mathPAD

Vol 3 / No 2 / November 1993

In diesem Heft

Redaktion:

Benno Fuchssteiner
Klaus Gottheil
Universität-GH Paderborn
33095 Paderborn
e-mail:
`benno@uni-paderborn.de`
`kg@uni-paderborn.de`

ISSN 0941 – 9187
V.i.S.d.P.: B. Fuchssteiner

Liebe mathPAD Leserin, lieber mathPAD Leser,

der bisherige Mitherausgeber der mathPAD, Waldemar Wiwianka, kam am 18. Juli 1993 bei einem Verkehrsunfall in Norwegen ums Leben.

Waldemar war der Projektmanager des Computeralgebra-Projektes MuPAD, das System würde ohne ihn und seine entschlossene Einsatzbereitschaft in seiner heutigen Form nicht existieren.

Viele der fortschrittlichen MuPAD-Eigenschaften sind seiner Weitsicht und seiner Begeisterung für die Schaffung eines modernen parallelverarbeitenden Systems für den wissenschaftlichen und akademischen Bereich zu danken.

Obwohl die Entwicklung von MuPAD mit aller Energie weiter vorangetrieben wird, fehlt uns Waldemar schmerzlich bei der Systemarbeit. Waldemar Wiwianka war bei seinem Tode gerade etwas mehr als dreißig Jahre alt, trotzdem war er im wissenschaftlichen Bereich weithin bekannt. Er bereicherte die Arbeit der verschiedensten Projektgruppen und Ausschüsse, er war Präsident der Sun Users Group Deutschland.

In Paderborn hat er sich bei der Planung und Einrichtung des hervorragenden Rechnerlabors der Mathematik bleibende Verdienste erworben; es wird aller Anstrengungen bedürfen, den Vorsprung, den er uns verschafft hat, nicht zu verspielen.

Waldemar Wiwianka wurde am 8. August 1961 in der Stadt Ortelsburg im früheren Ostpreußen geboren. 1966 ließ sich seine Familie in Westfalen nieder. Seit 1980 war er Student an unserer Hochschule, wo er sein Studium in Mathematik und Informatik im Jahr 1985 abschloß. Waldemar war ein Paradebeispiel für die positiven Reformaspekte unserer Hochschule: er kam zu uns mit dem Fachoberschulabschluß, seine Studiendauer lag weit unter dem Durchschnitt.

Waldemar Wiwianka promovierte 1990, sowohl seine Diplomarbeit wie auch seine Dissertation wurden unter meiner Betreuung geschrieben. In seiner Dissertation entwickelte er effiziente Algorithmen zum Test der vollständigen Integrabilität einer großen Klasse nichtlinearer partieller Differentialgleichungen. Diese Algorithmen waren die Basis einer Vielzahl von dynamischen Systemen, die in der Folgezeit bei uns untersucht wurden.

Nach seiner Promotion erhielt er eine Einladung zu einem einjährigen Forschungsaufenthalt in dem IBM Entwicklungslabor in Yorktown Heights. Leider konnte er dort nur ein halbes Jahr arbeiten, da er in Paderborn dringend gebraucht wurde. Aber auch von diesem verkürzten Aufenthalt konnten wir für den Entwurf und die Entwicklung des MuPAD Systems großen Nutzen ziehen.

Ich habe Waldemar Wiwianka lange gekannt — aber doch viel zu kurz. Er war unser Student und hat uns dann durch seine Tatkraft erlaubt, neue technische Herausforderungen nicht nur zu sehen, sondern auch anzunehmen. Für Studenten wie Waldemar hat es sich gelohnt, Lehrer zu werden. Ich kannte ihn seit seinem ersten Semester, danach als Mitarbeiter und Kollegen, und später als Freund.

Für mich und andere war Waldemar Wiwianka mehr als nur eine Bereicherung unserer täglichen Arbeit. Eine Bereicherung aber schon deshalb, weil er die Distanz des Alters vergessen ließ, und auf uns alle etwas von seinem jugendlichen Schwung und seiner ansteckenden Unbekümmertheit übertrug.

Wir waren Lehrer und Schüler; er hat bei mir Mathematik gelernt, ich bei ihm vieles andere: daß man Autorität haben kann ohne autoritär zu sein, daß man leiten und führen kann, ohne den Chef herauszukehren, daß man zu jedermann freundlich sein kann, ohne Positionen und Prinzipien preiszugeben, daß man mit Engagement und Arbeitskraft das erreichen kann, was vorher unmöglich schien, und daß man Perspektiven nicht nur sehen soll, sondern auch umsetzen kann.

Was man mit Waldemar Wiwianka zusammen anfaßte, das gelang, das wurde ein Erfolg. Er war einer der tatkräftigsten Menschen, die mir begegnet sind, der seine Arbeitskraft, wenn es um ein Ziel ging, welches ihn gefangen nahm, rücksichtslos einsetzte, und der seine eigene, große Tatkraft noch dadurch vervielfältigen konnte, daß er andere mitriß.

Waldemar Wiwianka hätte, wenn er eines der vielfach verlockenden Angebote aus der Industrie angenommen hätte, eine steile Karriere vor sich gehabt, die ihn sicher bald in die Vorstandsetagen der deutschen Industrie geführt hätte. Ich habe mich deshalb oft gefragt, was es war, was ihn bei uns gehalten hat, ob er wegen seiner Energie und Tatkraft nicht zu schade für die Hochschule sei, für die Karriere in einer Institution, die ja häufig gerade den anpackenden und tatkräftigen Charakteren eine schmerzhafte Verengung ihrer Perspektiven aufnötigt.

Er war Hochschullehrer, aber zur Lehre im herkömmlichen Sinne ist ihm keine Zeit geblieben, er ist ja nur 31 Jahre alt geworden — ein Alter in welchem eine nicht geringe Zahl unserer Studenten gerade ihr Diplom macht. Hinzu kam, daß die Aufgaben, die er wahrnahm, für den Moment zu wichtig waren, und ihm deshalb keine Zeit für die Lehre blieb. Oft habe ich mich aber kritisch gefragt, ob er ein guter oder ein nicht so guter Lehrer werden würde — ich war mir da nicht so sicher wie sonst, wenn es um ihn ging. Die Antwort auf diese Frage ist: er war einfach anders als viele von uns; die häufig kühle Art, die mitunter notwendig distanzierte Art des Dozierens hätte ihm sicher nicht so gelegen. Aber wenn man „Lehren“ im ursprünglichen Sinne betrachtet: die anpackende Art des Lehrens, das Lehren durch Vorbild, das Vermitteln von Motivation und Engagement, das Lehren nicht durch Abstraktion von Sachwissen, sondern durch dessen Anwendung, dann war er ein begnadeter Lehrer, dann war er der Typ des Lehrers, der uns häufig fehlt, dessen Fehlen ein oft empfundener schmerzlicher Mangel unserer Hochschulen ist. Jeder weiß das, der gesehen hat, wie er in seinem Verantwortungsbereich in kurzer Zeit eine Mannschaft geformt hat, die in diesem Bereich zum Besten gehörte, was wir zu bieten haben, sowohl technisch wie organisatorisch.

Trotz aller Arbeitskraft und allen Engagements war Waldemar Wiwianka kein verschlossener arbeitswütiger Mensch — ganz im Gegenteil. Man mußte ihn einfach so richtig gerne haben. Gut gelaunt, optimistisch war er, ein Mensch, der voller freudiger Erwartung ins Leben blickte, jemand der mit gegensätzlichen und diametralen Charakteren gleichermaßen gut zurecht kam. Jeder, der von uns einmal finster, verärgert und brütend auf dem Gang gestanden hat, konnte nicht umhin, sein Kommen frühzeitig zu bemerken, und spätestens, wenn er vor einem stand, hatte er einem ein — zumindest vorübergehendes — Lächeln aufgezwungen. Ich habe von allen Kollegen am meisten von ihm gehabt: kaum ein Tag verging, an dem wir nicht in einer Kaffepause — oder beim Mittagessen — die anstehenden Probleme besprechen konnten. Häufig habe ich ihm dabei mein Herz ausschütten dürfen.

Noch immer kann ich nicht ohne Trauer am kleinen Büro am Anfang des Ganges vorbeigehen, früher lenkten mich meine ersten Schritte dorthin, noch bevor ich zu meinem eigenen Büro ging.

Der Respekt vor dem Toten verlangt, daß wir die Lücke zu schließen versuchen, zu schließen in dem Sinne, daß wir dafür sorgen, daß die Spuren, die er hinterläßt, nicht in Trauer verweht werden, sondern daß sie kräftige Eindrücke hinterlassen, daß wir das, was ihm am Herzen gelegen hat, fortführen und zum Erfolg bringen.

Das werden wir tun, er fehlt uns dabei sehr.

Benno Fuchssteiner

Waldemar Wiwianka

„It was my favorite Problem!“

Fermat-Bezwinger Andrew Wiles über die Mühe, die Lösung und die Zukunft. Jan Felix Fuchssteiner sprach mit dem britischen Mathematiker am 2. Juli in Princeton — eine Woche nachdem er die Lösung des großen Fermat'schen Satzes in einem Vortrag in Cambridge vorgestellt hatte. Das Interview erschien in der Wochenzeitung „Die Zeit“.

J.F.F: When did you become interested in Fermat's last Theorem?

Wiles: From the age of ten I became interested in Fermat's last Theorem and tried to solve it throughout my teenage years. Of course I did not succeed... But I tried until I was a student. Then I stopped.

J.F.F: Because you became discouraged?

Wiles: No, because I discovered that so many other people had tried those kinds of methods I was using. I also realized that, once becoming a professional mathematician, I should try to solve other problems that would teach me something, even if I didn't succeed. At that time, Fermat's last Theorem seemed simply out of reach.

J.F.F: Why is Fermat's last Theorem an interesting problem? Because it looks so simple?

Wiles: I think it is partly because it is so easy to state, but more important: it is the history of Fermat. He left behind a number of problems, some took a hundred, some 150 years to solve. And quite a few of them generated useful mathematics. And this was the last one...

J.F.F: When and how did you approach the problem again?

Wiles: What happened is, that about eight years ago, the German mathematician Gerhard Frey introduced an astonishingly simple idea by which you could link Fermat's last Theorem to another conjecture, called the Taniyama conjecture. He couldn't reduce the Fermat problem to the Taniyama conjecture, he reduced it to the Taniyama conjecture plus another conjecture, which became known as the Epsilon conjecture. The Taniyama conjecture is something which at the time everybody believed, the Epsilon conjecture wasn't so clear. But within a year it was proven, and it became clear that Fermat's last Theorem was true because we believed the Taniyama conjecture, and secondly, that Fermat's last Theorem is going to be proved. Mathematicians could ignore Fermat's last Theorem and get on with their business for another hundred years, but the Taniyama conjecture could not be ignored. It was too central, so many problems were linked to it.

J.F.F: So you proved the Taniyama conjecture to prove Fermat's last Theorem?

Wiles: I didn't prove it in complete generality, but I proved parts of it that were connected to Fermat's last Theorem.

J.F.F: What are those elliptic curves?

Wiles: The Taniyama conjecture is basically about elliptic curves. From an algebraic point of view, you can just think of them as equations in the form $y^2 = x^3 + ax + b$, with one additional property, that the cubic equation should have distinct roots. Fermat's last Theorem is an elliptic curve, even though it looks different.

J.F.F: And you really worked on that for seven years?

Wiles: Yes.

J.F.F: Did you have any serious competition, especially in the near past?

Wiles: I do not know, not that I am aware of. I think that even though the link Taniyama conjecture — Fermat's last Theorem was made, many people thought the proof was out of reach.

J.F.F: If you worked on the proof for seven years, how did you sustain your reputation as a Princeton Mathematics professor?

Wiles: Let's say, with some difficulty. I do not know how people reacted. Some people might have thought that I wasn't working too hard.

J.F.F: But they didn't know that you were working on Fermat's last Theorem?

Wiles: Until a few months ago, nobody in my department knew what I was working on.

J.F.F: How did your family react, that you spent so much time on mathematics?

Wiles: I think they accepted it. I put my mathematics and family first and basically cut other things out of my life. I made a conscious effort, especially because I have a young family, to confine my thinking to specific times of day. If I had really let myself go twelve hours a day, it would have been really hard on the family. I knew it would be a very long process, I could not have kept up working twelve hours a day. But I never really let the problem go. I can be walking along, talking about something else, but the problem always came back to me, it was always on my mind. On the other hand, I wasn't shutting myself away.

J.F.F: Did you do any other research or teaching during the seven years? Did you write any papers?

Wiles: I didn't work on other problems, I just finished some ideas I had before. I completed some papers. I don't think I wrote anything else. I didn't devote any hard thinking to other mathematics. Nearly all my teaching and the theses my graduate students wrote, were related to the Taniyama conjecture.

J.F.F: At what time were you sure that you were on the right track?

Wiles: Two years ago I made a breakthrough, which convinced me that the proof was possible.

J.F.F: Did you think of any other ways of approaching the problem?

Wiles: In a general sense no, in a specific sense I certainly tried false leads. I knew the kinds of methods I was going to try.

J.F.F: You said that the final stage was reducing the problem to a calculation. What kind of calculation was that? Was it simply plugging in numbers?

Wiles: No, this could easily be misinterpreted. It is not a calculation in a sense that you simply put on a computer. It is a calculation in a sense that I have to compute the size of a certain group. It is similar to a kind of calculation that was done in the nineteenth century, but the analytic methods that were used then did not apply in this case. I had to find an algebraic method to make it work.

J.F.F: You could not have used computers?

Wiles: I never use computers, no, when I say calculation, it is in a sense that there is no very quick idea of doing it.

J.F.F: Do you think computers should, or could, play an important role in your field? Is it just a problem that the right tools are missing?

Wiles: In my field, computers have successfully been used to test conjectures. I think this is a big psychological advantage, because it is easier to prove a conjecture that you believe is true. However I don't think they have helped generate proofs in any sense.

J.F.F: Can one use your way of approaching this problem to solve other important problems?

Wiles: Oh, yes, I think this is just the beginning. There is a big network of overlapping problems. The Taniyama conjecture was the best known, but certainly not the only one.

J.F.F: What do you expect to be the consequences of your methods?

Wiles: Well, I hope to give people a way to a whole series of problems.

J.F.F: Do you think Fermat really had the answer, or just emptied a bottle of wine before writing this sentence on the margin of his book?

Wiles: I am sure he thought he had the answer. My feeling is that since he never wrote this in any of his letters, that it seems plausible that one moment he just thought he had the proof and therefore he wrote it in his book. I think if he had found it, it would have been his style to tell people. It would surprise me if he had just kept it to himself.

J.F.F: How long is your proof, would it fit on the margin of a book?

Wiles: Certainly not. I would estimate that my manuscript would be, typed, around 250 pages. Including the work of other mathematicians since Fermat I had to build on, one cannot estimate the entire length of the proof.

J.F.F: Were people guessing as your lecture in Cambridge proceeded that you would prove Fermat's last Theorem?

Wiles: They were guessing both ways. It was clear from my first lecture that it was a possibility. I said that I had methods to prove some cases of the Taniyama conjecture, and that I would discuss others.

J.F.F: How was the reaction of the people, when you finally announced that you had a proof?

Wiles: Oh, you should ask them.

J.F.F: Were they standing up? Or were they totally astonished?

Wiles: Well, the second is possible, they were not standing up... I don't know. You would have to ask them. I think, they were quite excited. There was a lot of speculation.

J.F.F: How did you feel when you announced that you had solved the problem?

Wiles: Well, you have to understand, that for me the excitement was when I knew that I have solved the problem; telling other people that I have solved the problem is not that important. It is rewarding, but it is not the same exciting feeling as when you first solved it.

J.F.F: How long before you told other people did you have the proof?

Wiles: I think the final part was about six, seven weeks before.

J.F.F: Then you knew that you could prove it, you didn't have it in writing?

Wiles: No, that was not the problem. Basically everything was in writing, but it was just checking.

J.F.F: Were you, or maybe, are you afraid that anyone disbelieves your proof or even proves it wrong?

Wiles: When you work so long on your own, you have to realize that it is very easy to fool yourself. What I made sure before presenting the lecture, was that I made sure that the critical arguments were correct. It is very hard to imagine that the method I used did not solve the problem. But I wouldn't guarantee that every detail is correct.

J.F.F: What are you going to do now? Do you already have a next challenge, or do you say, after seven years, I am going to take a short vacation of mathematics?

Wiles: I don't think I should try anything too hard in the next while.

J.F.F: Do you expect your life to change now that you have completed your task?

Wiles: It already has changed, I do not wake up in the morning thinking how to solve the next step.

J.F.F: Are you happy that your proof has caught the public attention?

Wiles: I think that anything that draws people to mathematics is desirable.

J.F.F: Are you a bit sad that you solved the problem?

Wiles: Yes, there is a slight feeling of loss in the sense that this problem has been there for a long time. I think it drew many of us into mathematics. But it is not a sense of loss, in the personal sense, that I regret that I solved it.

J.F.F: But this problem encouraged you to do mathematics?

Wiles: Yes, but I am not saying that I would not have done mathematics without it. However it was my favorite problem as a child. It was the kind of problem to excite young people. Especially because of its history.

J.F.F: So one doesn't have to fear that now nobody is interested in mathematics, because you „took away“ Fermat's last Theorem?

Wiles: No, hopefully not. There are interesting other problems.

J.F.F: Are you planning to develop another simple looking conjecture to puzzle and confuse future mathematicians? Maybe the „Wiles last theorem“?

Wiles: Oh no. That is a nice idea, but I think there are probably some already there.

Andrew Wiles

„Something very big...“

Jan Felix Fuchssteiner

mathPAD wollte das Umfeld kennenlernen, in dem Andrew Wiles seine bahnbrechenden Forschungen zum Fermat'schen Satz machen konnte und interviewte deshalb Dr. Simon Kochen, den Chairman des Mathematics Department der Universität Princeton. Simon Kochen ist ein bekannter Logiker.

mathPAD: Did you know that Dr. Wiles was working on Fermat's last Theorem?

Kochen: I had no idea what he was working on until his first lecture, when the speculation started. He told me several months earlier that he is working on something very big. But he wouldn't tell me what it was.

mathPAD: Isn't it unusual to work on a problem for seven years without discussing it with anyone?

Kochen: It is very unusual. Despite of the general public view of mathematicians as working in an isolated ivory tower, they are a very gregarious bunch: they like to talk about their work, about other peoples work. So it is very unusual to have someone work for seven years without discussing his mathematics with other people. On the other hand, I understand very well why he did this. Certainly in the case of Fermat's last theorem if you start telling people you are working on it they think you are crazy. It is too famous a problem, very many famous mathematicians have come to grief on it. The circumstances were unusual, but it is an unusual problem.

mathPAD: Dr. Wiles used a lot of his professional time on the problem. What did his colleagues think?

Kochen: At the same time he was working on other questions in number theory. Andrew Wiles didn't all of a sudden prove this theorem. He is a very eminent number theorist. He solved, or partially solved three very well known conjectures before. He was for example getting grants from Washington all the time. In order to get this, you have to show work. One can't just build on ones past reputation.

mathPAD: Dr. Wiles told me, that the satisfying moment was when he knew, that he had proven it, not when he told other people. Isn't that unusual? Don't mathematicians have some kind of competitive spirit and want to talk about their achievements?

Kochen: Yes, they definitely do. But I think I understand very well what he is saying, too. A mathematician who has been working very hard on a problem and realizes, that he has it. Those few moments are the most enjoyable part of the whole process. You finally have the insight and realize what the problem really means. Those are the best moments.

mathPAD: How would you evaluate the importance of the proof?

Kochen: I think, its of really very top importance. It could have been that someone had found a solution to Fermat's theorem with very elementary methods. That would have been the end of the problem. But still the proof would have been a great achievement. What makes it much more remarkable and interesting, is the method of proof. Wiles proved an important part of the Taniyama conjecture, that is called the semi stable case. That was sufficient to prove Fermat's last theorem. The Taniyama conjecture is very important in number theory. It is unfortunately much harder to explain. It involves the curves called elliptic curves, and they play a central role in a good part of mathematics — in complex analysis and number theory, and all kinds of problems, in physics and applications and so on. It is also central to the arithmetical part of number theory where you are looking for rational solutions rather than complex solutions. The Taniyama conjecture says a lot about the kind of solution you can get for elliptic curves over the rationals. What shows how powerful these methods and the proof of that conjectures are, is its application to the Fermat problem. Wiles shows, that it connects to the mainstream of number theory and mathematics. To prove this thing, he used complex analysis, geometry and number theory — a lot of work from many other mathematicians. The

Taniyama conjecture says that elliptic curves over the rationals are modular. He proved an important part of that semi-stable elliptic curves over the rationals are modular. And that is enough to imply the Fermat theorem. But the implication itself is not an obvious one, it was essentially conjectured by Gerhard Frey, who showed how from a solution of the Fermat equation you can use it to form an elliptic curve which in fact would not be modular, although it would be semi-stable. And then, once you proved, that all semi stable elliptic curves are modular you know there could not have been a solution to the equation. So that is the general idea. And another mathematician was able to actually use this idea of Frey's in order to prove this fact that this particular elliptic curve is not modular. That is were Andrew Wiles comes in to prove the semi stable part of the Taniyama conjecture, and that completes the proof. This is a very big part. The point I was going to make: not only does it connect with things that are going on, but it also shows that it is not a dead end. People will still sort of refine it as far as the Fermat problem is concerned and maybe find other simpler proofs, but the Taniyama conjecture first has not been settled completely and it will also have many other applications. So I think that is what makes the achievement such a great one, not only does he prove Fermat's last theorem, but he really proves a conjecture, which is not about a single equation but a whole family of curves which play a central role in mathematics. Other important theorems will follow from these general methods.

mathPAD: Did you ever expect the problem to be solved?

Kochen: Not in my lifetime. Although I had heard of this approach, it seemed to me that the Taniyama conjecture itself was such a powerful question. I didn't think that the most of it would be solved in my lifetime.

In eigener Sache:
MuPAD erhielt Softwarepreis

Die Akademische Software Kooperation hat in diesem Jahr MuPAD als herausragende Lehrsoftware im Bereich Mathematik mit dem Deutsch-Österreichischen Hochschulsoftwarepreis 1993 ausgezeichnet. Erstmals wurden in diesem Jahr Software-Entwicklungen aus den Ländern Deutschland, Österreich, Schweiz, den Niederlanden und Dänemark zur Auszeichnung zugelassen. Die Verleihung wurde im Rahmen eines feierlichen Festakts durch den Bundesminister für Bildung und Wissenschaft am 29. September 1993 vor Vertretern aus Hochschule, Wirtschaft und Politik in Berlin vorgenommen. MuPAD wurde zu diesem Anlaß im Rahmen des CIP-Kongresses vom 29. September bis 1. Oktober den Kongreßgästen und der Fachpresse vorgeführt.

MuPAD auf der CeBIT

Zur Vorbereitung eines Exponates für eine Messe braucht man einen langen Atem. Für die Vorstellung von MuPAD auf der CeBIT '93 bedeutete das konkret: Die Planungsphase begann bereits 1992 in Paderborn und Aachen. Das MuPAD-Team wurde dabei durch UNICONSULT in Paderborn und den Messe-Organisator „Forschungsland Nordrhein-Westfalen" in Aachen tatkräftig unterstützt. Plakate, zweisprachige Handzettel und Infoblätter wollten ebenso erstellt sein wie eine CeBIT-Sondernummer der mathPAD.

