitte an:

SciFace
Scientific Interfaces

SciFace Software GmbH & Co. KG
Technologiepark 12
33100 Paderborn

Telefon : 05251 / 6407-51
Telefax : 05251 / 6407-99
Email : info@sciface.com

[1] MuPAD-Light wird nur als “single user”-Lizenz vergeben.

First *MuPAD* User Workshop '97

Ralf Hillebrand
Universität GH Paderborn, tonner@uni-paderborn.de

On the 22th and 23th of September 1997 the first MuPAD User Workshop takes place in Paderborn, Germany.
It is supported by the Land Nordrhein-Westfalen *and* SciFace Software *(this company is the new founded distributor of MuPAD and will be presented on this workshop).*
Here are the abstracts of the talks.

Oliver Kluge (Paderborn): *MuPAD* Pro for Windows 95 / NT

SciFace Software introduces *MuPAD* Pro for Windows 95 / NT, based on the kernel technology of *MuPAD* 1.4.0. The main aspect of the talk will be the user interface and communication library of *MuPAD* Pro. The concept of the notebook with embedded graphics and the improved help system are presented. A very important aspect for third party developers is the communication library. This library enables the easy access to the *MuPAD* kernel from external applications. Finaly the future plans of the development of *MuPAD* Pro are discussed.

Frank Postel (Paderborn): Kurze Schritte mit langen Beinen - Wie man mathematische Expertensysteme für die Schule bremst

War vor wenigen Jahren das Computer-Algebra-System *MuPAD* vermutlich nur Insidern ein Begriff, erfreut es sich seit einiger Zeit wachsender Verbreitung und Beliebtheit. Auch hinsichtlich der Verwendung als Unterrichtswerkzeug für Schulen läßt sich eine immer größer werdende Lehrergemeinde erkennen, die das System *MuPAD* für ihren Unterricht in Betracht ziehen.
Dabei stellt sich die Frage, ob ein derart komplexes mathematisches Expertensystem wirklich eine geeignete Software für den Schulunterricht darstellen kann. Ist *MuPAD* in der Lage, sich gegen Systeme wie Derive (bzw. dem TI-92) oder MathPlus zu bewehren, die in den vergangenen Jahren zu den populärsten Vertretern ihrer Gattung in Schulen gehören?
Der Vortrag geht dieser Frage nach und zeigt Einsatzmöglichkeiten von *MuPAD* in Schulen, die von kleinen Anwendungsbeispielen für den Mathematikunterricht über Projekte mittlerer Größenordnung wie der Entwicklung von Tools, die dem Lehrer die Erstellung von Unterrichtsmaterialien (beispielsweise zur linearen Algebra) erleichtern sollen, bis hin zu sehr komplexen Vorhaben an dem Beispiel einer flexiblen tutoriellen Lernsoftware zur Algebra der Sekundarstufe I reichen.

Friedrich Schwarz (Paderborn): Geometrie und Computer

In diesem Vortrag wird über eine *MuPAD*-Library zur ebenen Elementargeometrie berichtet. Diese Library entstand als Begleitmaterial zu einer Vorlesung “Geometrie und Computer” für Studenten des Lehramts der Sekundarstufe I und wurde von Frau Anja Klein im Rahmen einer schriftlichen Hausarbeit erweitert.
Die Library ist modular aufgebaut: Die einzelnen fundamentalen Objekte der ebenen Geometrie (wie etwa Punkt, Gerade, Kreis, Kurve) werden jeweils in einem Domain erklärt; jedes solche Domain enthält mindestens die Methoden, die die Definition, die Ausgabe und die graphische Darstellung ermöglichen. Die eigentliche Library “Geometrie” enthält die wichtigsten geometrischen Konstruktionen, wie zum Beispiel die von Parallelen, Mittelsenkrechten, Winkelhalbierenden. Damit kann man einerseits zu geometrischen Objekten, die durch ihre Koordinaten gegeben sind, neue Objekte konstruieren, wie etwa zu einem Dreieck dessen Inkreismittelpunkt als

Schnittpunkt seiner Winkelhalbierenden, oder man kann geometrische Sätze “automatisch” beweisen (und nicht nur an konkreten Beispielen illustrieren), wie zum Beispiel den Satz, daß der Feuerbach-Kreis eines Dreiecks die bekannte 9-Punkte-Eigenschaft besitzt. (Die Möglichkeiten zum automatischen Beweisen elementargeometrischer Sätze werden noch ausgebaut werden.) Die Library “Geometrie” kann vom Benutzer erweitert werden, einerseits durch Hinzunahme weiterer geometrischer Objekte (Vielecke, Kegelschnitte) und andererseits durch Ergänzung der geometrischen Grundkonstruktionen.

Gerald Siek (Paderborn): Multimedia in den Wissenschaften

Multimedia ist in aller Munde. CDs mit bunten Bildern sind im Supermarkt für 10 Mark zu haben und jede Firma oder Institution die etwas auf sich hält, ist mittlerweile mit eigenen Webseiten im Internet vertreten. Der Vortrag Multimedia in den Wissenschaften”beleuchtet kritisch die Rolle multimedialer Dokumente beim Einsatz in Forschung und Lehre und fragt nach dem Beitrag, den Expertensysteme für die inhaltliche Verbesserung in diesem Bereich leisten können.