UNICONSULT führte begleitend eine Mailing-Aktion durch, bei der Einladungen zur Messe an potentielle Benutzer verschickt wurden. Die Sondernummer der mathPAD beschäftigte sich hauptsächlich mit unserem Exponat: MuPAD. Als zusätzliche Kurzübersicht wurde ein zweiseitiges „Fact Sheet" mit den wichtigsten Leistungsmerkmalen des Systems erstellt.

Rechtzeitig zur Messe erschien dann auch — für uns sehr erfreulich — das über 400-seitige Benutzerhandbuch zu MuPAD im Birkhäuser Verlag (Basel); außerdem ließ sich der Verlag überreden, das zum Zeitpunkt der Messe erscheinende „Mathematics Quarterly" (Auflage 30 000) weitgehend auf MuPAD auszurichten.

Nachdem die Vorbereitungen so gut gelaufen waren, haben wir kurz vor Beginn der Messe nur noch einen kleinen Nervenkitzel als „kick" vermißt. Dieser wurde dann auch prompt von der „Forschungsland-Truppe" geliefert: Nachdem wir unsere Rechner am Montag wie verabredet nach Hannover transportiert hatten, mußten wir feststellen, daß sich der Gemeinschaftsstand noch im Rohbau befand. Konkreter formuliert: die Stand-Komponenten waren kurz zuvor vom LKW gekippt worden. Nach einigen Stunden war man immerhin so weit, daß wir unsere Ausrüstung aufstellen konnten.

Das Forschungsland-Team bewies dann eiserne Nerven und hat den Stand noch über Nacht aufgebaut. Wer den Stand tags zuvor gesehen hatte, mochte es kaum glauben... So zeigten die Stand-Verantwortlichen bereits am ersten Tag die Tränensäcke unter den Augen, die eigentlich den letzten Messe-Tagen vorbehalten sind. Trotz dieser kleinen — wohl unvermeidlichen — Schwierigkeiten lief die Messe aber am ersten Tag wie geplant an.

Mit dem täglichen Ablauf der Messe waren wir zufrieden. Hierbei sei dankbar vor allem Herr Seel von UNICONSULT genannt, der uns unter anderem stets mit Informationen über anstehenden Begehungen versorgte. Was die Unterstützung durch den Organisator der Messe, Forschungsland NRW, angeht, kann man sich allerdings Verbesserungen vorstellen. So wurde etwa das Pressegespräch kurzfristig verschoben, daher war die Beteiligung seitens der Journalisten eher dürftig. Zusätzlich hatte man vergessen, rechtzeitig Einladungen zu verteilen.

Am MuPAD-Stand konnten wir neben dem Rektor der Universität, Vertretern der Stadt Paderborn, dem Kanzler der Hochschule und dem Stadtdirektor auch Frau Ministerin Anke Brunn begrüßen. Diese Gäste waren — unserem Eindruck nach — von den Fähigkeiten und der Perfektion des Systems MuPAD beeindruckt; sie ließen sich von der Begeisterung unserer Mitarbeiter bei der Präsentation anstecken. Vor allem Frau Ministerin Anke Brunn hat sich viel Zeit genommen; es war eine Freude, MuPAD einem so interessierten Besucher vorzustellen.

Mit der Besucherbilanz waren wir zufrieden. Nicht nur die Anzahl der Besucher war groß, sondern auch das spürbar große Interesse der Fachleute und der Fachpresse war erfreulich. Viele Besucher hatten bereits unsere Informationen, die wir an der Infothek ausgelegt hatten, in Händen und wollten das System nach einer Demonstration sofort mitnehmen. Noch heute kommen Anfragen, die sich auf die Messe beziehen.

Unser Hauptanliegen war es, zu zeigen, daß aus unserer Forschungs- und Entwicklungsarbeit ein mächtiges und vielseitiges Werkzeug entstanden ist. Wir hofften dabei auf eine aufgeschlossene, aber auch kritische Öffentlichkeit. Diese Einstellung erlaubte uns viele Kontakte zu interessierten potentiellen Benutzern und Anwendern aus der Industrie zu knüpfen. Die auf der CeBIT angebahnten Industriekontakte wurden inzwischen vertieft und ausgebaut.

MuPAD war zum ersten Mal auf einer Messe vertreten, und dies mit großem Erfolg. Die CeBIT hat uns die Möglichkeit gegeben, das System einer breiten Öffentlichkeit vorzustellen und uns — last but not least — auch viel Spaß gemacht. Die positive und kritische Resonanz, die MuPAD dabei gefunden hat, hat uns dazu veranlaßt, eine Reihe von Veränderungen vorzunehmen. So war die CeBIT auch im Hinblick auf die zukünftige Entwicklung von MuPAD ein Erfolg.

Waldemar Wiwianka auf der CeBIT '93: Frau Ministerin Anke Brunn als interessierte Besucherin am MuPAD-Stand.

What's new in MuPAD 1.2

Karsten Morisse

Seit dem Erscheinen von MuPAD 1.1 hat sich viel getan. Natürlich wurden alle Fehler die uns mitgeteilt oder die wir gefunden haben, dies waren eine Menge, eliminiert. Mehr Zeit wurde jedoch für Weiter- und Neuentwicklungen aufgewendet. In diesem Kurzbeitrag möchte ich die wesentlichen Neuerungen vorstellen.

- Mit Hilfe eines neuen Typs (`DOM_DOMAIN`) kann man sich nun auf Benutzerebene neue Datentypen (sogenannte Domains) definieren. Mit deren Hilfe ist die Implementation polymorpher Algorithmen möglich: Statt einen Algorithmus für verschiedene Instanzen einer algebraischen Struktur mehrfach zu implementieren, genügt die einmalige generische Implementation zusammen mit der Definition spezieller Domain-Methoden. Beispiele für das Arbeiten mit Domains finden sich in dem Artikel von H. Naundorf in diesem mathPAD-Heft. Für den schnellen Umgang mit Domains sorgt die Tatsache, daß sie im Kern integriert sind. In der Library findet sich eine experimentelle Version eines Kategoriekonzeptes. Eine Kategorie repräsentiert eine Klasse algebraischer Strukturen, mit ihrer Hilfe können generische Algorithmen für diese Klasse definiert werden.

  Zahlreiche Funktionen innerhalb des MuPAD-Kernes sind nun überladbar, so daß beispielsweise durch Definition einer Domain-Methode _plus die übliche Operatorschreibweise für die Addition verwendet werden kann.

- Für die Behandlung von Polynomen gibt es nun einen eigenen Datentyp (`DOM_POLY`) innerhalb des Kernes. Es handelt sich hierbei um eine dünne, distributive Darstellung. Über 20 Funktion unterstützen das Arbeiten mit diesem Datentyp. Einen Überblick über diesen neuen Datentyp gibt der Artikel von Klaus Gottheil in diesem mathPAD-Heft. Auf Library-Ebene werden auch Routinen für die rekursive Darstellung von Polynomen angeboten. Ebenfalls werden in der Library experimentelle Versionen von ggT- und Faktorisierungsalgorithmen für Polynome enthalten sein.

- Zur Performance-Analyse der in der MuPAD-Sprache geschriebenen Programme ist ein Profiler enthalten, der Auskunft über Häufigkeit und Dauer der aufgerufenen Programme gibt.

- Das Grafiktool VCam wurde vollständig überarbeitet. Dies betrifft sowohl die Handhabung des Interfaces wie auch die intern verwendeten Strukturen. Neu ist, daß Objekte nun auch mehrfarbig gezeichnet werden können.

- Die Bedienung des Debuggers ist benutzerfreundlicher geworden. Haltemarken können nun mit Bedingungen versehen werden, so daß beispielsweise umfangreiche Schleifen sehr schnell und gezielt nach Fehlern durchsucht werden können.

- Die Library-Pakete sind zur Vermeidung von Namenskonflikten mit Hilfe von Domains implementiert. Sie werden erst bei Bedarf bzw. auf Anforderung nachgeladen.

- Zahlreiche Funktionen sind neu hinzugekommen (eine gesonderte Aufstellung hierüber ist im Online-Manual enthalten), einige Funktionen wurden in ihrer Funktionalität überarbeitet. Dabei haben wir uns bemüht, Aufwärtskompatibilität zu gewährleisten.

Datenstrukturen und Speicherverwaltung in MuPAD

Karsten Morisse

Einführung

Bei dem Entwurf eines Computeralgebra-Systems müssen die Datenstrukturen sehr sorgfältig ausgewählt werden. Der Kern eines solchen Systems ist fast ausschließlich damit beschäftigt, Objekte zu generieren, zu analysieren oder zu manipulieren. Die Effizienz dieser Operationen ist weitestgehend vom Aufbau der Datenstrukturen abhängig. Die Festlegung der Datenstrukturen sollte unter Berücksichtigung verschiedener Aspekte erfolgen:

- Zur effizienten Speicherung der vielen in einem Computeralgebra-System vorkommenden Objekte müssen die Daten dynamisch veränderbar sein. Aus diesem Grund ist die Frage nach den Datenstrukturen eng verquickt mit der Frage nach einer Speicherverwaltung.

- Computeralgebra besteht im wesentlichen aus Datenmanipulation, d.h. aus der Ausführung der Grundoperationen: Extraktion und Substitution von Daten, Anlegen und Löschen von Daten. Eine effiziente Ausführung dieser Operationen ist somit Voraussetzung für ein effizient arbeitendes Computeralgebra-System.

- Ein einheitlicher Aufbau der Datenstrukturen vereinfacht viele Algorithmen des Computeralgebra-Kernes. Dieser Punkt ist auch hinsichtlich der Wartbarkeit und Pflege der Algorithmen von Bedeutung.

In diesem Beitrag möchte ich einen Einblick in die Interna des MuPAD-Kernes geben und die verwendeten Datenstrukturen vorstellen. Hierbei ist zu unterscheiden zwischen den Datenstrukturen aus Benutzersicht und den Datenstrukturen aus der internen Sichtweise. Tatsächlich ist es so, daß alle dem Benutzer zugänglichen Datentypen, wie z.B. Mengen, Tabellen oder Ausdrücke, auf nur einen internen Datentyp abgebildet sind, der jedoch weitere interne Typen enthalten kann. Die internen Datentypen werden durch die Speicherverwaltung bereitgestellt.

Speicherverwaltung

Die Speicherverwaltung ist Teil von Mammut (Memory Allocation and Management Unit), welches die Schnittstelle des Kernes zur Hardware bzw. zum Betriebssystem bildet. Mammut ist eine universell einsetzbare Speicherverwaltung, die jedoch als Grundlage zur Implementation eines Computeralgebra-Systems nahezu optimal geeignet ist.
Für alle Daten wird intern eine einheitliche Datenstruktur, der S-Pointer, verwendet. Ein S-Pointer ist ein Zeiger auf einen Header. Solch ein Header ist eine (C-)Struktur, bestehend aus den Komponenten:

- `data` – Ein Zeiger auf einen Zeiger, der wieder auf einen Header zeigt. Hierdurch wird auf die Daten verwiesen, die sich hinter dem Header verbergen. Dieser Datenbereich ist intern unterteilt in einen sogenannten Mem-Teil und einen Point-Teil. Der Mem-Teil kann Daten beliebigen C-Typs speichern, der Point-Teil besteht aus einer potentiell beliebigen Anzahl von S-Pointern. Gespeichert werden Mem-Teil und Point-Teil in einem zusammenhängenden Block. Dadurch kann die übliche C-Adressarithmetik zum Durchlaufen des Datenteils verwendet werden.

- `signature` - Jedes Datum ist mit einer Signatur versehen. Es ist intern sichergestellt, daß identische Objekte dieselbe Signatur besitzen. Die Umkehrung gilt allerdings nicht. Es kann vorkommen, daß verschiedene Objekte eine gleiche Signatur besitzen.

- `size` - Beziffert die Größe des Datenbereichs, der zu dem Header gehört. Gemessen wird diese Größe in Anzahl von S-Pointern. Damit ergibt sich diese Größe als die Summe des Mem-Teils und des Point-Teils eines S-Pointers.

- `count` - Anzahl der S-Pointer im Point-Teil auf Nachfolger.

- `refs` - Anzahl der S-Pointer, die auf diesen Header zeigen. Durch Verwendung dieses Referenz-Zählers entfällt eine zur Laufzeit teure Garbage Collection. Ein Objekt wird in dem Augenblick freigegeben, in dem refs den Wert 0 erreicht.
- `status` - In dieser Struktur werden Statusinformationen gespeichert. Intern ist dies ein Bit-Feld der Größe 16. Die einzelnen Komponenten sind:
  - `type` - Spezifiziert den Speicherverwaltungstyp des Datums.
  - `system` - Ein gesetztes Bit besagt, daß die Daten im Datenbereich nicht durch die Speicherverwaltung angelegt wurden. Somit können auch Libraries verwendet werden, die ihren Speicher selbst verwalten.
  - `unmoveable` - Ein gesetztes Bit deutet die nicht-verschiebbarkeit des Speichers an.
  - `memtype` - Speichertyp des Speichers.
  - `lock` - Dient zur Blockierung des Headers im parallelen Fall.

In Abbildung 1 ist ein S-Pointer schematisch dargestellt. Alle von der Speicherverwaltung erzeug-

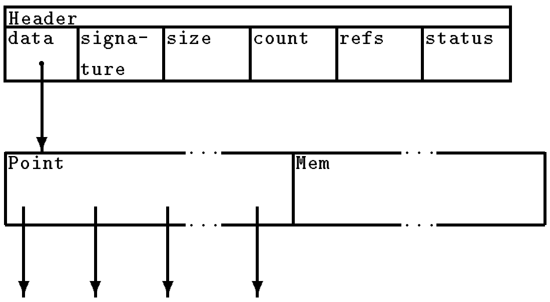


Abbildung 1: S-Pointer

ten Header besitzen, unabhängig von den dahinter verborgenen Daten, eine einheitliche Größe. Der Datenbereich eines Headers, der Point-Teil und der Mem-Teil, können beliebige Größe annehmen. Auf diese Weise können algebraische Ausdrücke, bestehend aus assoziativen Operatoren und deren Operanden, auf kanonische Weise intern durch n-äre Bäume abgebildet werden. Ein Knoten des Baumes besteht dabei aus Verweisen auf den Operator und die durch ihn verknüpften Operanden.

Obwohl die Speicherverwaltung im Prinzip nur einen Datentyp, den S-Pointer, nach aussen weitergibt, so reicht diese flexible Struktur dennoch aus, um eine Vielzahl verschiedener Datentypen zu realisieren. Dies geschieht durch eine unterschiedliche Anwendung des Point-Teiles und des Mem-Teiles im Datenbereich des S-Pointers. Die Speicherverwaltung stellt auf diese Weise zahlreiche Elementartypen zur Verfügung. Alle diese Typen sind streng typisiert, d.h. intern kann man einem Datum ansehen, welchen Typ es besitzt. Diese Information kann der type-Komponente im status-Teil eines Headers entnommen werden. Dieser Mechanismus vereinfacht die interne Realisierung des MuPAD-Kernes. Prinzipiell wird nur mit einem Datentyp, dem S-Pointer, gearbeitet. Man erspart sich die Übergabe von Zusatzinformationen an Funktionen, die die Parameter näher spezifizieren. Die Funktionen arbeiten auf S-Pointern und die tatsächlich gespeicherten Daten sowie deren Typen können dem Datum ohne großen Aufwand entnommen werden. Die Elementartypen der Speicherverwaltung sind intern mit `MMM_`<name> bezeichnet. Beispiele hierfür sind: `MMM_STRING`, `MMM_PARI` oder `MMM_CATEGORY`. Auf die Speicherverwaltungs-Typen hat der MuPAD-Anwender jedoch keinen direkten Zugriff. Ihm stehen nur die Datentypen auf Kategorieebene zur Verfügung. Diese Typen sind alle vom Speicherverwaltungs-Typ `MMM_CATEGORY`. Dieser Typ hat im Mem-Teil mindestens Platz für eine Struktur bestehend aus zwei short-Werten. Im ersten Wert, dem `cat`-Wert, ist der Kategorietyp durch eine Zahl spezifiziert, der zweite Wert `spec` hat je nach Typ eine differenzierte Bedeutung. So ist hier beispielsweise bei Zahlen und Ausdrücken das Vorzeichen untergebracht. Im Point-Teil sind die eigentlichen Daten gespeichert. Bei einigen Kategorien können hier Verweise auf Speicherverwaltungs-Typen stehen, bei anderen Kategorien stehen hier wiederum Verweise auf andere Kategorien. Somit sind also alle dem MuPAD-Anwender direkt zugänglichen Typen in zweifacher Weise typisiert. Einmal sind sie vom Speicherverwaltungs-Typ `MMM_CATEGORY`, und zum zweiten besitzen sie einen Kategorietyp. Eine vollständige Beschreibung von Mammut findet sich in [Nau92].

Kategorietypen

Die Datentypen aus Benutzersicht sind also alle vom Speicherverwaltungs-Typ `MMM_CATEGORY`. Ich möchte nun einiger dieser Kategorien vorstellen. In MuPAD wird unterschieden zwischen elementaren und zusammengesetzten Kategorietypen (ein vollständiger Überblick über die verfügbaren Kategorietypen ist in [FGK+93] zu finden).

Elementare Kategorietypen

Die elementaren Kategorietypen bestehen aus Benutzersicht aus nur einem Objekt und sind im allgemeinen nicht weiter zerlegbar. Alle numerischen Typen werden mit Hilfe des PARI-GP-Paketes (siehe [BBC+91]) behandelt. Dies betrifft sowohl die interne Repräsentation als auch die Arithmetik. Zur Speicherung dient der Speicherverwaltungstyp `MMM_PARI`. Die PARI-Datenstruktur ist zu diesem Zweck direkt im Mem-Teil eines S-Pointers eingebettet. Über den Umweg zweier zusätzlicher Dereferenzierungen kann innerhalb des Kernes somit die PARI-Langzahlarithmetik verwendet werden. Alle numerischen Datentypen sind wie in Abbildung 2 dargestellt aufgebaut, wobei das tatsächliche Da-

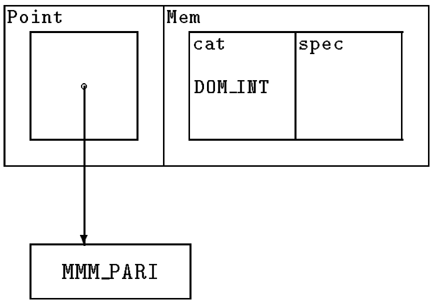


Abbildung 2: Numerische Kategorietypen

tum im PARI-Format im `MMM_PARI`-Objekt gespeichert ist und der konkrete Typ, z.B. `DOM_INT` für ganze Zahlen, in der `cat`-Komponente enthalten ist.
Zur Speicherung von Zeichenketten bis zu einer Länge von $2^{32} - 1$ Zeichen dient der Typ `DOM_STRING`. Die Speicherung des Strings er-

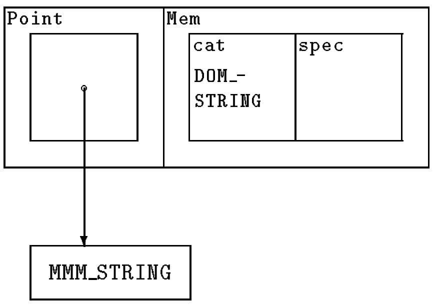


Abbildung 3: Zeichenketten

folgt als gewöhnlicher C-String im Mem-Teil des `MMM_STRING`-Objektes. Vor dem String steht als weitere Information die Länge der Zeichenkette.
Bezeichner besitzen den Typ `DOM_IDENT`. Das

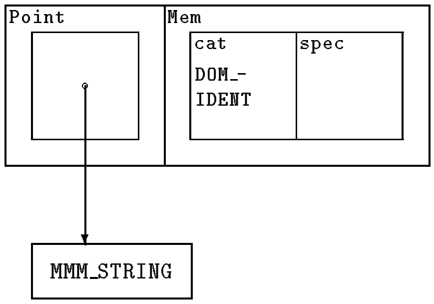


Abbildung 4: Bezeichner

Vorzeichen eines Bezeichners ist in der `spec`-Komponente gespeichert. In MuPAD gibt es intern keinen Subtraktionsoperator. Eine Subtraktion wird stattdessen dargestellt als die Addition mit dem additiven Inversen. Ein mit einem negativen Vorzeichen versehener Bezeichner a wird aus logischer Sicht wie der Ausdruck `a*(-1)` behandelt. Die Operanden von `-a` sind also `a` und `-1`, der Typ von `-a` ist die Multiplikation (`"_mult"`). Ganz analog verhält es sich mit der Division. Einen Divisionsoperator gibt es intern nicht. Dargestellt wird die Division als die Multiplikation mit dem multiplikativen Inversen. Dieses Inverse ist intern durch ein entsprechendes Vorzeichen in der `spec`-Komponente markiert. Ein mit diesem Vorzeichen versehener Bezeichner a wird aus logischer Sicht behandelt wie der Ausdruck `a^(-1)`. Die Operanden sind `a` und `-1`, der Typ ist die Exponentiation (`"_power"`). Ein weiteres Vorzeichen ist die logische Verneinung. Intern ebenfalls durch ein Bit in der `spec`-Komponente gekennzeichnet, stellt es sich nach außen durch einen Aufruf der Funktion `_not` dar. Die Speicherung des Bezeichnernamens geschieht in einem `MMM_STRING`-Objekt. Im Point-Teil des `MMM_STRING`'s hängen mehrere Verweise, die je nach Bezeichnertyp unterschiedliche Bedeutung haben. Bei Environment-Variablen sind dort Verweise auf C-Routinen untergebracht, die beispielsweise bei der Zuweisung an die Variable aufgerufen werden. Bei gewöhnlichen Bezeichnern hängt dort als nullter Sohn der Laufzeitkeller der entsprechenden Variablen.

Zusammengesetzte Kategorietypen

Mittels der zusammengesetzten Kategorietypen können mehrere Daten (u.U. unterschiedlicher Ka-

tegorietypen) zu einem Datum kombiniert werden. Bei einigen Typen darf eine beliebige Anzahl von Daten beliebiger Kategorietypen zusammengefaßt werden, bei anderen ist eine feste Anzahl von Daten bestimmter Kategorietypen erforderlich.
Die Speicherung von Mengen (DOM_SET) und Tabellen (DOM_TABLE) erfolgt in einem assozia-

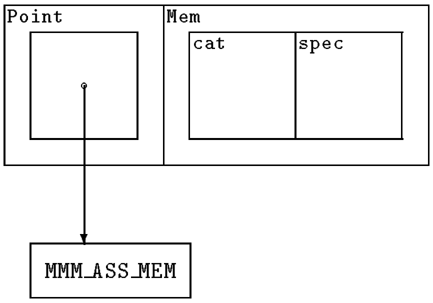


Abbildung 5: Mengen und Tabellen

tiven Speicher (interner Speicherverwaltungstyp MMM_ASS_MEM). Dieser inhaltsadressierte Speicher ist intern durch die Verwendung von Hash-Tabellen realisiert. Die zu speichernden Mengenelemente werden dabei mit einem leeren Objekt assoziiert, bei Tabellen werden die Tabelleneinträge mit dem zugehörigen Index assoziiert. Die cat-Komponente enthält wiederum den konkreten Kategorietyp.
Zur Zusammenfassung mehrerer Objekte beliebiger Kategorietypen zu einem Datum unter Beibehaltung der Ordnung dient der Typ DOM_LIST. Im Point-Teil stehen die Verweise auf die Li-

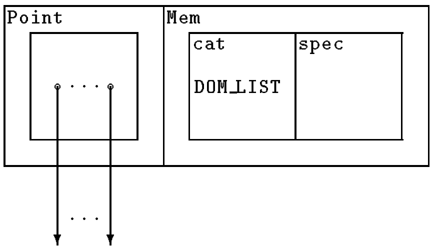


Abbildung 6: Listen

steneinträge, die einen beliebigen Kategorietyp besitzen dürfen.
Zur Speicherung von Prozeduren dient der Typ DOM_PROC. Im Point-Teil eines solchen Datums stehen 6 Verweise. Der erste ist ein MMM_CODE-Objekt. Dies ist ein Verweis auf die interne Routine MEV_eval_interpreter. Diese generische Funktion wird zur Evaluierung eines Prozeduraufrufes angesprungen. Dabei werden ihr neben den Standardargumenten innerhalb des Evaluierers der Prozeduraufruf in Form eines Ausdrucks und die Prozedurdefinition übergeben. Sie aktualisiert den Stack (formale Parameter werden mit dem Wert des entsprechenden aktuellen Parameters auf den Stack geschrieben, lokale Variable werden ohne Wert, d.h. mit MMMNULL, auf den Stack geschrieben) und evaluiert dann den Rumpf der Prozedur. Der zweite Eintrag im Point-Teil verweist auf eine Ausdruckssequenz der formalen Parameter der Prozedur. Der dritte verweist auf die Ausdruckssequenz der lokalen Variablen. Eine leere Sequenz wird dabei jeweils durch ein DOM_NIL gekennzeichnet. Der vierte Eintrag verweist auf den Prozedurrumpf. In der Regel ist dies eine Anweisungssequenz. Der fünfte Eintrag referenziert die Remember-Tafel einer Prozedur und der letzte Eintrag verweist auf den Namen, der selbst wieder ein beliebiges MuPAD-Datum sein darf. Fehlen diese, so stehen an deren Stelle jeweils ein DOM_NIL. Mit Hilfe dreier Bits (OPTIONS_CHECKED_BIT,HOLD_BIT,REMEMBER_BIT) werden die Optionen einer Prozedur in der spec-Komponente des Mem-Teiles gespeichert. Das OPTIONS_CHECKED_BIT gibt an, daß die Optionen der Prozedur bereits ausgewertet wurden. Das HOLD_BIT besagt, daß bei einem Prozeduraufruf die übergebenen Parameter nicht evaluiert werden. Schließlich sorgt das REMEMBER_BIT für das Anlegen und Unterhalten einer Remember-Tafel.

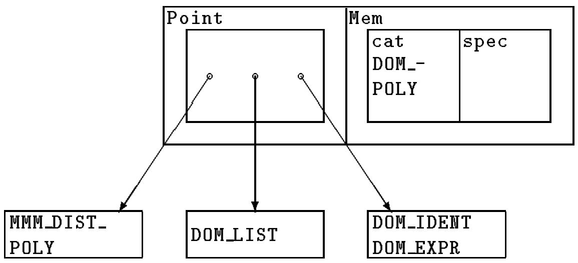


Abbildung 7: Polynome

Polynome werden intern mittels einer dünnen, distributiven Darstellung gespeichert. Das geschieht durch Verwendung des Speicherverwaltungstyps MMM_DIST_POLY. Darin sind sowohl Exponenten als auch Koeffizienten der beteiligten Terme abgelegt. Auf diese Struktur hat der Benutzer allerdings keinen direkten Zugriff. Die weiteren Einträge im Point-Teil sind eine Liste der Unbestimmten sowie der Koeffizienten-Ring des Polynoms. Die Reihenfolge in der Liste gibt die Ordnung der Unbestimmten an. Das erste Listenelement ist die Hauptvaria-

ble, die übrigen Einträge sind die Nebenvariablen. Der Koeffizienten-Ring wird durch den Bezeichner `Expr`, einen Ausdruck `IntMod(n)` oder ein Domain angegeben.
Der Typ `DOM_DOMAIN` dient zur Speicherung von Domains. Er besteht im Point-Teil aus einem Verweis

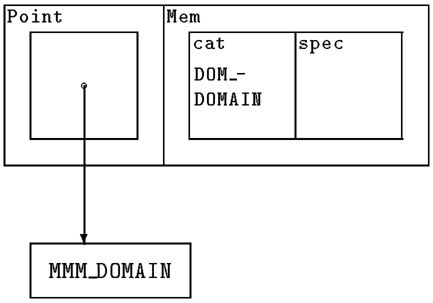


Abbildung 8: Domain

auf den Speicherverwaltungstyp `MMM_DOMAIN`, der wiederum im Point-Teil `MEVC_NUMBER_OF_QUICK_VALUES`+1 viele Einträge besitzt. Diese Größe ist eine interne Konstante, die zum Übersetzungszeitpunkt festgelegt ist. Der erste Eintrag ist eine `DOM_TABLE`, in der die Methoden und andere Eigenschaften der Domain abgelegt werden. Die übrigen Einträge dienen ebenfalls zur Speicherung von Methoden und Eigenschaften der Domain. Da ein Zugriff auf die Methoden über die Tabelle in manchen Fällen zu langsam ist, gibt es zusätzlich noch diesen direkten Zugriffsmechanismus. Hierauf hat der Benutzer jedoch keinen Einfluß, diese Technik steht nur innerhalb des Kernes zur Verfügung.
Die Elemente einer Domain werden in dem Typ `DOM_EXT` abgelegt. Der Point-Teil eines solchen Datums darf beliebig viele Einträge enthalten. Der

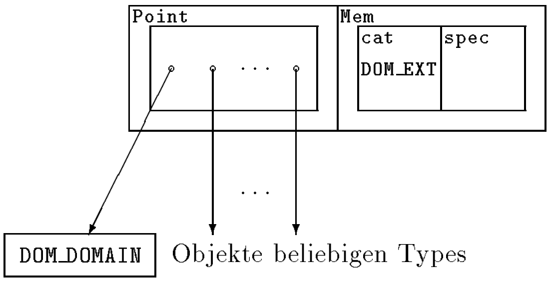


Abbildung 9: Elemente einer Domain

erste Eintrag ist ein Verweis auf das Domain, zu dem das Element gehört. Die übrigen Einträge sind Verweise auf Objekte beliebiger Typen. Hier steht es dem Anwender frei, wie er ein Domainelement repräsentieren möchte.

Der allgemeinste Kategorietyp in MuPAD ist derjenige des Ausdruckes. Alle durch Operatoren und deren Operanden gebildeten algebraischen Ausdrücke sowie alle Anweisungen der MuPAD-Programmiersprache werden in Objekten des Typs `DOM_EXPR` gespeichert. Die Speicherung erfolgt dabei durch Verweise innerhalb des Point-Teiles des `DOM_EXPR`-Objektes. Für jeden Operanden eines algebraischen Ausdrucks gibt es einen Verweis auf dieses Datum. Die bei einer Anweisung gespeicherten Teile sind die nicht aus Schlüsselwörtern bestehenden Komponenten. Damit der Evaluierer entscheiden kann, wie er sich bei einem `DOM_EXPR`-Objekt zu verhalten hat, muß ihm die genaue Art des Objektes natürlich mitgeteilt werden. Zu diesem Zweck besitzt jedes Objekt noch einen weiteren Verweis im Point-Teil. Dieser als nullter Operand bezeichnete Eintrag ist der erste Eintrag innerhalb des Point-Teiles und kann auf eine beliebiges MuPAD-Datum verweisen. Interpretiert wird dieses `DOM_EXPR`-Objekt in funktionaler Notation. Im Zuge der Evaluierung dieses Datums passiert dann folgendes: Zunächst wird der Typ des nullten Sohnes geprüft. Handelt es sich um eine Funktionsumgebung oder eine Prozedur, so wird die entsprechende Routine mit den übrigen Söhnen als Argumente ausgeführt. Im Falle eines Bezeichners ohne Wert wird ein nicht weiter evaluierbarer Funktionsaufruf zurückgegeben. Evaluiert sich der Bezeichner z.B. zu einer Funktionsumgebung oder einer Prozedur, so wird wie oben verfahren. Bei den übrigen Kategorietypen wird ein Funktionsaufruf als Aufruf einer konstanten Funktion interpretiert und der nullte Sohn wird als Ergebnis zurückgegeben. Somit ist ein durchgängiges Konzept für eine funktionale Programmiersprache realisiert. Der nullte Operand bezeichnet die Funktion, die übrigen Söhne sind die Funktionsargumente. Diese Schreibweise, Freunde der Programmiersprache LISP werden sich freuen, wird dem MuPAD-Anwender auch nach außen gereicht. Zu jedem Operator und zu jeder Anweisung gibt es ein funktionales Äquivalent in Form einer Underline-Funktion. Beispielsweise gibt es zum Additionsoperator `+` die Funktion `_plus`, zur Zuweisung `:=` existiert die Funktion `_assign`.

Die gewählte Datenstruktur für algebraische Ausdrücke kommt der mathematischen Interpretation eines solchen Ausdruckes, ein assoziativer Operator verknüpft beliebig viele Operanden, sehr nahe. Die Abspeicherung erfolgt in Form eines n-ären Bau-

mes. Die Wurzel des Baumes repräsentiert den vollständigen Ausdruck. Die Nachfolger sind der Operator und die Operanden. Letztere können Bezeichner (Blätter des Baumes) oder durch einen Operator höherer Bindungspriorität gebildete algebraische Ausdrücke (Teilbäume des Baumes) sein.
Die Datenstruktur des n-ären Baumes, insbesondere der identische Aufbau von Ausdrücken und Anweisungen, und ihre technische Realisation verhilft grundlegenden Routinen, wie z.B. dem Simplifier, zu einer hohen Verarbeitungsgeschwindigkeit. Da der Point-Teil eines jeden Datums aus einem zusammenhängenden Speicherblock besteht, können die einzelnen Verweise durch einfache C-Adressarithmetik bestimmt werden. Somit entfällt ein mühsames Hinunterhangeln zu den Operanden, welches beispielsweise im Falle einer binär orientierten Datenstruktur notwendig wäre.
Neben der hohen Verarbeitungsgeschwindigkeit hat die Verwendung einer uniformen Datenstruktur den zusätzlichen Vorteil, daß sich viele Algorithmen innerhalb des Kernes vereinfachen. Ein typisches Beispiel hierfür sind die Funktionen zur Datenmanipulation. Der i-te Operand ist in der Regel durch den Zugriff auf den i-ten Verweis innerhalb des Point-Teiles eines Datums zu erhalten. Es ist im allgemeinen nicht nötig, daß gegebene Datum näher zu untersuchen.
Durch die Verwendung einer einheitlichen Datenstruktur für algebraische Ausdrücke und Anweisungen kommt eine weitere, sehr wichtige Eigenschaft von MuPAD zutage: Die vollständige Verschmelzung dieser Objekte. Abgesehen von der natürlich unterschiedlichen Funktionalität besteht kein prinzipieller Unterschied zwischen Anweisungen und algebraischen Ausdrücken. Anweisungen in MuPAD sind algebraische Daten. Somit können sie genauso wie die Daten aller anderen Typen zur Laufzeit des Systems generiert und manipuliert werden.
Zur Veranschaulichung wollen wir noch einige Beispiele für `DOM_EXPR`-Objekte angeben. Dabei zeichnen wir die einzelnen Knoten nur schematisch. Eine genaue Unterscheidung in Mem- und Point-Teil lassen wir zugunsten der Übersichtlichkeit weg. Auch berücksichtigen wir aus gleichem Grund die eindeutige Datenhaltung nicht. Identische Objekte werden, obwohl intern nur einmal gespeichert, mehrfach dargestellt.

Literatur

[BBC+91] C. Batut, D. Bernardi, H. Cohen, M. Olivier, and N.-P. Skoruppa. The PARI-GP package. *mathPAD*, 1(3):30–32, September 1991. Sonderausgabe DMV-Tagung.

[FGK+93] B. Fuchssteiner, K. Gottheil, A. Kemper, O. Kluge, K. Morisse, H. Naundorf, G. Oevel, T. Schulze, and W. Wiwianka. *MuPAD Benutzerhandbuch*. Birkhäuser Verlag Basel Boston Berlin, 1993.

[Nau92] H. Naundorf. Mammut: Memory allocation management unit. Diplomarbeit, Universität-GH Paderborn, Juni 1992.

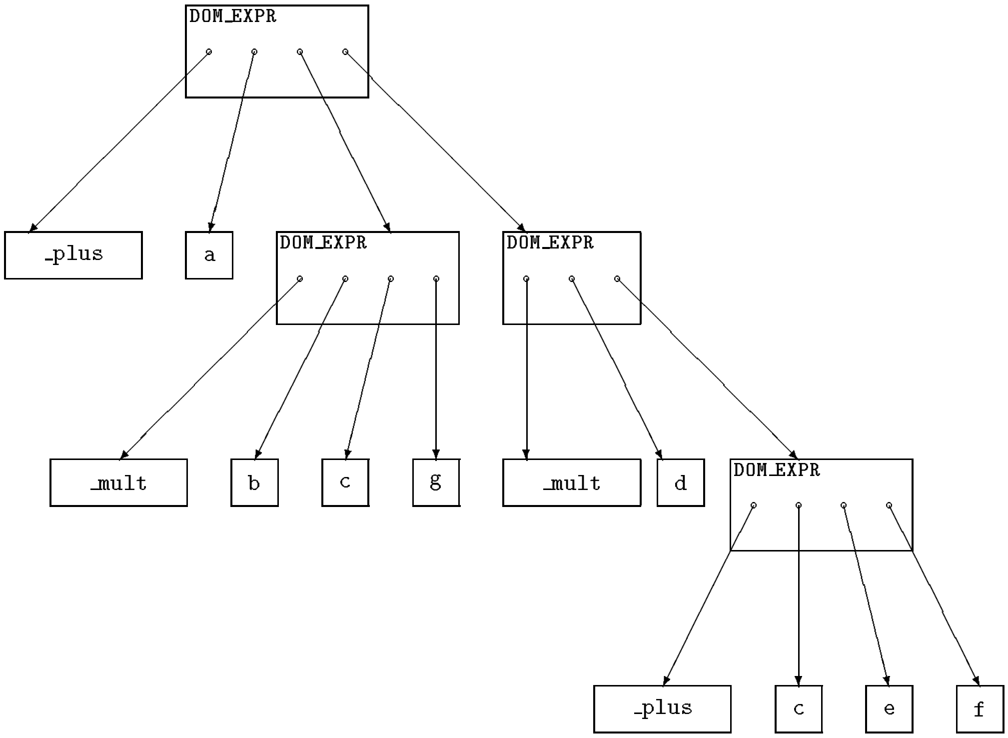


Abbildung 10: Der algebraische Ausdruck `a + b*c*g + d*(c+e+f)`

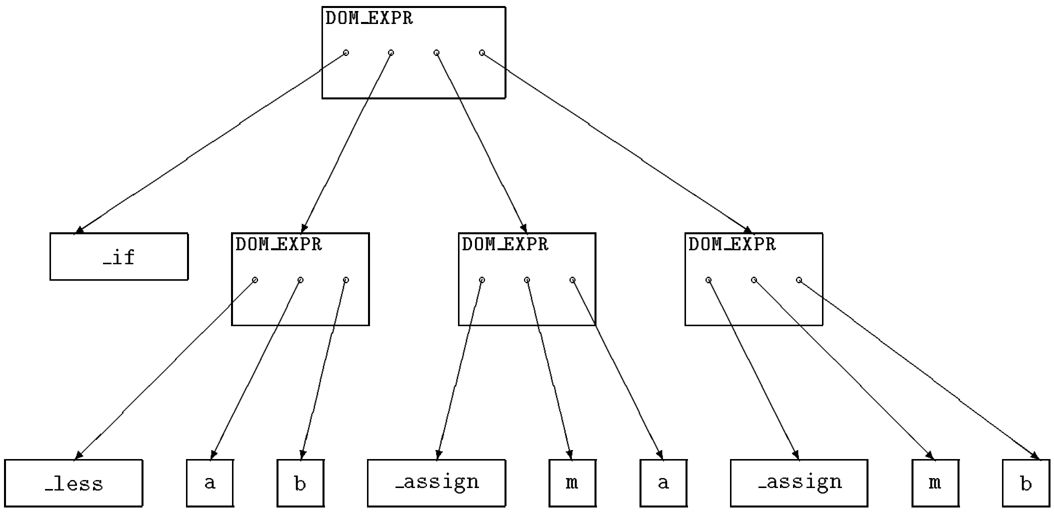


Abbildung 11: Die Anweisung `if a<b then m:= a else m := b end_if`

Polynome in MuPAD

Klaus Gottheil

Im MuPAD-Kern werden ab Version 1.2 spezielle Datenstrukturen und Algorithmen für Polynome bereitgestellt. Diese erlauben schnelle Grundoperationen auf Polynomen.

Einleitung

Die Computeralgebra-Systeme Maple und Mathematika kennen keine speziellen Datenstrukturen für Polynome. In ihnen werden „normale" Ausdrücke zur Darstellung von Polynomen verwendet. Dies hat den Vorteil, daß keine Konvertierung zwischen Polynomen und Ausdrücken notwendig ist. Außerdem ist die Benutzung der Polynom-Funktionen intuitiv, da aus Sicht des Benutzers gewisse Ausdrücke und Polynome „ein und dasselbe" sind.

Die MuPAD-Gruppe hat sich bei der Planung für die Version 1.2 hingegen entschlossen, einen speziellen Datentyp für Polynome einzuführen (`DOM_POLY`). Einer der Gründe dafür ist, daß in MuPAD die Terme von Summen vom internen Simplifizierer sortiert werden. Diese Sortierung entspricht dabei i.a. nicht der kanonischen Darstellung von Polynomen (gleich welcher Ordnung). Ein anderer Grund ist der Speicherplatz, der von Ausdrücken belegt würde. Die neue Datenstruktur ist wesentlich kompakter.

Neben dem polynomialen Ausdruck werden in einer Polynom-Datenstruktur noch die Unbestimmten und der Koeffizienten-Ring gespeichert. Diese Typ-Informationen sind bei der Programmierung von Algorithmen nützlich, sie müßten andernfalls als zusätzliche Parameter übergeben werden.

Um dem interaktiven Benutzer das Leben zu erleichtern, können bei den entsprechenden Funktionen immer auch Ausdrücke anstelle von Polynomen als Argumente angegeben werden. Diese werden dann intern in Polynome umgewandelt.

Die Benutzersicht

Polynome haben wie gesagt den Datentyp `DOM_POLY`. Ein normaler Ausdruck kann mit der Funktion `poly` in ein Polynom umgewandelt werden. Der Aufruf

```
p:= poly(x*(x + a*y) + 12, [x,y]);
```

etwa erzeugt das Polynom `p` in den Unbestimmten `x` und `y`.

Polynome werden in funktionaler Form mit Hilfe der Funktion `poly` ausgegeben, ähnlich wie bei Tabellen. Das oben definierte Polynom `p` z.B. wird als

```
poly( x^2 + a*x*y + 12, [x,y] )
```

ausgegeben.

Als Besonderheit kann man für Polynome noch den Koeffizienten-Ring angeben. Neben Polynomen über beliebigen MuPAD-Ausdrücken kann man noch Polynome über der Ring $\mathbb{Z}_n := \mathbb{Z}/n\mathbb{Z}$ für positive ganze Zahlen n sowie mit Domain-Elementen als Koeffizienten definieren.

Der Ring $\mathbb{Z}_7$ wird in MuPAD als `IntMod(7)` geschrieben („integer modulo 7"). Der „Ring der MuPAD-Ausdrücke" wird bei Polynomen mit `Expr` bezeichnet.

Die Angabe des Koeffizienten-Rings

Durch die Angabe des Koeffizienten-Rings ergibt sich eine Typisierung der Polynome. Ein häufig benötigter Ring ist der Restklassen-Ring $\mathbb{Z}_n$. Polynome über $\mathbb{Z}_n$ werden z.B. bei der Faktorisierung verwendet und Operationen auf solchen Polynomen müssen möglichst schnell sein.

Eine Möglichkeit, in $\mathbb{Z}_n$ zu rechnen, wäre nun, daß man vor der Rechnung die Operationen wie `_mult` und `_plus` umdefiniert, so daß sie über $\mathbb{Z}_n$ rechnen. Wenn die Polynom-Operationen dann die Koeffizienten normal evaluieren, so wird durch diese Änderung der Grundoperationen die Rechnung in $\mathbb{Z}_n$ durchgeführt. Dies hat aber zwei Nachteile: Erstens ist die dauernde Umdefinition der Grundoperationen und die Verwaltung der „aktuellen Operationen" mühselig, zweitens werden die Grundoperationen als MuPAD-Prozeduren recht langsam im Vergleich zu den Kern-Routinen. Daher wurden die

Grundoperationen für die möglichen Koeffizienten-Ringe im Kern implementiert.
Neben $\mathbb{Z}_n$ und beliebigen Ausdrücken kann noch ein Domain als Koeffizienten-Ring verwendet werden. Dieses Domain muß gewisse Grund-Operationen zur Verfügung stellen, damit es als Koeffizienten-Ring verwendet werden kann. (Eine Einführung in das Domain-Konzept von MuPAD wird im Artikel „Domains in MuPAD“ in dieser Ausgabe der mathPAD geboten.)
Die Verwendung eines Domains als Koeffizienten-Ring hat den großen Vorteil, daß man für die Koeffizienten eine kanonische Darstellung erhält, sofern das Domain solch eine kanonische Darstellung realisiert. Bei normalen Ausdrücken kann dies nicht immer sichergestellt werden. (MuPAD hat ja z.B. keine kanonische Darstellung rationaler Funktionen. Verwendet man jetzt MuPAD-Ausdrücke zur Darstellung rationaler Funktionen als Koeffizienten, so kann die 0-Funktion nicht unbedingt erkannt werden. Damit erhält man eventuell Polynome, deren Koeffizienten zwar ungleich 0 sind, die aber eine Darstellung der 0 im Ring der rationalen Funktionen sind. Solche Polynome sind dann nicht in Normalform.)

Polynom-Arithmetik

Polynome können negiert, addiert, multipliziert, potenziert und differenziert werden. Das Inverse eines Polynoms ist nicht definiert. Polynome können als Operatoren verwendet werden, dabei wird das Polynom als Polynomfunktion aufgefaßt.
Bei einer Summe oder einem Produkt werden zuerst alle Polynome eines Typs zusammengefaßt und dann „für sich“ bearbeitet. Es wird nicht versucht, eine Typkonvertierung durchzuführen. Der Ausdruck

```
2 * poly(x,[x])
```

z.B. bleibt so stehen, es wird nicht das Polynom `poly(2*x,[x])` zurückgeliefert. Es wäre sehr schwierig, den vom Benutzer gewünschten Typ des Ergebnisses zu „raten“. Im obigen Beispiel könnte das gewünschte Ergebnis schließlich auch der Ausdruck `2*x` sein.
Mit den Funktionen `op`, `subsop`, `subs` und `subsex` kann wie gewohnt auf Polynomen gearbeitet werden. Dabei ist aber zu beachten, daß diese Funktionen auf dem „nach außen sichtbaren“ polynomialen Ausdruck nicht besonders effizient arbeiten. Hier sollte, wenn irgend möglich, eine der speziellen Polynom-Funktionen verwendet werden.