Stefan Wehmeier (Paderborn): Domains

MuPAD is one of the two major CA-systems with native object- oriented capabilities. Domains - arbitrary definable data types with methods and properties - provide the key functionality for object orientation. This lecture introduces the domain concept of *MuPAD* and gives a short overview about internal issues and possible pitfalls.

Ralf Hillebrand (Paderborn): Usage of Development Tools

Every programmer knows that most of the development time of software packages is needed for finding bugs. This can be done the hard way ”by hand”by analysing the code, by inserting print commands in the code and try to find out what happens during a progam run or by the usage of an interactive debugger. It is no question that a programmer would prefer to use a debugger.
Since *MuPAD* has a builtin programming language which is used to implement the mathematical knowledge there was also a need for a debugger for its language. This debugger mdxcan be used to analyse existing mathematical algorithms by stepping through the code. It may also be used to develop own programs and library packages in *MuPAD*.
This talk will give an introduction in the usage of the *MuPAD* debugger and other programming facilities like the profiler, the userinfo mechanism, the test coverage tool (tcov) and automatical checking for undeclared variables in procedures and libraries (checkFunction and checkLib).

Paul Zimmermann (Nancy): *MuPAD* Programming Tips and Tricks

During this talk, we will present using concrete examples how to write nice and efficient programs in *MuPAD*. The main ingredients to write a nice and efficient program are to use as much as possible the functional constructs of the system (map, select, ...) together with the sequence construct (\$) and the prefix operators (_plus, _mult). Another ingredient is to use the overloading facility provided by *MuPAD*, which enables one to write short and easily readable functions. Last but not least, thinking in a generic way, i.e. not only for a special kind of input (for example rational coefficients) often helps to write nicer code.

Andreas Sorgatz (Paderborn): Dynamic Modules - Software Integration in *MuPAD*

The talk introduces the concept of dynamic modules of the computer algebra system (CAS) *MuPAD*. It is used to integrate user-defined C/C++ code as well as complete software packages into *MuPAD* and offers an alternative to interprocess communication (IPC) for many applications. The variety of different packages and algorithms which

can be utilized simultaneously from within *MuPAD* increases drastically. In this sense dynamic modules apply the principle of software integration to the broad field of mathematical applications.
¿From the user's point of view, a module is similar to a *MuPAD* library package and is used in the same way. From a technical point of view, a dynamic module is a compiled machine code library that can be linked at run-time. Modules can be unloaded at any time during the session. Special advantages of dynamic modules are:

- Efficiency: Data exchange is done without any IPC overhead.
- Open System: Users can access internal methods of the kernel and can extend it by nearly any feature desired.
- Modular System: Modules lead to a small and freely configurable run-time kernel.
- Universal Shell: Dynamic modules offer a new view on a CAS. On the one hand it works in the conventional way, providing its sophisticated capabilities of doing symbolic computation and on the other hand it can be used as a powerful and flexible universal mathematical shell to integrate, utilize or test various software packages in order to use them all together to solve mathematical problems.

Fabrice Rouillier (Nancy): Application of Modules (RealSolving/FRISCO)

RealSolving is a toolbox devoted to the study of the real roots of zero-dimensional systems of polynomials. Only exact computations are performed using recent algorithms. The new methods used in this toolbox have been mostly developed and implemented during the PoSSo project, and are actually expanded in the FRISCO project.
The first goal of this toolbox was to count the real roots of zero dimensional systems of polynomials through Hermite's method in the multivariate case and trough the Sturm-Habicht algorithm in the univariate case.
The library has been extended then in order to use this two algorithms for computing all the not empty sign conditions defined by a set of polynomials over a finite set of real algebraic numbers.
For reasons of efficiency, the methods enumerated before have been customised using special data structures (representation of the finite dimensional vector space associated to a zero dimensional system- modular technics ...) or new algorithms (reduction of quadratic forms - fast computation of the multiplication table associated to the induced finite dimensional vector space).
Many progresses have been made during the PoSSo project about the study of the real roots but one of the most significant (in a practical point of vue) is surely the Rational Univariate Representation of the roots of a zero dimensional system. It gives a way to reduce the multivariate problem in an univariate one without loosing informations about the multiplicities.
More classical algorithms are also available (Uspensky method for isolating the real roots of an univariate polynomial - sub-resultant algorithm).
Combined to Gb (Software dedicated to the Groebner basis computations - implemented by J.C. Faugere), RealSolving has been used in many applications (Steward platforms - optimisation - Real geometry).
Since either Gb or RealSolving are specialised C.A.S., it is compulsory to interface them with general C.A.S. for a comfortable use. After a summary of the functionalities we will describe the interface with *MuPAD* through its module package.