Ein Aufruf von `op` mit einem Polynom als Argument liefert immer 3 Operanden: Einen polynomialen Ausdruck, eine Liste der Unbestimmten und eine Darstellung des Koeffizienten-Rings.
Polynome werden aus Geschwindigkeits-Gründen nie implizit evaluiert. Auch Funktionen, die als Argument ein Polynom haben und als Ergebnis einen Ausdruck liefern, evaluieren das Ergebnis nicht. Soll das Ergebnis einer solchen Funktion evaluiert werden, so muß der Aufruf in `eval` geschachtelt werden.

Polynom-Funktionen

Die mit MuPAD Version 1.2 zur Verfügung gestellten Polynom-Funktionen werden hier nur kurz aufgezählt. Näheres ist dem Benutzer-Handbuch zu entnehmen. Bei allen Funktionen kann wie gesagt anstelle eines Polynoms auch ein Ausdruck und optional eine Liste mit Unbestimmten als Argument angegeben werden. Dies mancht die interaktive Benutzung unter Umständen einfacher.

`poly` wandelt einen Ausdruck in ein Polynom um. Weiterhin kann mit `poly` der Typ eines Polynoms geändert werden.

`expr` wandelt ein Polynom in einen Ausdruck um.

`degree` liefert den totalen Grad eines Polynoms oder den Grad bezüglich einer Unbestimmten.

`degreevec` liefert eine Liste mit den Exponenten des führenden Terms.

`lcoeff` liefert den führenden Koeffizienten.

`lmonomial` liefert das führende Monom.

`lterm` liefert den führenden Term.

`nthcoeff` liefert den n-ten Koeffizienten.

`nthmonomial` liefert das n-te Monom.

`nthterm` liefert den n-ten Term.

`nterms` liefert die Anzahl der Terme eines Polynoms.

`tcoeff` liefert den Koeffizienten des letzten Terms.

`coeff` liefert eine Sequenz sämtlicher Koeffizienten oder die Koeffizienten der Terme mit einem bestimmten Faktor.

`mapcoeffs` wendet eine Funktion auf die Koeffizienten eines Polynoms an.

`multcoeffs` multipliziert sämtliche Koeffizienten eines Polynoms mit einem Faktor.

`divide` führt eine Euklidische Division zweier Polynome durch. Weiter kann ein multivariantes Polynom geteilt werden, falls eine exakte Teilung ohne Rest möglich ist.

`pdivide` führt eine Pseudo-Division zweier Polynome durch.

`content` berechnet den ganzzahligen Inhalt.

`norm` liefert die Norm eines Polynoms.

`evalp` evaluiert ein Polynom an einer Stelle.

`genpoly` erzeugt aus einer Zahl durch eine b-adische Expansion ein Polynom.

`iszero` prüft, ob ein 0-Polynom vorliegt.

Die Polynom-Datenstruktur

Polynome haben in MuPAD eine dünne, distributive Darstellung mit lexikographischer Ordnung der Monome. Dies gilt sowohl für univariante als auch für multivariante Polynome. Die Darstellung eines Polynoms der Form

$$\sum_{i=0}^{n} c_i x_1^{e_{i,1}} \cdots x_m^{e_{i,m}}$$

darf man sich denken als Liste der Form

$$(c_n, (e_{n,1}, \ldots, e_{n,m}), \ldots, c_0, (e_{0,1}, \ldots, e_{0,m})).$$

Hierbei stellt $c_i, (e_{i,1}, \ldots, e_{i,m})$ das i-te Monom des Polynoms dar. Es werden natürlich nur solche Monome gespeichert, deren Koeffizienten ungleich 0 sind. Die Reihenfolge der Monome in der Liste entspricht der lexikographischen Ordnung der Exponenten. Das Polynom

$$3x^3y^2 - 7xy^5 + x - 5y^2 + 2$$

entspricht damit der Liste

$$(3, (3, 2), -7, (1, 5), 1, (1, 0), -5, (0, 2), 2, (0, 0)).$$

Die Koeffizienten werden dabei als gewöhnliche MuPAD-Ausdrücke gespeichert. Die Exponenten werden dagegen als Maschinen-Ganzzahlen gespeichert. Die Summe der Exponenten eines Terms darf dabei $2^{31} - 1$ nicht übersteigen, dies dürfte in der Praxis aber kein Problem sein. Die Exponenten werden dicht gespeichert, d.h. auch Exponenten, die den Wert 0 haben, werden in der Exponenten-Liste eines Monoms gespeichert.

Neben den Monomen wird in der Polynom-Datenstruktur noch eine Liste der Unbestimmten $x_1, \ldots, x_m$ sowie der Koeffizienten-Ring gespeichert. Weiter wird die Position des führenden Monoms bezüglich der Grad-Ordnung gespeichert. Auf das führende Monom kann also sowohl bezüglich der lexikographischen Ordnung als auch bezüglich der Grad-Ordnung schnell zugegriffen werden.

Nach „außen“, für den Benutzer, ist die oben beschrieben Liste der Monome nicht sichtbar. Die Funktion `op` liefert als ersten Operanden eines Polynoms immer einen polynomialen Ausdruck.

Eine rekursive Darstellung

Obwohl Polynome im Prinzip distributiv dargestellt werden, ist eine rekursive Darstellung mit Hilfe von Domains leicht zu realisieren. In der MuPAD-Library ist dafür die Funktion `Poly` enthalten. Mit Hilfe von `Poly` können Domains erzeugt werden, deren Elemente gewöhnliche MuPAD-Polynome sind.

Mit Hilfe der so erzeugten Domains lassen sich Polynome definieren, deren Koeffizienten MuPAD-Polynome sind, die also eine rekursive Darstellung haben. Hier werden die Vorteile der Domains deutlich sichtbar.

Domains in MuPAD

Holger Naundorf

Domains sind der Schlüssel zum objekt-orientierten Programmieren in MuPAD. Die damit gegebenen Möglichkeiten werden kurz vorgestellt.

Einführung

Viele Benutzer von Computer-Algebra-Systemen haben früher oder später den Wunsch, sich eigene Typen zu definieren. So gibt es in MuPAD keinen Typ, der Elemente der Gruppe $\mathbb{Z}_7$ mit der Addition darstellt. Auch wenn in einem Computer-Algebra-System dieser Typ existierte, so gäbe es noch unendlich viele andere Typen, die nicht alle abgedeckt werden können. Will man nun einen Algorithmus schreiben, der auf dieser Gruppe arbeitet, so muß man selbst dafür sorgen, daß die Verknüpfung der Elemente korrekt durchgeführt wird. Das entwickelte Programm ist natürlich nur für diese eine Gruppe ausgelegt. Soll der gleiche Algorithmus auf Elemente einer anderen Gruppe angewandt werden, muß dieser neu implementiert werden. Normalerweise sind zwar nur ein paar kleine Änderungen durchzuführen, – die Verknüpfungen sind durch die richtigen zu ersetzen – aber auch dies ist fehleranfällig und kostet Zeit. In MuPAD könnte man zwar eine Funktion schreiben, die automatisch dieses Programm in das entsprechende für die andere Gruppe umwandelt, aber auch diese Funktion will erst geschrieben sein und die große Anzahl von verschiedenen Algorithmen, die dadurch entstehen und praktisch das gleiche tun, verschwendet viel Speicherplatz.

Domains

Aus diesen Gründen ist es in MuPAD möglich, eigene Datentypen – die Domains – zu definieren. Ein neues Domain kann mit Hilfe der Funktion `domain` erzeugt werden:

```
d:=domain():
```

Ein Domain ist eine Ansammlung von Daten, die mit Hilfe von Indizes und der Funktion `domattr` ausgelesen und verändert werden können. Sowohl bei den Indizes als auch bei den Daten kann es sich um beliebige MuPAD-Daten handeln. `domattr` liefert den Wert `FAIL`, wenn kein Eintrag unter dem entsprechenden Index existiert.

```
domattr(d,"name"):="Beispiel":
domattr(d,"name");
        "Beispiel"
domattr(d,"x");
        FAIL
```

Nach der bisherigen Beschreibung ähneln die Domains dem Datentyp `DOM_TABLE` sehr stark. Diese Ähnlichkeit ist nicht zufällig, denn zur Implementierung von Domains wurde auf Tabellen zurückgegriffen. Das Entscheidende bei Domains ist allerdings, daß nicht nur der Benutzer die Möglichkeit hat, Daten aus einem Domain zu lesen, sondern daß dies auch vom System MuPAD selbst gemacht wird. So liefert die Eingabe `d;` nun als Ergebnis

```
        Beispiel
```

Für die Ausgabe des Domains hat MuPAD das unter dem Index `"name"` gefundene Datum ausgegeben – das auszugebende Objekt bestimmt also selbst, wie es ausgegeben werden soll. Dies wird im allgemeinen als objekt-orientiert bezeichnet. MuPAD versucht noch in vielen anderen Situationen, Daten aus einem Domain auszulesen – dies geschieht allerdings normalerweise bei Operationen auf Daten, die das Domain als Typ tragen.
Als Indizes werden in MuPAD fast immer Strings verwendet, da diese meistens eine intuitive Bedeutung haben und sie bei der Evaluierung nicht verändert werden. Auf diese Weise spart man sich beim Zugriff das Schützen des Index mit der Funktion `hold`.
Um dem Benutzer Schreibarbeit zu ersparen, gibt es in MuPAD den Operator `::`, der eine Kurzform für `domattr` darstellt. Bei dem zweiten Argument muß es sich allerdings um einen String handeln, der als Name einer Variablen zulässig ist. Das zweite Argument wird ohne Anführungsstriche geschrieben. So entspricht die Eingabe `d::name` dem Befehl `domattr(d,"name")`.

Anders als alle anderen Datentypen in MuPAD wird ein Domain nicht kopiert, bevor es verändert wird. Dadurch wirkt sich eine Änderung eines Domains an allen Stellen aus, an denen auf dieses Domain verwiesen wird. Dies nennt man Referenzeffekt. Auf diese Weise ist es möglich, daß ein Domain ein Datum enthält, das das Domain selbst als Teildatum besitzt:

```
d1:=d:
d::self:=d:
bool(d=d::self and d=d1);
          TRUE
```

Dies ist wichtig, da Methoden – ein anderer Name für in einem Domain gespeicherte Funktionen – häufig wissen müssen, zu welchem Domain sie gehören. Da der Referenzeffekt nicht immer gewünscht ist, kann man mit der Funktion `domain` eine Kopie eines Domains erstellen.

```
d1:=domain(d):
```

Obwohl `d1` genau die gleichen Einträge hat wie `d`, repräsentieren die beiden Domains unterschiedliche Daten und werden deshalb nicht als gleich erkannt:

```
bool(d=d1);
          FALSE
```

Dies ist so implementiert, da eine Veränderung eines Domains das andere Domain nicht mehr beeinflußt:

```
d1::x:="y":
d::x;
          FAIL
```

Domain-Elemente

Das Konzept der Domains wird erst sinnvoll durch die Möglichkeit, Objekte – die Domain-Elemente – zu erzeugen, die ein Domain als Typ haben können. So wie die Zahl `1` in MuPAD den Typ `DOM_INT` – also ganze Zahl – hat, kann man zu jedem Domain Objekte erzeugen, die dieses Domain als Typ haben. Ein Domain-Element kann mit Hilfe der Funktion `new` erzeugt werden:

```
e:=new(d,5,6,7):
```

erzeugt ein Element des Domains `d`, das als Einträge die Werte 5, 6 und 7 hat – ein Domain-Element kann man sich immer als eine Art Liste vorstellen. Auf diese Einträge kann mit Hilfe der Funktion `extop` zugegriffen werden:

```
extop(e,2);
          6
```

Mit der Funktion `extsubsop` können – ähnlich der Funktion `subsop` – Einträge verändert werden. Die Funktion `extnops` dient dazu, die Anzahl der Einträge in einem Domain-Element zu bestimmen.

Überladung

Der entscheidende Punkt im Zusammenhang mit Domain-Elementen ist, daß die meisten Systemfunktionen überladbar sind. Dies bedeutet, daß eine Systemfunktion, die auf ein Domain-Element trifft, in dem Domain des Elementes mit ihrem Namen als Index nach einer Methode sucht, die die Ausführung übernimmt. So kann in einem Domain z.B. leicht festgelegt werden, wie der Sinus eines Elementes berechnet werden soll.

```
d::sin:=proc(x)
   begin
      sin(extop(x,1));
   end_proc:
sin(e);
          sin( 5 )
```

Die Funktion `sin` ruft die gefundene Methode mit allen eigenen Argumenten auf. Mit Hilfe von Domain-Elementen ist es also auch möglich, dem Sinus mehr als ein Argument zu übergeben.
Auch die Funktionen `_plus` und `_mult` – die `+` und `*` repräsentieren – können überladen werden. Allerdings kommt hier nicht der sonst übliche einfache Überladungsmechanismus zum Tragen. Stattdessen wird die in einem Domain gefundene Methode nur mit allen Argumenten aufgerufen, die in diesem Domain liegen. Aus den einzelnen Ergebnissen der Methoden wird dann ein entsprechendes Datum aufgebaut. Verknüpfungen innerhalb einer algebraischen Struktur sind damit leicht zu implementieren.

Interne Methoden

Da nicht alle Operationen auf Daten in MuPAD von Systemfunktionen durchgeführt werden, gibt es in Domains auch noch Indizes, die keiner Systemfunktion entsprechen und trotzdem wichtig für die Beeinflussung des Verhaltens auf Elementen eines Domains sind. Zu diesen gehören:

"evaluate": Bestimmt, wie ein Domain-Element evaluiert wird. Existiert kein Eintrag, so evaluiert das Element zu sich selbst.

"func_call": Bestimmt, wie sich ein Domain-Element als 0ter Operand eines Ausdruckes verhält. Mit Hilfe dieser Methode kann der Programmierer eigene Typen erzeugen, die Funktionen oder Algorithmen repräsentieren.

```
d::func_call:=proc(f,x)
   begin
      extop(f,1)*x^2+extop(f,2)*x
                    +extop(x,3)
   end_proc:
e(2);
         39
```

"negate": Bestimmt, wie das additive Inverse eines Domain-Elementes berechnet wird.

"invert": Bestimmt, wie das multiplikative Inverse eines Domain-Elementes berechnet wird.

"print": Bestimmt, wie ein Domain-Element ausgegeben wird. Existiert kein Eintrag, so wird eine Standardausgabe gewählt.

```
e;
         DOM_EXT(Beispiel,5,6,7)
d::print:=proc(x)
   begin
      [extop(x)]
   end_proc:
e;
         [5,6,7]
```

Es gibt noch einige andere intern benutzte Methoden, deren Beschreibung würde an dieser Stelle allerdings zu weit führen.

Anwendung

Soll ein Domain eine Gruppe, einen Ring oder einen Körper repräsentieren, so müssen nur die entsprechenden notwendigen Methoden definiert werden und mit dem neuen Typ kann ganz normal gerechnet werden. Schwächen sind dabei derzeit noch:

- Mit `+` und `*` können nur Elemente desselben Domains miteinander verknüpft werden. Die Multiplikation mit anderen Typen, wie z.B. ganzen Zahlen ist nicht trivial möglich.
- Da die Variablen nicht typisiert sind, ist bei der Rechnung nicht bekannt, Elemente welchen Typs sie repräsentieren – die benutzten Vereinfachungsregeln sind immer die in $\mathbb{R}$ geltenden. Dabei kann es z.B. zu Fehlern kommen, wenn die Multiplikation auf der eigenen Struktur nicht kommutativ ist.

Logische Schnittstelle

Da bei der Implementation eines Domains noch nicht klar ist, ob die gewählte Repräsentation der Domain-Elemente optimal ist, sollte streng zwischen einer logischen und einer physikalischen Sicht eines Domain-Elementes getrennt werden. Die logische Sicht eines Domains muß bei der ersten Implementation festgelegt werden, die physikalische Sicht kann sich später bei Bedarf noch ändern. Aus diesem Grund sollten folgende Grundregeln beachtet werden:

- Jedes Domain sollte eine Methode "new" besitzen, mit der neue Domain-Elemente erzeugt werden. Diese Funktion überprüft zunächst die Argumente auf Korrektheit und erzeugt dann das entsprechende Domain-Element. Der Benutzer eines Domains sollte auf keinen Fall selbst mittels `new` neue Domain-Elemente erzeugen. Auf diese Weise wird verhindert, daß nicht normalisierte oder unsinnige Domain-Elemente erzeugt werden.
- Der Benutzer von Domain-Elementen sollte nicht mit einer der Funktionen `ext...` auf den Inhalt eines Domain-Elementes zugreifen. Dazu sollten nur die normalen Funktionen `op`, `subsop`, etc. benutzt werden, die im Bedarfsfall vom Programmierer mit den entsprechenden Methoden umdefiniert werden können.

Schluß

Zusammenfassend läßt sich sagen, daß MuPAD durch die Integration von Domains objekt-orientiertes und polymorphes Programmieren ermöglicht. Dadurch kann sich die Zeit, die für die Entwicklung von Programmen benötigt wird, drastisch verringern. Außerdem wird die Codegröße stark verkleinert. Ein weiterer Vorteil ist, daß der Benutzer das System nach den eigenen Wünschen konfigurieren und auch mit nicht vordefinierten algebraischen Strukturen intuitiv rechnen kann.

Rechnen in Gruppen

Holger Naundorf

Ein Problem, mit dem Algebraiker oft konfrontiert werden, ist das Rechnen in Gruppen. In dem an der Universität Paderborn aufgetretenen Fall sind dies Lp-Gruppen, die im Zusammenhang mit der Darstellungstheorie von algebraischen Kurven auftreten.

Man möchte zum einen Elemente einer Gruppe addieren und mit ganzen Zahlen mulitiplizieren, zum anderen möchte man aber auch symbolisch rechnen können, wobei die Assoziativität und die Distributivität – also die normalen Rechenregeln von + und * – ausgenutzt werden sollen.

Auf vielen Computer-Algebra-Systemen entstehen dadurch Probleme, die nur sehr schwer gelöst werden können. Ich stelle hier das Prinzip vor, wie dies in MuPAD zu lösen ist.

Gruppe als Domain

Wie in dem Artikel „Domains in MuPAD“ in diesem Heft bereits angedeutet, ist es in MuPAD einfach, eine algebraische Struktur durch ein Domain zu repräsentieren. Die normale Addition kann für die additive Verknüpfung von Gruppenelementen – diese werden durch Domain-Elemente repräsentiert – benutzt werden. Beim symbolischen Rechnen werden weiterhin die normalen Vereinfachungsregeln benutzt. Allerdings tritt das Problem auf, daß die Multiplikation nicht versucht, Gruppenelemente mit ganzen Zahlen zu verknüpfen.

Neue Multiplikation

Bei vielen anderen Systemen wäre man an dieser Stelle am Ende seiner Möglichkeiten und müßte kapitulieren oder eine gänzlich andere Lösungsstrategie suchen. In MuPAD hat man dagegen die Möglichkeit, die Schwäche des Systems selbst zu beheben, indem man neu definiert, wie die Multiplikation durchgeführt werden soll. Um dabei nicht wieder auf das Problem zu stoßen, das A- und das D-Gesetz selbst implementieren zu müssen, greift man auf die normale Multiplikation zurück. Dazu speichert man den Inhalt der Variablen `_mult` in einer anderen Variablen zwischen und speichert in `_mult` den neuen Algorithmus für die Multiplikation. Dieser führt folgende Schritte durch:

1. Multipliziere die Argumente der Funktion mit Hilfe der normalen Multiplikation. Speicher das Ergebnis in `x`. Durch diesen Schritt werden symbolische Vereinfachungen durchgeführt und die verschiedenen Gesetze angewandt.

2. Repräsentiert `x` weder eine Addition noch eine Multiplikation, so gib `x` als Ergebnis der Funktion zurück.

3. Repräsentiert `x` eine Multiplikation, so überprüfe, ob sowohl ein Gruppenelement als auch eine ganze Zahl in dem Produkt vorkommen. Ist dies der Fall, so multipliziere diese miteinander und gib das so vereinfachte Produkt zurück.

4. `x` repräsentiert eine Addition und ist durch die Anwendung des Distributivgesetzes entstanden. Untersuche jeden Summanden, ob ein Gruppenelement und eine ganze Zahl zusammenfaßbar sind. Gib das dadurch entstehende Datum zurück.

Kleine Probleme treten allerdings noch im Zusammenspiel mit der Funktion `float` auf. In der nächsten MuPAD-Version werden auch diese Probleme lösbar sein.

Schluß

Obwohl in diesem Beispiel auch noch einige Schwächen von MuPAD zu Tage treten, zeigt sich, welche Flexibilität und Mächtigkeit dieses System bietet. Der Benutzer hat fast immer die Möglichkeit, Schwächen, die in MuPAD existieren, selbst zu beheben. Dies ist insbesondere unter dem Aspekt wichtig, daß ein Programm niemals perfekt sein wird.

Axiome, Kategorien und Domains

Klaus Gottheil

Um die Konstruktion neuer algebraischer Strukturen zu erleichtern, wird mit MuPAD Version 1.2 das experimentelle Library-Paket `domains` zur Verfügung gestellt. Mit Hilfe dieses Paketes können parametrisierte Domains konstruiert werden, wobei Kenntnisse über Gemeinsamkeiten gewisser Klassen von Domains in Form generischer Algorithmen genutzt werden können.

Einleitung

Im MuPAD-Kern wird dem Benutzer ab Version 1.2 mit dem Datentyp „Domain“ ein Werkzeug zur Implementierung eigener algebraischer Strukturen an die Hand gegeben. Diese sind weitgehend in den MuPAD-Kern eingebunden und können die vorhandenen Operatoren wie `+`, `*` und `^` nutzen, wodurch die interaktive Benutzung der so definierten Strukturen sehr komfortabel möglich ist. Eine Einführung in das Domain-Konzept bietet der Artikel „Domains in MuPAD“ in dieser Ausgabe der mathPAD.

Nun ist es wenig effektiv, eine algebraische Struktur wie $\mathbb{Z}_7 := \mathbb{Z}/7\mathbb{Z}$ sozusagen „von Hand“ zu konstruieren, vielmehr möchte man in diesem Fall eine Struktur wie $\mathbb{Z}_n$ konstruieren, um daraus — abhängig vom „Parameter“ n — alle möglichen Restklassenringe über $\mathbb{Z}$ zu gewinnen. Um in der MuPAD-Sprache zu bleiben: Man möchte Domains in Abhängigkeit von Parametern erzeugen. Zur Konstruktion parametrisierter Domains werden im `domains`-Paket sogenannte *Domain-Konstruktoren* verwendet.

Oft will man ein Domain implementieren, indem man eine vorhandene Implementation übernimmt („erbt“) und nur einige Einträge in der vorhandenen Implementationen ändert. Das neue Domain wird also von einem vorhandenen Domain abgeleitet. In solch einem Fall spricht man beim ursprünglichen Domain von einem *Super-Domain* des neuen Domains, das abgeleitete Domain heißt *Sub-Domain* des ursprünglichen Domains.

Ein mit Hilfe des `domains`-Paketes definierter Domain-Konstruktor erlaubt die Angabe eines solchen Super-Domains. Das Super-Domain kann dabei natürlich wieder von den Domain-Parametern seines Sub-Domains abhängen. Ein Domain-Konstruktor muß nur diejenigen Einträge neu definieren, deren Implementation vom Super-Domain abweicht. Die mit Hilfe des `domains`-Paketes erzeugten Domains bilden also Hierarchien (ein Wald von Bäumen).

Bei der Definition neuer Domains fällt einem schnell auf, daß gewisse Domains Gemeinsamkeiten besitzen, da sie einer gemeinsamen Klasse algebraischer Strukturen angehören. So sind $\mathbb{Z}_6$ und $\mathbb{Z}_7$ Ringe, besitzen damit unter anderem die Operationen `+` und `*` und haben bezüglich dieser Operationen neutrale Elemente. In jedem Ring kann die Operation „potenzieren mit einer natürlichen Zahl“ leicht durch fortgesetztes Quadrieren implementiert werden. Nun wäre es sehr ineffektiv, wenn solch eine Operation für jeden Ring neu implementiert werden müßte. Vielmehr wünscht man sich natürlich, solch eine Operation ein für alle Male implementieren zu können und dann für sämtliche Ringe zu verwenden.

Eine Klasse algebraischer Strukturen wie „die Ringe“ wird im `domains`-Paket *Kategorie* genannt. Eine Kategorie „postuliert“ gewisse Grundoperationen und Eigenschaften derjeniger Domains, die zu ihr gehören. Die konkrete Implementation dieser Grundoperationen ist Sache der Domains. Mit Hilfe dieser Grundoperationen können dann weitere Operationen für die Kategorie definiert werden, die dabei als *generische Algorithmen* — wie das oben erwähnte fortgesetzte Quadrieren — realisiert werden.

Wird nun ein Domain neu erzeugt, so legt der zugehörige Domain-Konstruktor anhand der Parameter fest, zu welchen Kategorien das Domain gehört. Der Konstruktor trägt dann die in diesen Kategorien definierten Operationen in das Domain ein. Gehört das neu erzeugte Domain etwa zur Kategorie der Ringe, so wird die Operation „potenzieren mit einer natürlichen Zahl“ in das Domain eingetragen. Dabei hat man bei der Definition von Domain-Konstruktoren natürlich die Frei-

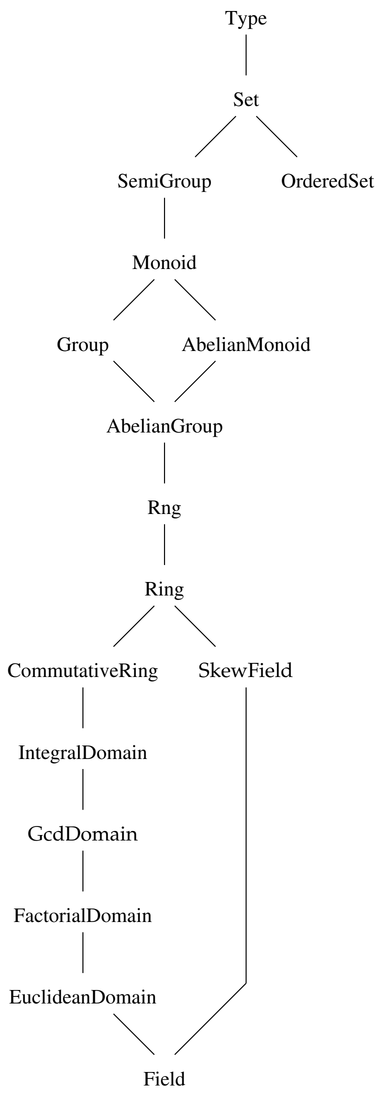


Abbildung 1: Hierarchie der Kategorien im `domains`-Paket

heit, solche Operationen auch selber zu implementieren, wenn man spezielle Eigenschaften der Domains ausnutzen möchte.

Kategorien bilden im `domains`-Paket ähnlich wie Domains Hierarchien. So gehört jedes Domain, das zur Kategorie der Schiefkörper gehört, automatisch auch zur Kategorie der Ringe. Anders ausgedrückt: Jeder Schiefkörper ist ein Ring. Kategorien brauchen nur diejenigen Eigenschaften und Operationen zu definieren, die nicht bereits von allgemeineren Kategorien zur Verfügung gestellt werden.

In diesem Zusammenhang wird die allgemeinere Kategorie der Ringe *Super-Kategorie* der Kategorie der Schiefkörper und die speziellere Kategorie der Schiefkörper *Sub-Kategorie* der Kategorie der Ringe genannt. Im Gegensatz zu den Domains kann eine Kategorie aber mehrere Super-Kategorien haben. Die abelschen Gruppen haben zum Beispiel sowohl die Gruppen als auch die abelschen Monoide als Super-Kategorien. Die Kategorien bilden also nicht nur Bäume, sondern azyklische Graphen. Die — bisher noch recht einfache — Hierarchie der im `domains`-Paket definierten Kategorien ist in Abbildung 1 dargestellt.

Kategorien können selber wieder von Parametern abhängen. So gibt es unterschiedliche Kategorien von Polynom-Ringen. Diese Kategorien hängen von der Kategorie des Koeffizienten-Rings ab. Um nicht für jede mögliche Kategorie von Koeffizienten-Ringen die zugehörige Kategorie von Polynom-Ringen neu implementieren zu müssen, gibt es im `domains`-Paket sogenannte *Kategorie-Konstruktoren*. Diese erzeugen — wie könnte es anders sein — Kategorien in Abhängigkeit von Parametern.

Die Eigenschaften von Kategorien werden mit Hilfe sogenannter *Axiome* postuliert. Das sind beim `domains`-Paket im wesentlichen Attribute, deren Existenz man erfragen kann. Die Kategorie Ring hat zum Beispiel das Axiom `unitElement("_mult")`. Dieses Axiom soll ausdrücken, daß ein Domain der Kategorie Ring ein neutrales Element bezüglich der Multiplikation besitzt. Die Axiome können von den Kategorie- und Domain-Konstruktoren abgefragt werden, um abhängig davon z.B. eine Operation zu definieren.

Auch Axiome können — wie das obige Beispiel zeigt — von Parametern abhängen. Daher gibt es im `domains`-Paket *Axiom-Konstruktoren*, die Axiome in Abhängigkeit von Parametern erzeugen.

Kategorien in MuPAD

Die oben beschriebenen Objekte — von Kategorien bis zu Axiom-Konstruktoren — werden im `domains`-Paket als Elemente spezieller Domains realisiert. Kategorien und Axiome sind Elemente der Domains `Category` und `Axiom`. Die Konstruktoren für Domains, Kategorien und Axiome sind Elemente der Domains `DomainConstructor`, `CategoryConstructor` und `AxiomConstructor`.

Soll ein neuer Konstruktor definiert werden, so wird dazu die Methode `"new"` der entsprechenden „Meta-Domain“ verwendet. Instanzen dieser „Meta-Domains“ — also Domains, Kategorien und Axiome — werden in Form eines Funktionsaufrufes mit der Methode `"func_call"` erzeugt.

Domains konstruieren

Domains werden im `domains`-Paket wie gesagt mit Domain-Konstruktoren erzeugt. Ist `Zmod` ein Domain-Konstruktor für $\mathbb{Z}_n$, so wird mit dem Ausdruck `Zmod(7)` das Domain $\mathbb{Z}_7$ erzeugt. Ein Domain-Konstruktor wird wie folgt definiert:

`DomainConstructor::new(`
name,
parameter,
⟨ *parameter-test*, ⟩
super-domain,
kategorien,
axiome,
einträge...)

Die einzelnen Argumente haben dabei folgende Bedeutung:

name gibt den Namen des Domain-Konstruktors an. Dieser wird nur für die Ausgabe verwendet.

parameter gibt die formalen Parameter des Konstruktors an. Dies muß eine Liste von Bezeichnern sein. Die Liste darf auch leer sein.

parameter-test gibt einen Ausdruck an, der zum Test der aktuellen Parameter dient. Sind keine Parameter vorhanden, so fehlt dieser Ausdruck.

super-domain gibt das Super-Domain an, von dem dieses Domain seine Implementation erbt. Hier muß ein Ausdruck angegeben werden, der sich zu einem Domain oder zu `NIL` evaluiert. Bei `NIL` hat das Domain kein Super-Domain.

kategorien gibt die Kategorien an, in denen die mit diesem Konstruktor erzeugten Domains enthalten sind. Hier muß ein Ausdruck angegeben werden, der sich zu einer Liste von Kategorien evaluiert.

Es müssen nur die jeweils „kleinsten" Kategorien angegeben werden. Gehört ein Domain zur Kategorie Schiefkörper, so gehört es damit automatisch auch zur Kategorie Ring.

axiome gibt Axiome für die Domains an. Dies muß wie bei den Kategorien ein Ausdruck sein, der sich zu einer Liste von Axiomen evaluiert. Es müssen hier nur solche Axiome angegeben werden, die nicht bereits von den Kategorien eingetragen werden.

einträge schließlich ist eine beliebig lange Sequenz von Gleichungen der Form *name*=*eintrag*. Hiermit werden die Einträge in die Domains definiert. Unter dem Index *name* wird *eintrag* in die Domains eingetragen.

Die Argumente werden bei der Definition eines Domain-Konstruktors nicht evaluiert. Die Evaluierung der Argumente geschieht vielmehr erst dann, wenn ein Domain vom Konstruktor erzeugt wird: Die Ausdrücke zum Test der Parameter sowie zur Definition der Kategorien und Axiome können die Bezeichner aus *parameter* enthalten. Bei der Erzeugung eines Domains werden diese „formalen" Parameter durch die aktuellen Parameter des Domains ersetzt. Erst danach werden die so erzeugten Ausdrücke evaluiert. Die Einträge können neben den formalen Parametern auch noch den speziellen Bezeichner `this` enthalten. Dieser wird bei der Erzeugung eines Domains durch das neue Domain ersetzt.
Ein Domain-Konstruktor für $\mathbb{Z}_n$ könnte damit etwa wie folgt aussehen:

```
Zmod:= DomainConstructor::new(
# Name #
    Zmod,
# Parameter: Modulus #
    [ Mod ],
# Parameter-Test #
    (if domtype(Mod) <> DOM_INT then
         error("no integer modulus")
     end_if;
     if Mod < 2 then
         error("modulus must be > 1")
     end_if),
# Super-Domain: nicht vorhanden #
    NIL,
# Kategorien #
    [ (if isprime(Mod) then
           Field
       else
           CommutativeRing
       end_if) ],
# Axiome: keine #
    [ ],
# Eintraege #
    "new" = proc(x) begin
        new(this, x mod Mod)
    end_proc,
    "zero" = this::new(0),
    "one" = this::new(1),
    ...
):
```

Der Konstruktor heißt `Zmod` und hat als Parameter den Modulus `Mod`. Ist der Modulus n eine Primzahl, so ist das Domain $\mathbb{Z}_n$ ein Körper, andernfalls nur ein kommutativer Ring. `Zmod` hat kein Super-Domain und keine eigenen Axiome.
Die beiden Bezeichner `this` und `Mod` werden bei der Konstruktion eines neuen Domains durch die aktuellen Werte (das Domain und den Modulus) ersetzt. Die Operation `"new"` hat in `Zmod(7)` also die Form:

```
proc(x) begin
    new(Zmod(7), x mod 7)
end_proc
```

Die Darstellung der Domain-Elemente wird implizit durch die Implementation der Operationen festgelegt. Sie kann durchaus von den Domain-Parametern abhängen. Im Falle von `Zmod` haben alle Domain-Elemente die gleiche einfache Form: Sie haben genau einen Eintrag, eine ganze Zahl aus der entsprechenden Restklasse.
Die Struktur eines Algorithmus kann übrigens von der Kategorie des jeweiligen Domains abhängen. Dies betrifft bei `Zmod` etwa die Operation `"invert"`, die das multiplikative Inverse einer Restklasse berechnet. Ist der Modulus eine Primzahl, das Domain also ein Körper, so kann das modulare Inverse immer berechnet werden. Ist der Modulus dagegen keine Primzahl, so hängt die Existenz des Inversen von der jeweiligen Restklasse ab. Das muß sich in der Operation `"invert"` des jeweiligen Domains wiederspiegeln:

```
"invert" = (
if isprime(Mod) then
    proc(x) begin
        this::new(1/extop(x,1))
    end_proc
else
    proc(x) begin
        if igcd(extop(x,1), Mod) = 1 then
            this::new(1/extop(x,1))
        else
            FAIL
        end_if
    end_proc
end_if)
```

Man könnte natürlich auch zur Laufzeit innerhalb der Operation `"invert"` abfragen, ob das Domain-Element aus einem Körper ist und entsprechend reagieren. Wie dies genau erfolgt, ist letztlich eine Entwurfsentscheidung bei der Implementation des Domain-Konstruktors. Die erste Alternative hat aber den Vorteil, daß zur Laufzeit keine Abfrage erfolgen muß und ist daher effizienter.
Die Einträge eines Domains werden wie folgt erzeugt: Zuerst wird die Definition des Eintrags im Domain-Konstruktor gesucht. Ist sie dort nicht vorhanden, so wird sie per Tiefensuche im Super-Domain und in dessen Super-Super-Domains gesucht. Wird der Eintrag in keinem der Super-Domains definiert, so wird die Definition der Reihe nach in den einzelnen Kategorien des Domains gesucht. Die Kategorien wiederum suchen per Tiefensuche in ihren Super-Kategorien nach der Definition.

Kategorien erzeugen

Jede Kategorie muß beim `domains`-Paket mit einem Kategorie-Konstruktor erzeugt werden. Die Definition eines Kategorie-Konstruktors ähnelt der eines Domain-Konstruktors:

`CategoryConstructor::new(`
name,
parameter,
⟨ *parameter-test*, ⟩
super-kategorien,
axiome,
einträge...)

Die Argumente haben eine analoge Bedeutung wie bei Domain-Konstruktoren:

name gibt den Namen des Kategorie-Konstruktors an.

parameter gibt die formalen Parameter des Konstruktors an. Dies muß wieder eine Liste von Bezeichnern sein.

parameter-test gibt einen Ausdruck an, der zum Test der aktuellen Parameter dient. Sind keine Parameter vorhanden, so fehlt dieser Ausdruck wieder.

super-kategorien gibt die Super-Kategorien an, in denen die mit diesem Konstruktor erzeugten Kategorien enthalten sind.

axiome gibt Axiome für die erzeugten Kategorien an.

einträge ist eine beliebig lange Sequenz von Operatornamen oder Gleichungen der Form *name*=*eintrag*. Bei einer Gleichung wird *eintrag* wieder unter dem Index *name* in die Domains der Kategorie eingetragen. Wird nur ein

Operatorname angegeben, so wird damit angedeutet, daß dies eine Grundoperation ist, deren Implementation anderweitig erfolgen muß.

Die Argumente werden wieder erst bei der Erzeugung eines Domains der so definierten Kategorie evaluiert. Für `this` und *parameter* werden das zu erzeugende Domain bzw. die aktuellen Kategorie-Parameter eingesetzt.
Als Beispiel soll ein Konstruktor für die Kategorie der Euklidischen Ringe dienen:

```
EuclideanDomain:=
CategoryConstructor::new(
# Name #
    EuclideanDomain,
# Parameter: keine #
    [ ],
# Parameter-Test entfaellt #
# Super-Kategorien #
    [ FactorialDomain ],
# Axiome: keine #
    [ ],
# Eintraege #
    "rem",
    ...
):
```

In diesem vereinfachten Beispiel hat der Kategorie-Konstruktor keinen Parameter, es kann also nur eine Kategorie mit dem Konstruktor erzeugt werden. Ein Euklidischer Ring ist ein Faktor-Ring, der die zusätzlichen Operationen `"rem"` etc. besitzt. Die Operation `"rem"` ist eine Grundoperation, deren Implementation hier nicht angegeben werden kann. Wird `"rem"` ausgeführt, so ergibt dies einen Laufzeitfehler. Die Grundoperationen müssen schließlich noch von den jeweiligen Domain-Konstruktoren „ausgefüllt“ werden.
Die Berechnung des ggT kann mit Hilfe der Grundoperationen durch den Euklidischen Algorithmus definiert werden:

```
"gcd" = proc(x, y)
    local tmp;
begin
    x:= this::normalPart(x);
    y:= this::normalPart(y);

    while y <> this::zero do
        tmp:= this::rem(x, y);
        x:= y;
        y:= tmp
    end_while;
    this::normalPart(x)
end_proc
```

Die Einträge `"zero"` und `"normalPart"` werden anderweitig in Super-Kategorien definiert. Genau wie bei den Operationen der Domain-Konstruktoren wird bei der Konstruktion eines Domains der Bezeichner `this` durch das jeweilige Domain ersetzt.

Axiome definieren

Da Axiome nur Attribute sind und weiter keine Funktionalität haben, sind ihre Konstruktoren wesentlich einfacher als Domain- oder Kategorie-Konstruktoren. Sie werden wie folgt definiert:

`AxiomConstructor::new(`
  name,
  ⟨ *parameter*, ⟩
  ⟨ *parameter-test* ⟩)

Hier gibt *name* den Namen des Konstruktors an. Hat der Konstruktor Parameter, so werden diese mit *parameter* angegeben. Aktuelle Parameter werden mit *parameter-test* auf ihre Gültigkeit geprüft. Um auszudrücken, daß eine Operation invertierbar ist, gibt es z.B. den Axiom-Konstruktor `invertible`:

```
invertible:= AxiomConstructor::new(
    invertible,
    [ operator ],
    (if domtype(operator) <> DOM_STRING
     then
         error("wrong operator name")
     end_if)
):
```

Der Ausdruck `invertible("_mult")` etwa erzeugt ein Axiom, das besagen soll, daß die Operation `"_mult"` eines Domains invertierbar ist.

General Nonsense?

Welche Vorteile bieten die hier vorgestellten Konzepte gegenüber der Implementierung von Domains „von Hand“? Nun — die Notwendigkeit der Konstruktion parametrisierbarer Domains und der Vererbung von Implementationen dürfte unbestreitbar sein. Man könnte parametrierbare Domains natürlich auch durch normale Prozeduren erzeugen. Solch ein Vorgehen wäre aber recht unstrukturiert, da es jedem Entwickler überlassen wäre, die Domains zu erzeugen und Operationen zu definieren. Domain-Konstruktoren vereinheitlichen die Implementation von Domains und tragen dadurch zu einer besseren Wartbarkeit bei. Außerdem übernimmt das `domains`-Paket viele Standard-Aufgaben bei der Konstruktion und entlastet dadurch den Entwickler, unnötige Fehlerquellen und Code-Duplikation werden vermieden.

Dies trifft in einem vielleicht noch größerem Maße auch auf die Verwendung von Kategorien zu. Die Vorteile generischer Algorithmen — Vermeidung von Re-Implementationen für alle möglichen Domains, dadurch verminderter Implementations-Aufwand und geringere Fehleranfälligkeit — liegen auf der Hand. Die Entwicklung generischer Algorithmen wird durch Kategorie-Konstruktoren in geordnete Bahnen gelenkt. Der Benutzer darf — last but not least — zudem eine Vereinheitlichung bei der Verwendung der Domains erwarten.

Diese Vorteile sind nicht nur Fiktion. Von M. Monagan wurde mit GAUSS ein Library-Paket für MapleV entwickelt, welches die Definition von Domains und Kategorien erlaubt, siehe [2]. Domains sind in GAUSS einfache Tabellen, Domain-Konstruktoren und Kategorien Maple-Funktionen, die diese Tabellen manipulieren. Die Operationen der Domains werden als Tabellen-Einträge gespeichert. Die Benutzung der Domains und ihrer Operatoren ist etwas gewöhnungsbedürftig, da diese nicht in Maple integriert sind und nicht mit den normalen Operatoren der Maple-Sprache angesprochen werden können. Außerdem kann die Ausgabe von Domain-Elementen nicht auf eigene Bedürfnisse angepasst werden.

Trotz dieser Defizite wurde das Paket erfolgreich eingesetzt, um zum Beispiel für die Berechnung von Gröbner-Basen unterschiedliche Polynom-Datenstrukturen und Term-Ordnungen zu implementieren, siehe [1]. Die Implementations-Details konnten wegen der Abstraktion durch Kategorien „versteckt“ werden, so daß unterschiedliche Datenstrukturen und Ordnungen miteinander verwendet werden können, und trotzdem nur ein einziger „generischer“ Algorithmus zur Berechnung der Gröbner-Basen implementiert werden mußte. Jeder, der sich einmal die Implementation des „normalen“ Gröbnerbasen-Paketes `grobner` in Maple angeschaut hat, wird diese Abstraktion zu würdigen wissen.

Die meisten der Konzepte im `domains`-Paket gehen auf Ideen aus AXIOM zurück, siehe [3]. Axiome, Domain- und Kategorie-Konstruktoren sind in AXIOM Sprachkonstrukte, die anders als beim `domains`-Paket fest in der Syntax verankert sind. Alle Objekte in AXIOM sind streng typisiert, dies ist wegen der Typlosigkeit der MuPAD-Sprache im `domains`-Paket nicht möglich. Viele neuere Algorithmen wurden zuerst in AXIOM implemetiert, da man dort algebraische Strukturen zur Hand hat, die man losgelöst von Ihrer jeweiligen Implementation für seine Probleme verwenden kann.

Ein zwiespältiges Feature von AXIOM ist die automatische Typ-Konvertierung: Das System versucht, Eingaben vom Benutzer sofort nach der Eingabe zu typisieren, also typlose Ausdrücke in Domain-Elemente umzuwandeln. Im `domains`-Paket muß der Benutzer dagegen Typen explizit „von Hand“ konvertieren. Natürlich ist es für den Benutzer eine Erleichterung, keine Typen angeben zu müssen. Auf der anderen Seite ist es aber sehr schwierig, den Typ eines Ausdrucks zu bestimmen, AXIOM verwendet dazu vielfach zeitraubende und fehleranfällige Heuristiken.

Normalerweise wird man das `domains`-Paket nicht direkt an der „Benutzer-Schnittstelle“ verwenden, sondern eher zur internen Implementation von Library-Paketen. Es wird „Schnittstellen-Routinen“ geben, die Ausdrücke als Eingaben akzeptieren und diese intern in Domain-Elemente umwandeln. Die Konvertierung ist dabei unproblematisch, da man in dieser Situation ja weiß, was das zu lösende Problem und damit der gesuchte Typ ist.

Das `domains`-Paket liegt bisher nur in einer experimentellen Form vor. Die Infrastruktur ist vorhanden, aber es fehlt noch an algebraischem „Fleisch“. Es gibt zwar einige Grund-Kategorien, von der Basis-Kategorie „Typ“ über die Kategorien „Menge“, „Halbgruppe“, „Monoid“ etc. bis zur Kategorie „Körper“, diesen Kategorien fehlen aber noch viele wichtige Operationen. Insgesamt müssen noch weitere Erfahrungen gesammelt werden, die wiederum der Infrastruktur zugute kommen werden. Doch dies wird Teil einer späteren MuPAD-Version sein.

Literatur

[1] D. Gruntz. Groebner Bases in GAUSS. *MapleTech*, (9):36–46, 1993.

[2] D. Gruntz und M. Monagan. Introduction to GAUSS. *MapleTech*, (9):23–35, 1993.

[3] R.D. Jenks und R.S. Sutor. *AXIOM, The Scientific Computation System.* Springer, 1992.