Werner M. Seiler (Karlsruhe): Differential Equations in *MuPAD*

Differential equations appear in many applications in natural sciences or engineering. They also represent a very active field within computer algebra. In this talk we will review some of the most popular approaches to both ordinary and partial differential equations. We will emphasize that concentrating exclusively on solving differential equations in closed form does not lead very far and that one must take a broader view.
We will also discuss what tools for differential equations are currently available in *MuPAD* and give an outlook on some planned extensions which will be part of the next releases of *MuPAD*. This concerns especially the development of a number of domains and categories for differential equations providing a unified basis for implementing fast and comfortably advanced algorithms.

Kalender

ICTMT '97

The Third International Conference on Technology in Mathematics Teaching,
Koblenz, September 29 – October 2, 1997
Deadline for paper submissions:
expired
Information:
`http://euler.uni-koblenz.de/ictmt3/index.html`

"33 Years of Gröbner Bases"

International Conference on Gröbner Bases,
RISC-Linz, Austria, February 2 – 4, 1998
Deadline for paper submissions:
expired
Information:
`gb-conf@risc.uni-linz.ac.at`

DIFFEQ '98

2nd International Conference on Differential Equations and Applications,
Saint-Petersburg, Russia, June 15 – 20, 1998
Deadline for paper submissions:
December 31, 1997
Information:
`diff98@osipenko.stu.neva.ru`

ICM98

International Congress of Mathematicians 1998,
Berlin, Germany, August 18 – 27, 1998
Deadline for paper submissions:

Information:
`http://elib.zib.de/ICM98`

CP97

Third International Conference on Principles and Practice of Constraint Programming,
Schloss Hagenberg, Austria,
October 29 – November 1, 1997
Deadline for paper submissions:
expired
Information:
`bcurtis@risc.uni-linz.ac.at`

International Conference on Teaching of Mathematics

International Conference on Teaching of Mathematics,
Samos, Greece, July 3 – 6, 1998
Deadline for paper submissions:
November 27, 1997
Information:
`http://icg.fas.harvard.edu/~samos98`

MNU '98

MNU-Jahrestagung 1998,
Leipzig, Germany, April 6 – 9, 1998
Deadline for paper submissions:
September 28, 1997
Information:
`http://www.informatik.uni-leipzig.de/~graebe/lfb/mnu.html`

ISSAC '98

International Symposium on Symbolic and Algebraic Computation,
Rotstock, Germany, August 13 – 15, 1998
Deadline for paper submissions:
January 20, 1998
Information:
`http://wwwteo.informatik.uni-rostock.de/ISSAC98/main.html`

MEGA '98

The Fifth International Symposium on Effective Methods in Algebraic Geometry,
Université de Rennes, Saint Malo, France, June 22 – 27, 1998
Deadline for paper submissions:
February 16, 1998
Information:
`mega98@univ-rennes1.fr`

Surfin' the Web

Interessante, nützliche und kuriose Plätze im „global village“.

Für die Tips in dieser Ausgabe danken wir Andreas Sorgatz[1] (so) und Winfried Fakler[2] (wf).

Computer Algebra

`http://lagrange.uni-paderborn.de/~ziegler/mupad/`
Das interaktive Tutorium *Computeralgebra in der Physik* zeigt die Bedeutung der Computeralgebra für die Physik auf und demonstriert dies anhand von konkreten Beispielen und Anwendungen, mit denen StudentInnen der Physik auf einfache und interessante Weise an die Computeralgebra herangeführt werden. (so)

`http://www.tu-harburg.de/rzt/rzt/ts/desktop/start.htm`
Tom Schramm's Desktop Evaluation Area enthält die Begleittexte und ca. 200 `mws-maple-v4` Beispieldateien der nächsten CDROM Ausgabe DeskTop Mathematik des Taschenbuch Mathematischer Formeln Und Moderner Verfahren von Prof. H. Stoecker. Das Buch enthält Oberstufen- bis Hochschulmathematik für Ingenieure und Wissenschaftler. Anhand der Beispiele kann der Benutzer die mathematischen Methoden des Buches leicht in eine praktische Anwendung seines Problems umsetzen. (so)

`http://www.mathematik.uni-kl.de/~zca/Singular/`
A Computer Algebra System for Polynomial Computations (so)

`http://SymbolicNet.mcs.kent.edu/areas/protocols/mp.html`
The Multi Protocol: a protocol for the efficient exchange of mathematical data. (so)

`http://www.mupad.de/R140/`
About MuPAD Release 1.4.0 (so)

`www.mupad.de/PROJECTS/ode.html`
MuPAD and ODE's. Please note that a new mailing list `mupad-ode@loria.fr` is available.
Feel free to contact us! (wf)

Mathematik

`www.emis.de`
Eine Mathe-Datenbank, auf der man Verschollenes suchen kann. (wf)

`math-www.uni-paderborn.de/HTML/Mathe/`
Mathematische Software & WWW-Server (fp)

`elib.zib.de/IMU`
The International Mathematical Union (IMU) (fp)

[1] `andi@uni-paderborn.de`
[2] `fakler@ira.uka.de`