PC^2 ... Zukunft des Parallelen Rechnens

Bernard Bauer, Alexander Reinefeld

***Supercomputing* und *Paralleles Rechnen* sind inzwischen in jedermanns Munde. Einige haben bereits erste Erfahrungen mit Anwendungen auf massiv parallelen Rechnern gesammelt, doch nur wenige nutzen die Möglichkeiten der neuen Technologien intensiv. Dabei bezweifelt kaum noch jemand, daß die äußerst rechenintensiven Probleme in Zukunft ausschließlich mit Parallelrechnern in vernünftigen Zeit- und Kostenmaßen gelöst werden können.**

Das Paderborn Center for Parallel Computing (kurz: PC^2 , sprich: P-C-Quadrat) möchte auf diesem Weg Begleiter und Partner für neue Anwender des Parallelen Rechnens werden. Dieser Artikel soll Ihnen einen kurzen Überblick über das PC^2 vermitteln. Da hier aber sicherlich nicht alle Fragen beantwortet werden können, sind Sie herzlich eingeladen, sich für weitere Informationen direkt an uns zu wenden.

Das PC^2 - Team

Das PC^2 ist eine zentrale wissenschaftliche Einrichtung der Universität-GH Paderborn. Die wissenschaftliche Verantwortung für die Arbeiten des PC^2 trägt ein interdisziplinär zusammengesetzter Vorstand, dem die in der nachstehenden Tabelle aufgeführten Professoren der Universität-GH Paderborn angehören. Vorstandsvorsitzender ist Prof. Dr. Burkhard Monien, auf dessen Initiative die Gründung des PC^2 zurückgeht. Dr. Alexander Reinefeld ist für die geschäftsführende Leitung verantwortlich. Zum weiteren Team des PC^2 gehören zur Zeit fünf wissenschaftliche und ein technischer Mitarbeiter sowie eine Sekretärin.

Professoren im Vorstand des PC^2

Prof. J. Lückel	Mechatronik
Prof. E. Mæhle	Datentechnik
Prof. B. Monien	theor. Informatik
Prof. F.-J. Rammig	prakt. Informatik
Prof. O. Rosenberg	Produktionswirtschaft

Aufgaben des PC^2

Das PC^2 - Team sieht seine Hauptaufgaben in der Nutzbarmachung und Verbreitung modernster Entwicklungen auf dem Gebiet des Parallelen Rechnens sowie in der Bereitstellung seiner Parallelrechnersysteme für interessierte Anwender. Daraus resultieren Aufgabenfelder in der Grundlagenforschung und anwendungsnahen Entwicklung, die durch Dienstleistungsangebote wie Aus- und Weiterbildung und einen überregionalen Rechenzentrumsservice im Bereich des *Parallel Computing* ergänzt werden.

Wie der interdisziplinär zusammengesetzte Vorstand schon erkennen läßt, liegt ein Schwerpunkt des PC^2 in der engen Kooperation zwischen Informatikern und Anwendern. Es schafft dadurch die Voraussetzungen zur Analyse und Überwindung der Probleme, die sich bei der Arbeit mit großen parallelen Systemen heute noch stellen. Das Besondere des PC^2 liegt in seiner einmaligen Symbiose von Theorie und Praxis.

Das PC^2 verfügt über eigene Parallelrechner, die den Mitgliedern der Universität-GH Paderborn und darüber hinaus auch externen interessierten und einschlägig arbeitenden Forschungsgruppen auf Antrag zur Nutzung zur Verfügung stehen. Derzeit arbeiten etwa 130 Benutzer in 60 Projekten.

Paralleles Rechnen

Die Parallelverarbeitung steht augenblicklich vor der allgemeinen Einführung im Bereich des Supercomputing. In vielen Anwendungsklassen ist inzwischen erkannt und auch anerkannt worden, daß nur durch den Einsatz von Parallelrechnern der enorm gestiegende Bedarf an Rechenkapazitäten gedeckt werden kann. Die praktische Berechnung der sogenannten *grand challenges* Probleme, wie z.B. Klimamodellierung oder Crash Simulationen, ist durch die Einführung massiv paralleler Rechensysteme in greifbare Nähe gerückt. Kein konventioneller Rechner wird je in der Lage sein, diese

Berechnungen in akzeptabler Zeit durchzuführen.
Nicht nur innovative kleinere Firmen, die bekanntlich am Markt immer in einer gewissen Vorreiterrolle fungieren, sondern auch etablierte Firmen haben massiv parallele Rechnersysteme eingeführt oder angekündigt. Daran hat die wissenschaftliche Forschung durch die Einbringung von neuen Ideen und Erkenntnissen natürlich einen entscheidenden Anteil. Bereits seit mehreren Jahren sind vielversprechende (theoretische) Ergebnisse bekannt, die jetzt beim Entwurf und Bau der Rechner selbst berücksichtigt werden bzw. mit Hilfe der Parallelrechner praktisch ausgeführt werden können.
Dennoch liegen nach wie vor viele Probleme in den interdisziplinären Bereichen sowie an der Schnittstelle zum Einsatz der Resultate in der Wirtschaft und Industrie. Außerdem besteht ein wachsender Bedarf an größeren Rechnersystemen, die der Forschung die Möglichkeit zur Überprüfung ihrer Ergebnisse geben und dadurch den Reifungsprozeß einer wissenschaftlichen Arbeit hin zu einem Produkt schneller ermöglichen.

Parallelverarbeitung, was ist das?

Anders als bei konventionellen sequentiellen Rechnern gibt es sehr viele unterschiedliche Architekturen für Parallelrechner. Gemeinsam ist allen, daß nicht nur ein Prozessor eine Aufgabe allein bearbeitet, sondern mehrere oder sogar sehr viele Prozessoren gleichzeitig zusammen eine Aufgabe lösen. Parallel arbeiten bedeutet zwangsläufig, daß die Prozessoren miteinander Informationen austauschen oder bestimmte Informationen (z.B. über die Terminierung) allen Prozessoren zugänglich sein müssen.
Am einfachsten ist es, wenn jeder Prozessor ein vollständiger, autonomer Rechner ist. Dann wird im leichtesten Fall das Problem in Teilprobleme partioniert und jedem Prozessor „seine" Teilaufgabe zugewiesen, die er zu bearbeiten hat. Der Informationsaustausch oder die Kommunikation zwischen Prozessoren wird mittels einzelner Nachrichten realisiert. Idealerweise teilen sich die Prozessoren dabei einen globalen, allen direkt zugänglichen Speicher (shared memory). Prozessor x schreibt dann seine Nachricht für Prozessor y einfach in den Speicherbereich des Prozessors y. Die Skalierbarkeit eines solchen Systems mit einer größeren Anzahl von Prozessoren ($\geq$ 100) ist aber bis heute noch nicht praktisch gezeigt worden. Problematisch sind gleichzeitige Schreib-Lese-Operationen auf gleiche Speicherzellen sowie die Konstruktion einer vernünftigen Verbindungsstruktur jedes einzelnen Prozessors.

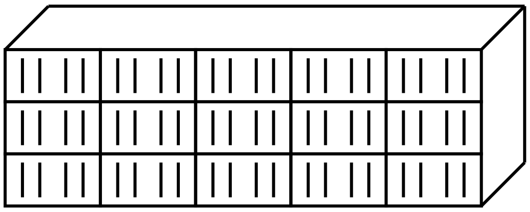


Abbildung 1: SC320 Darstellung

Die Prozessoren der Parallelrechner im PC^2 besitzen jeweils einen eigenen Hauptspeicher (distributed memory) und tauschen ihre Informationen direkt und synchronisiert aus.

Transputer

Der *Transputer* ist von der britischen Firma Inmos für die Parallelverarbeitung speziell entwickelt worden. Er besitzt neben den typischen Merkmalen eines gewöhnlichen Prozessors vier Verbindungskanäle, die sogenannten Links. Über diese Links können Transputer miteinander verknüpft werden und über dieselben dann kommunizieren. Ein Hardware-Scheduler sorgt dafür, daß dabei die Kommunikation unabhängig von der Rechenleistung ablaufen kann. Zur Zeit stehen zwei Systeme auf Transputerbasis zur Nutzung bereit: SC320 und GCel-1024.
Der „SuperCluster" SC320 ist das größte frei konfigurierbare Transputersystem weltweit. Hier können elektronisch alle Verbindungsstrukturen, deren Knoten maximal vier Nachbarn besitzen (math.: Grad $\leq$ 4), zwischen den 320 Transputern geschaltet werden. Die Gesamtleistung der Prozessoren vom Typ T800 summiert sich auf 1.1 GFLOPs (= 1.1 Milliarden Gleitpunktoperationen pro Sekunde) bei verfügbaren 1.2 GByte Hauptspeicher.
Der „GigaCube" GCel-1024 ist mit 1024 Prozessoren ein weiterer Superlativ. 256 Transputer gehören der Herzogenrather Firma ZiAM GmbH (Zentrum für Innovative Anwendungen Massiv paralleler Systeme), die mit Unterstützung der nordrhein-westfälischen Landesregierung kleinen und mittelständischen Unternehmen den Einstieg und Zugriff auf Parallelrechner ermöglicht. Als zweidimensionales Gitter sind die Transputer des GCel fest miteinander verbunden. Eine betriebssytemunterstützte (Software-) Umgebung be-

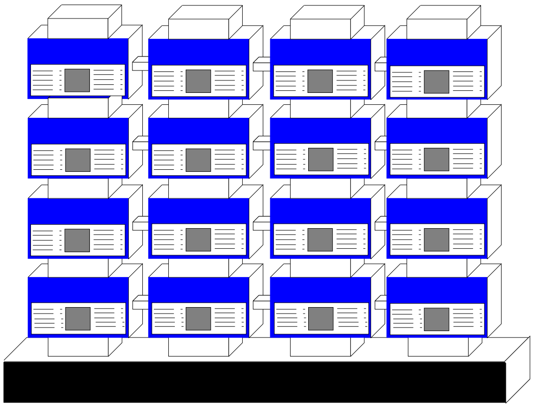


Abbildung 2: GCel-1024 Darstellung

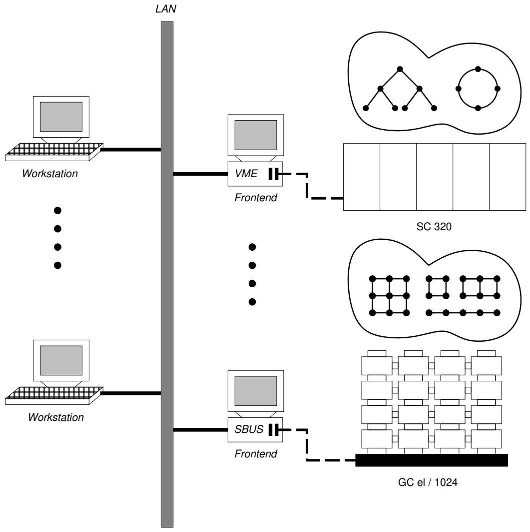


Abbildung 3: Zugang zu den Ressourcen

freit den Anwender jedoch von speziellen Positionskenntnissen und läßt die Prozessoren virtuell als Cluster erscheinen. Die eingesetzten T805 Transputer erreichen zusammen eine Rechenkapazität von 4.4 GFLOPs bei 4 GByte Hauptspeicher.

Beide Systeme sind als Multi-User-Systeme ausgelegt. Jeder Benutzer kann also, sofern verfügbar, einen Teil der Transputer für sich reservieren und diese exklusiv nutzen. Dies ist nicht zu verwechseln mit dem Multi-User-Betrieb im Time-Sharing auf einer Unix-Workstation, wo viele Benutzer gleichzeitig einen Prozessor im Zeitscheibenverfahren benutzen.

Die Parallelrechner-Systeme sind in Netzwerke (WIN) eingebunden, die auch den Zugriff für hochschulexterne Nutzer ermöglichen. Wissenschaftler aus ganz Deutschland und benachbarten europäischen Staaten nutzen diesen Zugang bereits intensiv; industriellen Interessenten sind die Systeme als Dienstleistungsrechner über die ZiAM GmbH zugänglich. Unabhängig hiervon werden die Rechnersysteme des PC^2 in gemeinsamen Forschungs- und Entwicklungsvorhaben eingesetzt.

Forschungsprojekte des PC^2

Naturgemäß gibt es im relativ jungen Gebiet des Parallelen Rechnens sehr viele offene Probleme, bei deren Erforschung sich das PC^2 - Team aktiv beteiligt. So wurde z.B. im PC^2 eine „Computing Center Software“ (CCS) entwickelt, die den Benutzern eine homogene Schnittstelle zu einem Netzwerk heterogener paralleler Supercomputer bietet. Gleichzeitig wird dabei die Ausnutzung der Rechenkapazitäten optimiert. Das modulare Softwarepaket besteht aus einer Menge asynchroner Prozesse, die miteinander über nicht blockierende Remote Procedure Calls kommunizieren. Ein Vorteil dieser Konzeption liegt in der Fehlertoleranz der verteilten CCS-Prozesse.

Für die Darstellung der Ressourcen wurde eine sog. "Resource Description Language" (RDL) entwickelt, mit der einerseits das physikalische Parallelrechnerumfeld spezifiziert und andererseits Resourceanforderungen formuliert werden können. RDL nimmt somit eine Schlüsselrolle im automatisierten Betrieb von Parallelrechnern ein.

CCS ist ein gutes Beispiel für ein erfolgreiches Joint Venture, das das PC^2 zusammen mit der RWTH Aachen und der Firma Parsytec durchgeführt hat. Die Software wird heute nicht nur in Paderborn, sondern auch in Aachen, Köln und demnächst auch in Athen, Warwick (UK) und Zürich eingesetzt.

Das Projekt „Virtuelle Topologien“ befaßt sich mit der Realisierung häufig verwendeter Kommunikationsstrukturen (virtuelle Topologien) auf Multiprozessorsystemen ohne gemeinsamen Speicher. Dazu wurden Methoden zur Erzeugung virtueller Topologien untersucht und Funktionen implementiert, die die Topologien erzeugen. Einige Topologiefunktionen sind in einer Anwenderbibliothek zusammengefaßt, die als Bestandteil der Laufzeitumgebung PARIX von der Firma Parsytec einem weiteren Kundenkreis kommerziell angeboten wird.

Im Projekt „Globale Funktionen“ werden häufig benutzte Standard-Kommunikationsfunktionen für Multiprozessorsysteme optimal realisiert. Einige globale Funktionen, wie z.B. das globale Minimum, sind in einer Anwenderbibliothek enthalten, die Be-

standteil der Laufzeitumgebung PARIX ist.
Außerdem agiert das PC^2 als Subcontractor im ESPRIT-Projekt GP-MIMD, das die Evaluierung der praktischen Nutzbarkeit massiv paralleler Systeme zum Ziel hat. Das PC^2 unterstützt einen ausgewählten Anwenderkreis bei der Portierung seiner (z.T. bereits parallelisierten) Anwendungsprogramme auf ein massiv paralleles System. Die bei der Portierung gesammelten Erfahrungen werden bewertet und in Statusberichten zusammengefaßt.
Einen insbesondere nach außen sichtbaren Erfolg konnte das PC^2 mit der Gründung der ZEUS-Initiative verbuchen, die im November 1992 in Paderborn vollzogen wurde. *ZEUS* steht für „Zentren EUropäischen Supercomputings“ und ist ein Zusammenschluß vergleichbarer Institutionen in Amsterdam, Athen, Köln, Daresbury (UK) und Paderborn. Die ZEUS-Zentren kooperieren im EG-Kontext u.a. bei der Portierung existierender Anwendungen auf hochparallele Rechnersysteme.

Veranstaltungen

Trotz seines erst jungen Bestehens kann das PC^2 auf eine ansehnliche Veranstaltungsbilanz im letzten Jahr zurückblicken. Unter der Mitwirkung des PC^2 wurden Konferenzen, Workshops und Seminare erfolgreich durchgeführt. Hier sind das dreitägige 1. Internationale Heinz Nixdorf Symposium (11/92) mit dem Thema *Parallel Architectures and their Efficient Use*, die ZEUS-Gründung (11/92) oder das „5th Annual ACM Symposium on Parallel Algorithms and Architectures“, SPAA'93, (7/93) als wichtigste Tagungen zu nennen. Das für alle Interessenten offene PC^2 -Kolloquium bietet während der Vorlesungszeiten regelmäßig ein breites Spektrum interessanter Vorträge, von Einführungen in die Parallelverarbeitung über die Darstellung spezieller Forschungsgebiete bis hin zu Firmenpräsentationen.

Perspektiven/Aussichten

Das PC^2 verfügt über einen Kreis sehr aktiver Anwender, die weitere Rechenleistung benötigen und fordern. Die erste Aussicht richtet sich daher eindeutig auf den Ausbau bzw. die Erweiterung der vorhandenen Ressourcen. Nach langem Warten scheint in 1994 der Markteinführung der Transputerweiterentwicklung T9000 nichts mehr entgegenzustehen. In Verbindung mit speziellen Routingchips (C104) lassen sich neben der mindestens 5-fachen Leistungssteigerung entscheidende Fortschritte und Vereinfachungen in der Kommunikation erzielen. Hier will das PC^2 zunächst ein System mit 320 Prozessoren anbieten, welches die doppelte Leistung des derzeitigen 1024er Systems bietet. Darüberhinaus sind weitere Kapazitäten in der konkreten Planung.
Eine weitere Perspektive und wichtiger Bestandteil der Arbeiten in 1994 für das PC^2 werden Förderung des Parallelen Rechnens in Wirtschaft und Industrie in Zusammenarbeit mit der ZiAM GmbH sein. Hier sehen wir die Zeit für gekommen, über den akademischen Horizont hinaus deutliche Akzente zu setzen. Auf der CeBit '94 ist das PC^2 auf dem Gemeinschaftsstand des *Forschungsland NRW* präsent und freut sich auf Ihren Besuch.
Sie können uns aber auch gerne vorher kontaktieren, wenn Sie nähere Informationen haben möchten oder wir etwas für Sie tun können.

PC^2
PADERBORN CENTER FOR PARALLEL COMPUTING

Universität–GH Paderborn
Warburger Str. 100
33095 Paderborn, Germany

phone: +49 5251 60 3342
fax: +49 5251 60 3436
email:
pc2–team@uni–paderborn.de

Eine kleine Einführung in Neuronale Netze

Andreas Kemper

Dieser Artikel ist der erste einer Reihe von Beiträgen, die einen kleinen Einblick in das weite Gebiet der Neuronalen Netze und ihrer Anwendungen geben wollen.
Der erste Teil beschäftigt sich mit dem biologischen Vorbild der Neuronalen Netze, dem Gehirn. Außerdem möchte ich auf eine erstes stark vereinfachtes Modell biologischer Neuronen, die sogenannten Schwellwertelemente, eingehen.

Biologische Neuronen

Defizite herkömmlicher Rechner und Algorithmen

Im Gegensatz zu herkömmlichen Rechenanlagen weist das menschliche Gehirn einige Fähigkeiten auf, die mit Hilfe von Computern überhaupt nicht oder nur sehr eingeschränkt simuliert werden können. So ist der Computer zwar in der Lage innerhalb von Sekundenbruchteilen zwei viele hundert Stellen große Zahlen miteinander zu multiplizieren, aber schon bei der vergleichsweise einfachen Aufgabe, einen Baum innerhalb eines Bildes zu erkennen tut er sich sehr schwer oder versagt sogar vollständig. Ein weiterer wichtiger Unterschied ist die Fehlertoleranz, mit der das Gehirn arbeitet. Während beim Ausfall eines Bauteiles eines Computers dieser in der Regel nicht mehr funktionstüchtig ist, stellt das Absterben einer einzelnen Zelle des Gehirns für seine Funktionsfähigkeit keine Gefahr dar. Außerdem besitzt das Gehirn die Fähigkeit zu lernen. Es kann sein zukünftiges Verhalten aufgrund vergangener Ereignisse verändern, also aus Erfahrungen lernen.
Diese Diskrepanz liegt in der vollständig unterschiedlichen Struktur des menschlichen Gehirns und eines herkömmlichen von Neumann Rechners begründet. Während im Falle herkömmlicher Rechner zwischen Programm und Daten unterschieden wird, ist diese Trennung in denen biologischen Systemen wie dem Gehirn nachempfundenen *Neuronalen Netzen* nicht vorhanden.

Die besonderen Eigenschaften des Gehirns sind daher die Motivation, verschiedene Strukturen und Funktionen des biologischen Gehirns mit Hilfe von Maschinen zu simulieren, um so Aufgaben zu bearbeiten, an deren effizienter Lösung man mit herkömmlichen Algorithmen und Rechner-Architekturen scheitert.

Das Gehirn

Das Gehirn besteht aus einem komplexen Geflecht auf vielfältige Art und Weise vernetzter Zellen, die miteinander Signale austauschen. Diese Zellen befinden sich auf der *Hirnrinde* (Neokortex), die aus einer 2-3 mm starken Schicht aus Nervenzellengewebe besteht. Diese Rinde ist stark gefaltet und stellt eine Fläche von etwa 0.2 qm dar. Innerhalb dieser Schicht lassen sich verschiedene Gebiete unterscheiden, denen speziellen Funktionen wie visuelle Wahrnehmung oder die Bewegungskontrolle zugeordnet sind. Jedoch ist niemals eine einzelne Zelle für eine konkrete Aufgabe allein verantwortlich. Insgesamt besteht das menschliche Gehirn aus etwa 10^{11} bis 10^{12} Nervenzellen, sogenannten *Neuronen.*

Neuronen

Ein Neuron besteht im wesentlichen aus drei Teilen: dem *Dendritenbaum*, der die Eingabe des Neurons darstellt, dem *Zellkörper* (Soma), der als Verarbeitungselement agiert, sowie dem *Axon*, das das Ergebnis weiterleitet. Der Dendritenbaum, der eine stark verästelte Baumstruktur darstellt, führt dem Zellkörper elektrische Impulse zu. Diese können sowohl *hemmend* (inhibitorisch) als auch *erregend* (exzitatorisch) wirken. Aufgabe des Zellkörpers ist es, diese eingehenden Signale aufzusummieren. Überschreitet diese Summe einen Schwellwert, so gibt der Zellkörper über das Axon einen Ausgabeimpuls ab. Auch das Axon verzweigt sich an seinem Ende in einige tausend Äste. Am Ende dieser Äste befinden sich die sogenannten *Synapsen.* Sie stellen den Übergabepunkt zwischen einem Axon der Ausgabezelle und der nächsten Zelle dar. Hier-

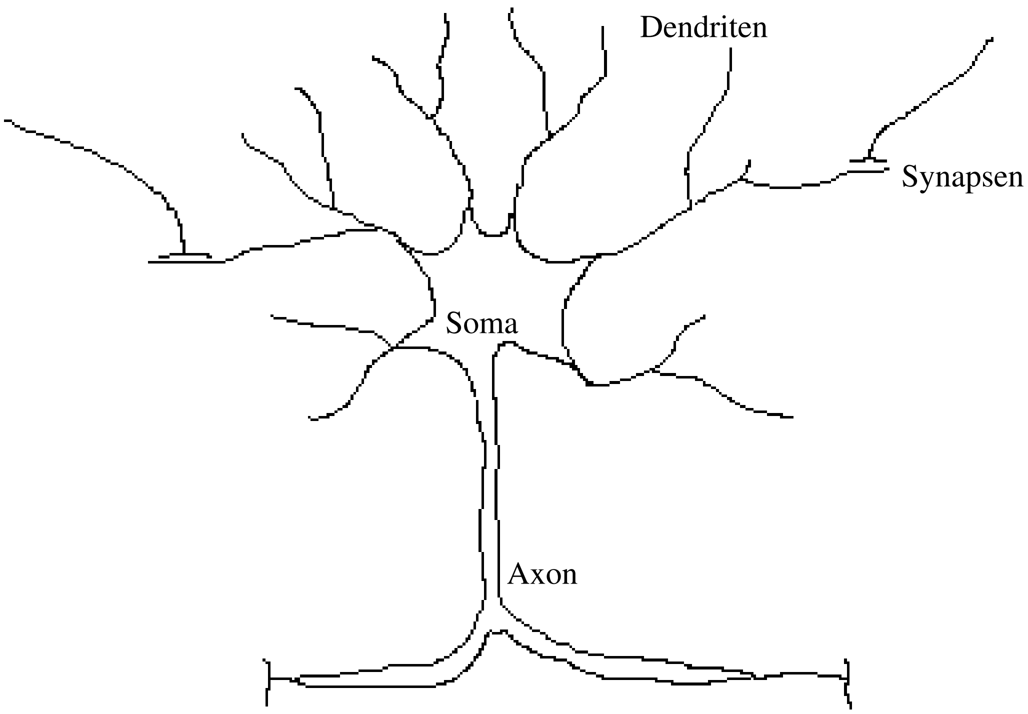


Abbildung 1: Neuron

bei handelt es sich um "chemische Kontakte", die durch die Ausschüttung einer chemischen Substanz (Neurotransmitter) realisiert werden. Diese Synapsen können sich dabei sowohl direkt am Zellkörper wie auch an einem Ast des Dendritenbaumes befinden. Die Synapse bestimmt hierbei, ob das von ihr weitergegebene Signal erregend oder hemmend auf die Zelle wirkt, der das Signal zugeführt wird. Innerhalb des Gehirns ist jeder Zellkörper über seinen Dendritenbaum mit durchschnittlich etwa 10.000 Synapsen verbunden. Jedoch gibt es auch Hirnregionen mit einer weitaus größeren Zahl von synaptischen Eingaben pro Zelle. So existieren im Kleinhirn (Cerebellum) Zellen mit jeweils etwa 80.000 synaptischen Eingängen. Das Signal, das im Falle des Überschreitens des Schwellwertes über das Axon abgegeben wird, besteht aus einer periodischen Entladung, deren Frequenz proportional zur Größe der aufsummierten Eingaben der Zelle ist. Es handelt sich also keinesfalls um eine rein binäre Ausgabe. Die Stärke jeder einzelnen Entladung ist jedoch konstant. Da der Zellkörper nach jeder Entladung jedoch eine Regenerationszeit von mindestens 1 ms benötigt, ist die maximale Taktfrequenz auf 1 KHz beschränkt. Durch diese Frequenzmodulation ist das Axon in der Lage, nichtbinäre Information weiterzuleiten.

Die Signale werden innerhalb der Axone durch Depolarisierung der Zellwände weitergeleitet. Dieses Signal wird durch die Verästelung des Axons nicht aufgeteilt, sondern wird in konstanter Stärke in jeden Ast des Axons geleitet. Die Ausbreitungsgeschwindigkeit eines Signals kann sehr unterschiedlich sein. So beträgt die Ausbreitungsgeschwindigkeit innerhalb des menschlichen Gehirns etwa 0,5 bis 2 m/s. Dies erlaubt zwei Gehirnzellen innerhalb von 20 bis 40 ms miteinander zu kommunizieren, ist jedoch für die Kommunikation mit Zellen innerhalb der Gliedmaßen wesentlich zu langsam. Daher existieren hier Mechanismen, die die Geschwindigkeit auf etwa 100 m/s erhöhen.

Synapsen

Die Stärke einer Synapse, mit der sie das ankommende Signal weiterleitet, ist nicht für alle Zeiten konstant. Eine Synapse, die wiederholt das Eingabesignal an das folgende Neuron weitergibt, wird ihre Stärke steigern, während andere Synapsen schwächer werden. Die Wirksamkeit einer Synapse ändert sich also in Abhängigkeit der Korrelation von präsynaptischen und postsynaptischen Aktivitäten. Dieses schon sehr früh von Donald Hebb aufgestellte und heute experimentell bewiesene Gesetz stellt einen der Grundmechanismen des

selbstständigen Lernens dar und wird häufig als *Hebb'sche Regel* bezeichnet. Das menschliche Gehirn umfaßt ca. 10^{15} Synapsen.
Die Größe eines Neurons kann zusammen mit seinem Dendritenbaum und dem Axon zwischen 0.01 mm (innerhalb des Gehirns) und einem Meter (innerhalb der Gliedmaßen) schwanken. Innerhalb des menschlichen Gehirns unterscheidet man im wesentlichen zwei Typen von Neuronen. Dies sind die sogenannten *Pyramidenzellen* sowie die *Sternzellen*. Die Pyramidenzellen haben meist weitreichende Axone mit anregend wirkenden Synapsen, während sich im Fall der Sternzellen die Axone sternförmig in der unmittelbaren Umgebung des Zellkernes verästeln und deren Synapsen zumeist hemmend wirken.

Formale Neuronen

Das Ziel neuronaler Netze ist es nicht, biologische Strukturen vollständig zu simulieren. Vielmehr sollen lediglich einige als wichtig erscheinende Grundkonzepte übernommen werden. Vielen dieser Modelle gemein ist das formale Neuron, das auf *McCulloch* und *Pitts* zurückgeht. Hierbei wird ein Neuron als ein *logisches Schwellwertelement* betrachtet, das zwei mögliche Zustände besitzen kann. Ein solches Neuron besitzt eine beliebige Anzahl von Eingängen x_i sowie einen Ausgang y. An jedem Eingang liegt im binären Fall einer von zwei möglichen Werten an (z.B. 1/0). Das Schwellwertelement multipliziert jeden dieser Werte mit einem jeder Eingangsleitung zugeordneten Gewicht w_i und addiert die sich ergebenden Werte. Überschreitet die so gebildete Summe einen bestimmten Schwellwert S so gilt das Neuron als erregt und gibt über seine Ausgangsleitung einen entsprechenden Wert ab. Dieser wird mit Hilfe der sogenannten *Ausgabefunktion* θ ermittelt.

$$y = \theta(\sum_i w_i x_i - S) \tag{1}$$

In einem binären Modell kann z.B. $\theta(x) = 0$ für $x \leq 0$ und $\theta(x) = 1$ für $x > 0$ als Ausgabefunktion gewählt werden. (Heavyside-Funktion).
Eine andere Möglichkeit ist die Wahl von -1 und $+1$ als Ausgabewerte. In diesem Fall kann als Ausgabefunktion z.B. die Vorzeichenfunktion *sign* verwendet werden.
Dieses Modell vernachlässigt selbstverständlich eine Vielzahl von Faktoren, wie bidirektionale elektrische Synapsen, Fragen der Energieversorgung sowie die Weiterleitung der Signale innerhalb der Dendriten und Synapsen, die man mit Hilfe von Differentialgleichungen (Kabelgleichungen) beschreiben kann.

Wirkliche Neuronen liefern oft eine nicht diskrete Ausgabe. Die Nichtlinearität der Ausgabefunktion ist eine weitere universelle Eigenschaft biologischer Neuronen. Dabei ist eher die Existenz dieser Nichtlinearität als ihre genaue Form von Bedeutung. Im Gegensatz zu diesem Modell produzieren biologische Neuronen, wie bereits beschrieben, eine Folge von Implusen. Die Berechnung der Ausgabe geschieht nicht gleichzeitig in allen Neuronen. D.h. es ist kein globaler Takt vorhanden. Diese Eigenschaft wird in einigen später beschriebenen Modellen simmuliert.

McCulloch und Pitts zeigten, daß durch geeignete Kombinationen derartiger formaler Neuronen jeder endliche logische Ausdruck berechnet werden kann. Logische Funktionen wie AND, OR oder NOT können mit Hilfe eines einzigen Neurons dargestellt werden. Die Funktion XOR benötigt, wie später gezeigt wird, jedoch mehrere Neuronen. Dieses Ergebnis ist in erster Linie von theoretischer Bedeutung, da es besagt, daß mit Hilfe neuronaler Netze alles das berechnet werden kann, was auch mit herkömmlichen Rechnern berechenbar ist. Dies ist aber für neuronale Netze nicht von Bedeutung, da sie ja eingesetzt werden, um Phänomene zu modellieren, die mit herkömmlichen Rechnern nur sehr schwer oder überhaupt nicht zu berechnen sind. Außerdem fehlt dem Ansatz von McCulloch und Pitts eine vernünftige Modellierung der Lernfähigkeit biologischer Systeme.

Die folgende Tabelle stellt die Funktion NOT als Schwellwertfunktion dar. Die booleschen Werte TRUE und FALSE werden hier als 1 und 0 dargestellt. Das entsprechende Neuron besteht aus einem Eingabeelement, das mit dem Gewicht w multipliziert wird. Die Schranke S trägt den Wert -1/2. Als Ausgabefunktion wird die Heavyside-Funktion verwendet.

x	wx	S	$\theta(wx - S)$
0	0	$-1/2$	1
1	-1	$-1/2$	0

Für die logische Funktion AND ist ein Neuron mit zwei Eingaben zu definieren. Die Eingaben werden mit den Gewichten w_1 und w_2 gewichtet, die hier beide den Wert 1 tragen. Die Schranke S beträgt $-3/2$. Damit ergibt sich die folgende Wertetabelle:

x_1, x_2	$\sum_{i=1}^2$	S	y
$0,0$	0	$-3/2$	0
$0,1$	1	$-3/2$	0
$1,0$	1	$-3/2$	0
$1,1$	2	$-3/2$	1

Wie bereits erwähnt, wird die absolute Größe des aufsummierten Signals in biologischen Netzen durch die Frequenz des binären Ausgangsimpulses weitergegeben. Dabei ist die Summe der gewichteten Eingabeaktivitäten in weiten Bereichen zur Ausgangsfrequenz proportional. In einem reellen Modell läßt sich dieser Effekt durch eine Rampenfunktion modellieren. Hierbei gibt die Ausgabefunktion bis zum Erreichen des Schwellwertes eine 0 aus, steigt dann proportional zur Eingabe bis zum Erreichen eines Sättigungswertes und bleibt dann konstant.
Ein Beispiel einer solchen Funktion ist

$$y := \begin{cases} z_{max} & z \geq S \\ kz & -S \leq z \leq S \\ -z_{max} & z \leq -S \end{cases} \tag{2}$$

wobei k durch $k := z_{max}/S$ gegeben ist.
Um die stetige Differenzierbarkeit der Ausgabefunktion zu erreichen, werden in vielen Fällen andere nichtlineare Ausgabefunktionen verwendet, die den beobachteten neurologischen Sättigungseffekt modellieren. Hierbei handelt es sich um sogenannte *Squashing Functions* (Quetschfunktionen) die auch als *sigmoide* Funktionen bezeichnet werden. Das Verhalten eines neuronalen Netzes ist jedoch kaum von der genauen Form der Quetschfunktion abhängig. Beispiele für Quetschfunktionen sind die *Fermi*-Funktion

$$S_F(z) := (1 + \exp(-kz))^{-1} \tag{3}$$

der hyperbolische Tangens

$$S_T(z) := 2S_F(2z) - 1 = \tanh(kz) \tag{4}$$

sowie die Kosinus-Quetschfunktion:

$$S_C(z) := \begin{cases} 1 & z \geq \pi/2 \\ \frac{1+\cos(z-\pi/2)}{2} & -\pi/2 \leq z \leq \pi/2 \\ 0 & z \leq -\pi/2 \end{cases} \tag{5}$$

Im nächsten Teil dieser kleinen Reihe möchte ich Neuronale Netzte abstrakt definieren und kategorisieren. Außerdem möchte ich auf ein erstes wichtiges Modell, das Perzeptron, eingehen.

Literatur

[1] J. Reinhardt B. Müller. *Neural Networks, An Introduction.* Springer Verlag, 1990.

[2] Rüdiger Brause. *Neuronale Netze.* B.G. Teubner, 1991.

[3] R. Ferber. Neuronale netze. In *Überblicke Mathematik, Jahrbuch 1992*, pages 137–157. S.D. Chatterji, B. Fuchssteiner, U. Kulisch, R. Liedl, W.Purkert, 1992.

[4] Klaus Schulten Helge Ritter, Thomas Martinez. *Neuronale Netze: Eine Einführung in die Neuroinformatik selbstorganisierender Netzwerke.* Addison Wessley, 1990.

[5] Helen Morton Igor Aleksander. *Neural Computing.* Chapmann and Hall, 1990.

[6] Richard G. Palmer John Hertz, Anders Krogh. *Introduction to the Theory of Neural Computation.* Addison Wesley, 1991.

[7] Monika Köhle. *Neuronale Netze.* Springer-Verlag, 1990.

[8] Palmer Krogh. *Introduction to the Theory of Neural Computation.* Addison Wesley, Indianapolis, 1991.

[9] Philip D. Wasserman. *Neural Computing Theory and Practice.* Van Nostrand Reinhold, New York, 1989.

Arbeitskreis „Mathematik in Forschung und Praxis" Mathematik in der Produktionstechnik

Karsten Morisse

Bereits zum siebenten Mal traf sich der 1990 gegründete Arbeitskreis „Mathematik in Forschung und Praxis" am 25. und 26. Oktober im Physikzentrum Bad Honnef. Initiiert vom Wissenschaftszentrum NRW bildet der Arbeitskreis ein interdisziplinäres Forum, in dem Fachleute aus unterschiedlichen wissenschaftlichen und beruflichen Arbeitsgebieten im Dialog aufeinandertreffen. Ziel ist es, Informationen sowie Entwicklungs- und Anwendungsmöglichkeiten der anwendungsorientierten Mathematik zu diskutieren und auszutauschen.

Die große Akzeptanz des Arbeitskreises (mehr als 400 Mitglieder) machte eine Regionalisierung notwendig. Bislang vom Wissenschaftszentrum NRW getragen, ist der Arbeitskreis nun in 8 Regionalgruppen gegliedert, die jeweils von einem Paar aus einem Industrieangehörigen und einem Hochschulforscher geleitet werden.

Thema des diesjährigen Herbstsymposiums, zu dem die Herren Kallrath (BASF AG) und Bock (IWR, Universität Heidelberg) eingeladen hatten, war „Mathematik in der Produktionstechnik". Die Grundlage für die zahlreichen und lebhaften Diskussionen der ca. 60 Teilnehmer bildeten 11 Vorträge, in denen mathematische Anwendungen aus den unterschiedlichsten industriellen Bereichen präsentiert wurden. Die Mehrheit der Vortragenden kam dabei aus der Industrie.

Einen Schwerpunkt bildete die *Robotik/Automation*. Neben der Modellierung der Kalibration und der Kraftregelung von Industrierobotern, wurden auch Off-Line Programmierung, Simulation und Bahnoptimierung angesprochen. *Energieproduktion* bildete einen weiteren Schwerpunkt. Fragestellungen zur Kostenoptimierung bei der Energieerzeugung unter Einsatz verschiedener Produktionsformen sowie die Spannungssteuerung in einem Hochspannungsverbundnetz standen hier im Vordergrund. Methoden der mathematischen Optimierung (lineare und nichtlineare Optimierung) sind dabei ein probates Mittel, die zur Einsparung von zigtausend Mark pro Tag führen.

Qualitätssicherung und *Chemische Industrie* waren weitere Themenkomplexe. Daß sich auch mit Reststoffen unter Einsatz der linearen Optimierung noch viel Geld verdienen läßt wurde ebenso präsentiert wie ein Softwarepaket zur Produktionsplanung in der chemischen Industrie sowie der Einsatz statistischer Methoden bei Produktion und Prüfung chemischer Produkte.

Interessante Anwendungen der *Fuzzy-Logik* wurden im letzten Schwerpunkt vorgestellt. Bei der Suche nach geeigneten Bohrstellen für Erdöl und Erdgas müssen Gesteinsformationen untersucht werden. Für die Bohrungsplanung sind die Formationsteufen eine wichtige, jedoch aufwendig zu bestimmende und fehleranfällige Einflußgröße. Zur Gewinnung minimale Fehlerintervalle für diese Kenngröße wird Fuzzy-Logik eingesetzt. Daß unser leibliches Wohlbefinden in Autos und Bussen in Zukunft mittels smarter Fluide „fuzzyfiziert" wird, wurde im letzten Vortrag vorgestellt. Fuzzy-Logik wird dabei zur Steuerung der Charakteristik einer Flüssigkeitsdämpfung eingesetzt.

Ergänzt wurden die Vorträge durch zwei Special Events. M. Grötschel, der Vorsitzende der DMV, schilderte Beiträge der kombinatorischen Optimierung zu Computerentwurf und -fertigung. Die Heidelberger Soziologin C. Bender trug über die Wissensgenese und neue Produktionskonzepte in der Industrie vor und spannte dabei einen Bogen von „Toyota bis Plato". In den Beiträgen wurden die gestaltenden strukturellen Veränderungen in den verschiedensten Bereichen durch die Mathematik begrüßt und als außerordentlich wichtig empfunden. Daß *High-Technology* gleich *Math-Technology* ist wurde ebenfalls herausgearbeitet.

Den Abschluß des Symposiums bildete eine Podiumsdiskussion zum Thema „Bedarfsorientierte Forschung — Fluch oder Segen". Konsens erzielte man darin, daß eine strikte Trennung in Grundlagenforschung und bedarfsgesteuerte Forschung nicht möglich ist, da diese beiden Dinge, zumindest teilweise, eng miteinander verquickt sind.

„Euromath?“

Benno Fuchssteiner

Vor etwa 10 Jahren machten die europäischen Mathematiker sich auf, um den technischen Fortschritt mitzugestalten: *Euromath* **wurde gegründet, um eine moderne kommunikationstechnische Arbeitsumgebung für Wissenschaftler, insbesondere für Mathematiker, zu entwickeln. In Folge der seit 1983 laufenden Vorbereitungen, und ausgestattet mit einem Grundsatzbeschluß des European Mathematical Council (1986 in Oxford) wurde eine paneuropäische Organisation, der European Mathematical Trust, gegründet, sowie verschiedene Project Groups, Steering Comittees und nationale Koordinationsausschüsse gebildet. Trotzdem, und obwohl heute die technische Verantwortung weitgehend vom Vertreter der DMV in der Euromath-Leitung getragen wird, blieb Euromath unter deutschen Mathematikern umstritten.** mathPAD **wollte die Erfolge von Euromath hinterfragen und sprach deshalb mit Dr. Flemming Topsoe, Dozent an der Universität Kopenhagen, Chairman des dortigen Euromath Zentrums und Chairman des Europäischen Mathematischen Trusts. Der Interviewer war Benno Fuchssteiner.**

mathPAD: Please, could you describe the goals of Euromath?

Topsoe: The shortest possible description: to create a modern research infrastructure for mathematics in Europe.

mathPAD: How will the mathematicians benefit?

Topsoe: A lot of things which are developed in computer technology are being exploited in ways that we had not seen 15 years ago. Euromath wants to assist the research environment by production and exchange of scientific articles, by creation of data banks, environment forums for communication and conferences, and all these kind of things.

mathPAD: Would you consider the goals of Euromath as ambitious or as goals easy to achieve. How much has been achieved?

Topsoe: Very, very ambitious and, if you consider the potential, very little has been achieved. I am sure that if we meet in ten years and see how researchers use computers as a general research tool then very little has as been realized today but our effort has catalized a healthy development.

mathPAD: What has been achieved in concrete details? What is the result of all the work invested into the project?

Topsoe: Best we start by focusing on very concrete things. We have developed a software tool, the Euromath System, which will assist us. A preliminary version was released on the occasion of the European Congress in Paris last year. This is a development version, that means its user interface is not quite as good as it should be and the facilities are limited. It is a structure editor which indicates to the users in which way the development is going for producing documents and exchanging them, and what the potential for multi-usage of documents is.

mathPAD: What is a structure editor?

Topsoe: A software tool capable of processing documents having a well defined structure. In so doing, the editor keeps track of the structure and ensures consistency. This makes multi-usage of documents possible, for example, import and export to computer algebra systems. As the focus is on structure — a feature readily appreciated by mathematicians — the actual presentation of documents (on screen, paper, tape, video, etc.) is not unique but varies according to specific style formats. A structure editor should always contain predefined instances of structure according to general user needs (e.g. „article“, „database query“, „inventory list“ etc.) and may allow user defined instances of structure according to specific user needs. Today, the main technical base for definition of structure is an ISO standard, SGML (Standard Generalized Markup Language) which is a generic language, instances of which are called DTD's (Document Type Definitions). Thus a key concern to us is the definition of DTD's for mathematics.

mathPAD: In order to give some idea about this on the concrete level and how it helps the working mathematicians: How many licences for the system have you distributed? How widely is it accepted? What is your projection about its acceptance in the future?

Topsoe: The figure was 48 licences by the year and now it is 50 and something. In our database we have about 700 mathematical research centers. So, it is only some 50 out of those who are using it at present, which definitely is not enough to reach a critical mass, it is far short of it.

mathPAD: Will that change in the future?

Topsoe: Yes, I really think it will change after the final release which will be early in 1994, hopefully already in January.

mathPAD: That will then be the first non-development version?

Topsoe: Exactly. We expect that from that date on there will be some pointers. One of them will be the Congress in Zürich. There will be other promotional efforts already starting this autumn. Then it will catch on. We believe a modest goal will be to reach within a year one hundred research institutes (not individuals).

mathPAD: How many of your present subscribers are connected to the Euromath organization? Did the majority of subscriptions come from people in the Euromath committees?

Topsoe: Not the majority but a fair amount, every research centre which has sent colleagues in a formal capacity to committees of the board, altogether perhaps 10 subscriptions, or something like that.

mathPAD: What is the price for the Euromath system and on what machines it is available?

Topsoe: At present you get the development version on an institute basis for 1500 ECU including the right to update. It also includes the right to take up a cheap licence for access to the FIZ data base.

mathPAD: On what machines is the system available?

Topsoe: Well, at present only on Unix Workstations (from SUN, IBM, Digital, Hewlett Packard)[1]. Pretty soon after release time, we expect that it will be available on high-end PC's, we hope in 1994, perhaps the end of 1994.

mathPAD: Who did the work on the structure editor?

Topsoe: It originated from a PhD thesis in INRIA, but now it is developed by a commercial company near Paris: GRIF S.A.

mathPAD: Was this a major project? Give us a figure of how much money was spent on this project from European grants?

Topsoe: The input has been significant. Not only was it going to the commercial company GRIF but also to other places. The order of magnitude is about 2 Million ECU.

mathPAD: Since the Euromath system is developed by a company, who owns the copyright? Is it owned by the mathematical community or will it be a commercial product, maybe outside of control from Euromath?

Topsoe: We have paid great attention to this issue. When you do not develop everything from scratch yourself you have to rely on an existing product and you cannot claim rights to everything. We do not have complete control but we are quite fortunate in that with our key business partner GRIF S.A. we have a very clear agreement that they are not interested in commercializing their product in the academic sector. So, we have made agreements with them that they are active in the commercial sector and we only in the academic sector. I would not say it is really exclusive, but their products will only find a way to the academic sector either via us, at special terms, or in other ways at commercial rates which typically will be more expensive by a factor of 10 or 20.

mathPAD: Does the licence fee for the licences which you sell tomorrow go to EMT or to GRIF?

Topsoe: Well, we have to pay to GRIF. The way we try to structure it, is to get favourable terms, so we have entered into bulk deal arrangements with GRIF. This we hope will be of benefit to both parties. But that depends on a substantial number of subscriptions being taken out.

mathPAD: The reluctance in Germany with respect to Euromath may come from the fact that the system was announced a long time ago and then neither delivered nor available. How can such a miscalculation arise?

Topsoe: We have been quite desperate about this. All kinds of delays have occurred. We have not

[1] The development version of Euromath is running on the following platforms: DECstations (Ultrix 4.x), 9000/700 series of HP (HP/UX), IBM RS/6000 (AIX), and SUN Sparcstation (SunOS 4.x), port to other platforms, including SUN Sparcstations (SOLARIS) and PC (DOS/Windows) is planned

been good enough. I think myself, that we failed in providing a professional directorship for the project. Somehow we should have had more professional people.

mathPAD: Has that been changed?

Topsoe: Now the project is coming to an end, so it is not a problem. This had various effects, one of them being that Helmut Lenzing and his committee had to allocate much more time to these kind of issues than was reasonable. For some time we had an efficient directorship, but unfortunately the person involved, Alistair Chalmers, had to withdraw.

mathPAD: When Euromath will come to an end will the organization discontinue its operation?

Topsoe: Yes and no. GRIF will discontinue its work in the framework of the contract with the European Commission.

mathPAD: Are there plans for a new contract?

Topsoe: Yes, there are very strong plans. We are considering two kinds of activities. One is to meet the challenges of the opening in Central and Eastern Europe. There, sophisticated pieces of software, say the Euromath system, may not be the first thing to consider, so our activities will perhaps be of a wider scope and also attempt to enhance the computing and computer networking infrastructure in the academic environment. The other part is to consolidate the development and to promote it. We are confident that we are able to pick up a sufficient sum from the European Commission for promotional activities — at least during 1994.

mathPAD: Coming back to the ownership of the copyright, are precautions taken that if the Euromath organization is discontinued that then the benefit of the cooperation will be for the benefit of the user community?

Topsoe: All rights are owned by EMT, as far as we can own the rights. It is written into our statutes that we have certain obligations towards the Mathematical Community in Europe in the case that EMT is wound up.

mathPAD: For me, there is a slight confusion. I understood that the copyright is not owned by EMT?

Topsoe: Yes, I said, in as far as we can own the rights.

mathPAD: Aha!

Topsoe: When you enter Commission research and development work there is automatically a flow of rights to the partners.

mathPAD: What will be the future of the Euromath Bulletin?

Topsoe: We believe we will get the funding to continue its publication and distribution for all of 1994. We hope to be able not only to distribute the Bulletin to the Euromath subscribers but to the whole community, these 700 institutes and more.

mathPAD: Will there be one issue in 1993?

Topsoe: Well, that is not quite clear. The problem is the following: Because of delays of the Euromath project there was not a sufficient amount of articles reflecting user experience with the new version of the system, the coming version of Euromath. And we did not want to deliver the next issue of the Bulletin without such articles.

mathPAD: Is it a good strategy to start a journal with the first issue appearing in spring 1992 and then the next not before 1994?

Topsoe: Well, I guess it will appear in 1993. But anyhow, no it is not a good strategy.

mathPAD: You have invested a lot of emotional power, a lot of strength, a lot of work into this project, a project for achieving something which gives perspectives, with respect to acceptance of modern technology, to the mathematical community. Was it worth it? Has enough been achieved? Is there a sound basis to build on in the future?

Topsoe: I would answer yes. I thought it was going to be easier. But with respect to the importance to the community, I think that we have been right.

mathPAD: Is the awareness in the community on incorporating modern technology into the ivory tower better than it was 10 years ago?

Topsoe: Yes. When we started, very few mathematicians used e-mail, and very few used TeX.

mathPAD: Is that an achievement of Euromath?

Topsoe: No, it is not. Actually many delays in the Euromath project have been very, very negative and may actually cause that we will not achieve as easily, if at all, what we want to do. The fact that users now have certain habits using TeX in one form or another. And they will say „why Euromath when we have TeX?“. However, this is a complete misunderstanding because a structure driven editor is very different. But the fact is, that the users have acquired habits. We should have been on the market some 3 or 4 years ago, then the situation would have been quite different.

mathPAD: What are other results of Euromath?

Topsoe: There are quite a lot. You could say, we have behaved extremely well seen from he point of view of Brussels. They wanted us to travel around and to meet each other. This we have done, we have established collaboration with representatives of the community in now 23 countries. This is an achievement of great value. Whether or not this is as great a value as Brussels thinks, that is another question. Discussing all these issues, in particular on what we call the Euromath Advisory Board, has had a great positive effect. An awareness has been created for these kind of problems and many have prepared themselves for the future by getting national funding.

mathPAD: Any other results?

Topsoe: At the time when Euromath started, Eastern Europe was closed to us and to the benefits of EC funding. At a meeting in Prague in 1986, when the representatives on the European Mathematical Council from Central and Eastern Europe advised us not to hesitate and to go ahead, Michael Atiyah as chairman of European Mathematical Council said we will try and do what we can for our colleagues from Eastern Europe as soon as we would be given the opportunity. To this obligation we have lived up. When opportunities came we have seized them, and many of us have been involved in activities towards that. Now we are setting up activities on a much greater scale and we have actually got some funding for this.

mathPAD: So Euromath has helped the East and the West of Europe to grow together more rapidly?

Topsoe: I think, you can really say so, well, I hope so.

Bericht zur Computerschach Weltmeisterschaft, Madrid, November 1992

Rainer Feldmann, Peter Mysliwietz

Seit 1974 versetzt alle drei Jahre ein Ereignis die Computerschachszene in Aufregung: die „großen" Computerschachweltmeisterschaften. Im Gegensatz zur alljährlich stattfindenden Computerschachweltmeisterschaft für Microcomputer, die mit schöner Regelmäßigkeit von dem einen oder anderen kommerziell erhältlichen Schachprogramm gewonnen wird, ging der Titel des Weltmeisters aller Klassen bisher immer an nichtkommerzielle, zumeist an Universitäten entwickelte Schachprogramme. Dabei wurde oftmals das überlegene Schachwissen der kommerziellen Schachcomputer durch den Einsatz von Spezialhardware kompensiert. Im November 1992 traf sich dann die Weltelite zu einem neuerlichen Kräftemessen in Madrid.

Zu dieser Weltmeisterschaft fuhren auch die wissenschaftlichen Mitarbeiter der Arbeitsgruppe von Prof. Dr. Burkhard Monien, Rainer Feldmann und Peter Mysliwietz, in der Hoffnung auf eine gute Plazierung. Hatte sich ihr an der Universität GH Paderborn entwickeltes, paralleles Schachprogramm *ZUGZWANG* in einer 256 Prozessor Version doch sogar kommerziellen Schachcomputern in vielen Testspielen als mindestens ebenbürtig erwiesen. Als es zudem noch in Zusammenarbeit mit Mitarbeitern des PC^2 gelang, pünktlich vor Beginn der Computerschachweltmeisterschaften den GCel Parallelrechner mit 1024 T805 Prozessoren einsatzbereit zu machen, gehörte *ZUGZWANG* zum erweiterten Kreis der Favoriten.

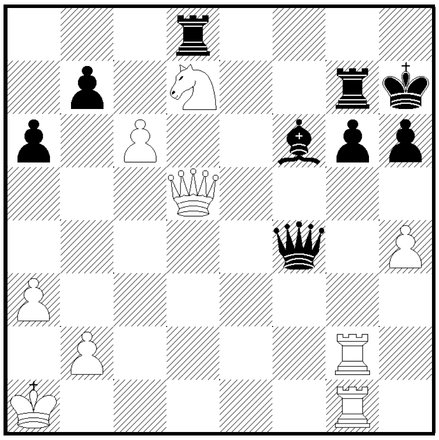
Einen eindeutigen Topfavoriten gab es für das nur auf fünf Runden angesetzte Turnier nicht, da der Sieger von 1989, *DEEP THOUGHT*, nicht anwesend war. Dementsprechend spannend gestaltete sich der Turnierverlauf. Betrachten wir den Weg des späteren Weltmeisters *CHESS MACHINE SCHRÖDER* und des späteren Vizeweltmeisters *ZUGZWANG* noch einmal:

In der ersten Runde wurden Schachprogramme aus dem gleichen „Haus" gegeneinander gepaart, um Spielabsprachen zwischen befreundeten Parteien in den letzten Runden zu vermeiden. Dementsprechend wurde *ZUGZWANG* gegen das zweite anwesende Paderborner Schachprogramm *ULYSSES* gesetzt, während Ed Schröders *CM SCHRÖDER* gegen den Mitfavoriten RISC 2500 antreten mußte. „Das Monster" *ZUGZWANG* (von den Gegnern so genannt wegen der mehrere Kubikmeter großen Hardware des GCel) kam gegen *ULYSSES* zu einem leichten Sieg, während sich die CM SCHRÖDER mit einem Remis begnügen mußte. In der zweiten und dritten Runde gewann *ZUGZWANG* gegen CUMULUS 2 bzw. KALLISTO.

Platz	Programm	Land
1.	*CHESS MACHINE SCHRÖDER*	(NL)
2.	*ZUGZWANG*	(D)
3.	*CUMULUS 2*	(F)
4.	*KASPAROV SPARC*	(USA)
5.	*FRITZ 2*	(NL)
6.	*RISC 2500*	(NL)
7.	*HITECH B**	(USA)
8.	*CHESS GENIUS*	(GB)
9.	*WOODPUSHER*	(GB)
10.	*M-CHESS PRO*	(USA)
11.	*HIARCS 6.88*	(GB)
12.	*LACHEX*	(USA)
13.	*PANDIX*	(H)
14.	*KALLISTO*	(USA)
15.	*ULYSSES*	(D)
16.	*PROCHESS*	(NL)
17.	*CENTAUR*	(RUS)
18.	*NIMZO-GUERNICA*	(A)
19.	*ECHEC*	(F)
20.	*NIGHTMARE*	(D)
21.	*MIRAGE*	(RUS)
22.	*DELICATE BRUTE*	(GB)

Abschlußtabelle der Weltmeisterschaft

Die *CM SCHRÖDER* konnte sich gegen die amerikanischen Konkurrenten *HITECH B** und *M-CHESS PRO* durchsetzen.
Jetzt gab es nur noch zwei verlustpunktfreie Programme: *ZUGZWANG* und die amerikanische Schachmaschine *KASPAROV SPARC*, die dementsprechend in der vierten Runde gegeneinander antreten mußten. Die *CHESS MACHINE* bekam es mit einem neuen, alten Konkurrenten zu tun: dem neuen Programm *CHESS GENIUS* des ewigen Widersachers Richard Lang.
Ein Sieg von *ZUGZWANG* bei einem gleichzeiten Punktverlust der *CM SCHRÖDER* hätte die Weltmeisterschaft zugunsten des Paderborner Programms entschieden. Umgekehrt hatte auch ein Punktverlust von *ZUGZWANG* bei einem gleichzeitigen Sieg der *CM SCHRÖDER* vorentscheidenden Charakter, da Schröders Programm im Vergleich zu *ZUGZWANG* bis daher die stärkeren Gegner gehabt hatte.
Die Spannung des zahlreich anwesenden, fachkundigen Publikums erreichte einen Höhepunkt, als in der vierten Runde die folgenden Stellungen erreicht wurden:



ZUGZWANG - KASPAROV SPARC
4. Runde, Weiß am Zug

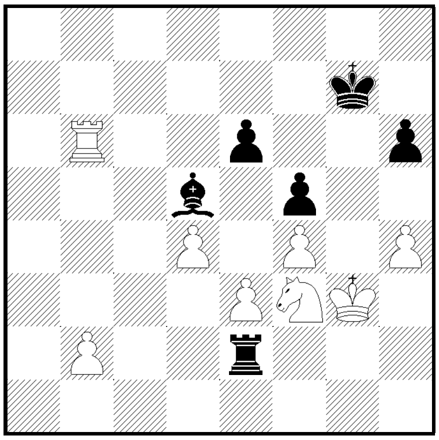
Hier steht Weiß klar besser und scheint auf dem Weg zum Titelgewinn, da die *CM SCHRÖDER* am Brett nebenan auf Verlust steht. Es folgt

43. ♖xg6,

was vernichtend aussieht, aber vielleicht den Titel verspielt hat:

43. ... ♗xb2+!
44. ♔xb2 ♖xg6
45. ♖xg6 ♕f2+

und hier hat Schwarz Dauerschach. Das Spiel endete wenig später mit Remis. Damit hat *ZUGZWANG* unnötigerweise einen halben Punkt abgegeben.



CHESS GENIUS - CM SCHRÖDER
4. Runde, Weiß am Zug

Ed Schröder wirkte hypernervös, denn diese Stellung ist klar verloren für Schwarz. Allerdings zeigt sich *CHESS GENIUS* auf einmal freigebig:

56. ♘e5 h5 57. e4 fxe4 58. f5 exf5

Auch im folgenden zeigt Weiß Endspielschwäche: 59. ♖g6+ ♔h7 60. ♖g5 ♗e6 61. ♖xh5+ ♔g7 62. ♖g5+ ♔f6 63. b4 ♖e3+ 64. ♔f4 ♖h3 65. ♖g6+ ♔e7 66. ♖h6 e3 67. ♘c6+ ♔d7 68. ♘e5+ ♔e7 69. ♘g6+ ♔d8 70. ♖h8+ ♔c7 71. ♖e8 ♔d6 72. ♖a8 ♗c4 73. ♘e5 ♗b5 74. ♘f7+ ♔e7 75. ♘h6 ♖xh4+ 76. ♔g5 ♖xh6 77. ♔h6 e2 78. ♖a5 e1♕ 79. ♖xb5 0:1
und die Partie endete zur Überraschung aller mit einem Sieg für Schröders *CHESS MACHINE*.

In der fünften Runde gewann die *CHESS MACHINE SCHRÖDER* gegen *KASPAROV SPARC*. Damit war der Ausgang der Partie von *FRITZ 2* gegen *ZUGZWANG* nur noch Formsache. Diese Partie endete Remis. Damit belegte *ZUGZWANG* einen guten zweiten Platz nach der letzten Runde, ein Resultat, mit dem die Paderborner sehr zufrieden waren. Zum ersten Mal gewann mit der *CHESS MACHINE SCHRÖDER* ein auf einem PC laufendes, kommerziell vertriebenes Programm die „großen“ Computerschachweltmeisterschaften.
Es stellt sich die Frage, ob die kommerziellen Programme diesen Erfolg bei der nächsten Weltmeisterschaft verteidigen können. Sicherlich sind

die PCs besonders in den letzten Jahren deutlich leistungsfähiger geworden. Aber man darf auch die Leistung des Programmierers der *CHESS MACHINE SCHRÖDER* nicht geringschätzen: Ed Schröder ist seit vielen Jahren bekannt als Programmierer von Schachcomputern der Spitzenklasse. Bei den Weltmeisterschaften im Jahre 1986 verpaßte er ganz knapp den Titel, weil in der letzten Runde sein Prozessor überhitzte. Fazit: Nicht nur die PCs, auch die Kühlungen sind leistungsfähiger geworden.

Kalender

SODA '94

SODA '94, fifth annual ACM-SIAM Symposium on discrete Algorithms, 23.–25. Januar 1994, Arlington, Virginia, USA

Kontakt:

Tel: +1–215–382–9800
Fax: +1–215–386–7999
E-mail: `meetings@siam.org`

INTERVAL '94

International Conference on Interval and Computer Algebraic Methods in Science and Engineering, 7.–10. März 1994, St. Petersburg, Russland

Kontakt:

Dr. V. M. Nesterov
Box 52
St.Petersburg, 195256, Russia
E-mail: `nest@nit.spb.su`

THE RHINE WORKSHOP ON COMPUTER ALGEBRA

The Rhine Workshop on Computer Algebra, 22.–24. März 1994, Karlsruhe

Kontakt:

Prof. Jacques Calmet
Universität Karlsruhe
Am Fasanengarten 5
D–76128 Karlsruhe
Tel: +49–721–608–4208
Fax: +49–721–696–893
E-mail: `calmet@dkauni2.bitnet`

GAMM-Jahrestagung

GAMM-Jahrestagung, 4.–8. April 1994, Braunschweig

Kontakt:

Prof. Dr. F. Goerisch
Institut für Angewandte Mathematik
Technische Universität Braunschweig
Postfach 3329
D–38023 Braunschweig
Tel: +49–531–391–7536
Fax: +49–531–391–8206
E-mail: `i1040108@ws.rz.tu-bs.de`

MEGA '94

Third International Symposium on Effective Methods in Algebraic Geometry, MEGA '94, 5.–9. April 1994, Santander, Spanien

Kontakt:

Prof. L. González-Vega
Departamento de Matemáticas
Facultad de Ciencias
Universidad de Cantabria
Santander 39071, Spain
Tel: +34–42–201437
Fax: +34–42–201402
E-mail: `mega94@ccucvx.unican.es`

DERIVE Conference

1st International DERIVE Conference, 11.–15. July 1994, Plymouth, UK

Arbeiten sind einzureichen bis:

31. Januar 1994

Kontakt:

Prof. John Berry
Centre for Teaching Mathematics
University of Plymouth
Drake Circus
Plymouth, Devon PL4 8AA, UK

14th. IMACS

14th. World Congress of the International Association for Mathematics and Computers in Simulation (IMACS), 11.–15. July 1994, Atlanta, USA

Arbeiten sind einzureichen bis:

15. Oktober 1993

Kontakt:

Dr. Thiab Taha
Computer Science Department
University of Georgia
Athens, GA 30602-7404, USA
E-mail: `thiab@pollux.cs.uga.edu`

CONPAR '94 / VAPP VI

International Conference on Parallel and Vector Processing: CONPAR '94 / VAPP VI, 6.–8. September 1994, Linz, Östereich

Kontakt:

Wolfgang Schreiner
RISC-Linz
Johannes-Kepler Universität
A–4040 Linz, Östereich
Tel.: +43–7236–3231–66
Fax: +43–7236–3231–30
E-mail:
`conpar94@risc.uni-linz.ac.at`

DMV Jahrestagung

DMV Jahrestagung, 19.–23. September 1994, Duisburg

Aufgelesen

Computerwahn

Krieg oder Ente?

In der Gemeinschaft Unabhängiger Staaten soll ein Computer über den Start atomarer Interkontinentalraketen entscheiden können... Das System „Tote Hand“ prüfe im Ernstfall automatisch, ob aus dem Hauptquartier noch „vernünftige Befehle“ kommen. Der Rechner sei zudem mit einer Reihe von Sensoren verbunden, die auf den Einschlag feindlicher Kernwaffen reagieren, um dann den Befehl zum Einsatz eigener Atomwaffen zu geben.

Die Computer Zeitung 24/42, 21.10.1993

Telekom

Breitbandnetz

Jetzt hat die Telekom das Aus für das Vermittelnde Breitbandnetz (VBN) beschlossen. „Wir bieten keine Anschlüsse für Neukunden mehr an“ bestätigt Wilfried Seibel von der Telekom... Das ist das Ende für das weltweit einmalige Netz mit einer Bandbreite von 140 Megabit pro Sekunde... Mittlerweile sind sogar die meisten Sportstadien mit einem VBN-Netz ausgestattet. Bei der Bewerbung für Olympia 2000 hatte der Berliner Regierende Bürgermeister das schnelle Netz noch als Argument für die Vergabe der Spiele in die Hauptstadt angeführt.
Aus jedem zweiten Politiker-Interview hechelte dem Bundesbürger während der letzten Jahre die Parole entgegen, Deutschland müsse in Sachen High-Tech den Anschluß an die Weltspitze wieder schaffen. Und dann wird beim Hochgeschwindigkeitsnetz ein Technologievorsprung von fünf Jahren... einfach verschenkt... Fragt sich nur welche Ziele der Bundespostminister mit dieser Industriepolitik verfolgt.

Die Computer Zeitung 24/42, 21.10.1993

Wissenschaftspolitik

All change at the top?

Suddenly science politics has undergone a transformation. The US now has a collection of new faces in key positions, while Italy, Germany and France have new science ministers. Unfortunately these appointees are a decidedly mixed bunch when it comes to technical experience and demonstrable interest in R&D. On balance the prospects are not exciting... The transition hits the BMFT at a time when leadership is badly needed.

PHYSICS world 6/6, June 1993

Multimedia

Kein Abschied von Gutenberg

prophezeit Florian Langenscheidt... „Im Jahre 2000 wird das durchschnittliche Verlagshaus einen Anteil an Electronic-Publishing-Produkten von rund 25 Prozent haben.“... Eine Verdrängung der herkömmlichen Printmedien befürchtet (der Marktforscher Thomas Leukamm) jedoch nicht: „Dank ihrer komplementären Eigenschaften ermöglichen die elektronischen Produkte den Verlegern neue Zusatzgeschäfte, oft sogar auf der Basis vorhandener Verlagssubstanzen“.

Die Computer Zeitung 24/43, 28.10.1993

Glitzy!

Multimedia is used as a modern name for interactive video... It facilitates glitzy presentations of boring data.

Computers in Physics 1/93, Januar 1993

Parallelrechnen

Durchbruch?

Im Institut für Methodische Grundlagen der GMD wurde unter Leitung von Prof. Dr. Ulrich Trottenberg soeben ein erster Durchbruch auf dem Gebiet der Parallelisierung erreicht: Erstmalig gelang es ein meteorologisches Produktionsprogramm auf einer Vielzahl von Parallelrechnern einzusetzen.

Der GMD-Spiegel 2/93,

At best?

The situation today is that clusters excel at delivering capacity, due to a large number of simultaneous users. Almost all massively parallel systems excel (at best) in delivering capability, due to a small number of simultaneous users. Vector supercomputers are actually excellent in both aspects.

Computers in Physics 2/93, März 1993

Hersteller

Marktführer!

Nach meinem Eindruck laufen wir allmählich mit der Firma Sun in Probleme: Sun entfernt sich in zunehmendem Maße von den Vorteilen, die diese Firma in den Gründertagen an den Tag gelegt hat und die wir gegenüber den Mitbewerbern von Sun zu schätzen wußten.

Leserbrief in SUG info 4/93,

Wabi

Ab November will Sunsoft die Verbindungslösung Wabi 1.0 (Windows Application Binary Interface) zusammen mit dem Betriebssystem Solaris ausliefern. Wabi soll ermöglichen, Windows-Applikationen unverändert auf der Unix Plattform laufen zu lassen.

Die Computer Zeitung 24/43, 28.10.1993

Gewinneinbruch

Der Preiskrieg im Markt für Personal Computer hat nun auch die kalifornische Apple Computer Inc. erreicht. Im Geschäftsjahr 92/93 mußte das Unternehmen einen Gewinneinbruch von 84 Prozent hinnehmen.

Die Computer Zeitung 24/43, 28.10.1993

Sculley steigt bei Apple endgültig aus.

Die Computer Zeitung 24/43, 28.10.1993

Das Gehalt von Scott McNealy, Chef von Sun Microsystems, wurde auf seinen eigenen Wunsch um ein Drittel gekürzt. Dafür verdoppelte (man) seine Kaufoption für Sun Aktien, die aber beim derzeitigen Kurs wertlos ist, seit Sun einen Gewinnrückgang bekanntgeben mußte.

Die Computer Zeitung 24/43, 28.10.1993

Extravaganza!

The Quadra 840AV is a big step in Mac history. Besides being the fastest Mac, new circuitry in the form of a digital signal processor allows fast processing of video and audio signals... An affordable audiovideo extravaganza.

MAC HOME journal 1/7, September 1993