mathPAD

Vol 6 No 1, September 1996

In diesem Heft

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

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

Liebe Leserin, lieber Leser,

wieder ist fast ein ganzes Jahr seit dem Erscheinen der letzten mathPAD verstrichen, obwohl wir uns fest vorgenommen hatten, für die Publikation den den Halbjahresrhythmus einzuhalten. Unsere Ausrede ist dieselbe wie in der Vergangenheit: Die Arbeit an MuPAD, der Support für unsere inzwischen etwa 30000 Nutzer und die übliche Routinearbeit in einem Hochschulinstitut haben uns so ausgiebig beschäftigt, daß für die Publikation der mathPAD nur wenig Zeit blieb.

In bezug auf unsere MuPAD-Arbeit waren die entscheidenden Meilensteine das Beta-Release der Version 1.3 (genannt 1.2.9), die CeBIT 96 sowie das Beta-Release von MuPAD für Windows 95, die sich inzwischen einer beachtlichen Nutzergemeinde erfreut. Ansonsten verweisen wir für Systeminformationen über MuPAD auf unsere inzwischen noch umfangreicher gewordenen Web-Seiten.

Bezüglich des vorliegenden Heftes möchten wir unseren Autoren einen ganz herzlichen Dank aussprechen.

Mit freundlichen Grüßen

B.F. & K.D.

Wieviele Rinder hat der Sonnengott?

Friedrich Schwarz
Universität Paderborn, fritz@uni-paderborn.de

An dem berühmten Rinderproblem von Archimedes kann man zeigen, wie gut MuPAD dank der PARI-Arithmetik mit wirklich großen ganzen Zahlen rechnen kann.

Im Jahr 1773 veröffentlichte G. E. Lessing, der damals Bibliothekar an der Herzoglich-Braunschweigischen Bibliothek in Wolfenbüttel war, einen griechischen Text aus einem Codex, den er in seiner Bibliothek gefunden hatte (vgl. [1]). Dieser Text enthält eine Aufgabe, die, wie in ihrem Vorspann vermerkt ist, einst Archimedes an Eratosthenes und die anderen Mathematiker in Alexandria geschickt hat. Eine philologisch zuverlässige Version des griechischen Texts und eine Übersetzung ins Lateinische findet man im zweiten Band der von I. L. Heiberg besorgten Ausgabe der Werke von Archimedes (vgl. [2]).

Die Aufgabe

Die Rinderherde des Gottes Helios besteht aus N_1 weißen, N_2 schwarzen, N_3 gescheckten und N_4 braunen Stieren und aus N_5 weißen, N_6 schwarzen, N_7 gescheckten und N_8 braunen Kühen.

(a) Es gilt

$$\left.\begin{aligned} N_1 &= \left(\frac{1}{2}+\frac{1}{3}\right) N_2 + N_4, \\ N_2 &= \left(\frac{1}{4}+\frac{1}{5}\right) N_3 + N_4, \\ N_3 &= \left(\frac{1}{6}+\frac{1}{7}\right) N_1 + N_4 \end{aligned}\right\} \quad \text{und} \quad \left\{\begin{aligned} N_5 &= \left(\frac{1}{3}+\frac{1}{4}\right)(N_2+N_6), \\ N_6 &= \left(\frac{1}{4}+\frac{1}{5}\right)(N_3+N_7), \\ N_7 &= \left(\frac{1}{5}+\frac{1}{6}\right)(N_4+N_8), \\ N_8 &= \left(\frac{1}{6}+\frac{1}{7}\right)(N_1+N_5). \end{aligned}\right.$$

(b) Die weißen und die schwarzen Stiere zusammen können sich in Form eines Quadrats aufstellen, d.h. $N_1 + N_2$ ist eine Quadratzahl.

(c) Die gescheckten und die braunen Stiere zusammen können sich in Form eines Dreiecks aufstellen, d.h. es gibt eine natürliche Zahl m mit

$$N_3 + N_4 = \frac{1}{2}\, m(m+1).$$

Zu ermitteln ist die Anzahl der Rinder in der Herde. Es dürfte reichen, die kleinste Lösung zu finden – falls es überhaupt Lösungen gibt. Gesucht ist also ein 8-tupel $(N_1, N_2, \ldots, N_8) \in \mathbb{N}^8$, das die Bedingungen (a), (b) und (c) erfüllt und das aus möglichst kleinen natürlichen Zahlen besteht.

Der Lösung erster Teil

(A) Man löst zuerst das lineare Gleichungssystem aus (a), was für *MuPAD* keine Schwierigkeit bedeutet. Man erhält: Mit den Zahlen

$$\begin{array}{llllllll} a_1 & := 103\,66482, & a_2 & := 74\,60514, & a_3 & := 73\,58060, & a_4 & := 41\,49387, \\ a_5 & := 72\,06360, & a_6 & := 48\,93246, & a_7 & := 35\,15820, & a_8 & := 54\,39213 \end{array}$$

ist die Menge der 8-tupel $(N_1, N_2, \ldots, N_8) \in \mathbb{N}^8$, die den sieben Bedingungen in (a) genügen,

$$\{(a_1 v,\ a_2 v,\ a_3 v,\ a_4 v,\ a_5 v,\ a_6 v,\ a_7 v,\ a_8 v) \mid v \in \mathbb{N}\}.$$

(B) Für $v \in \mathbb{N}$ erfüllt $(a_1 v, a_2 v, a_3 v, a_4 v, a_5 v, a_6 v, a_7 v, a_8 v)$ die Bedingung in (b), genau wenn $(a_1 + a_2) \cdot v$ eine Quadratzahl ist. Es ist $a_1 + a_2 = 178\,26996 = 2^2 \cdot 3 \cdot 11 \cdot 29 \cdot 4657$, und daher ist $(a_1 + a_2) \cdot v$ eine Quadratzahl, genau wenn mit $b := 3 \cdot 11 \cdot 29 \cdot 4657 = 44\,56749$ gilt: Es gibt ein $w \in \mathbb{N}$ mit $v = b \cdot w^2$.

(C) Es ist $a_3 + a_4 = 115\,07447 = 7 \cdot 353 \cdot 4657$. Für $w \in \mathbb{N}$ erfüllt das 8-tupel

$$(a_1 bw^2,\ a_2 bw^2,\ a_3 bw^2,\ a_4 bw^2,\ a_5 bw^2,\ a_6 bw^2,\ a_7 bw^2,\ a_8 bw^2)$$

genau dann die Bedingung aus (c), wenn es eine natürliche Zahl m mit

$$\frac{1}{2} m(m+1) \ = \ a_3 bw^2 + a_4 bw^2 \ = \ 3 \cdot 7 \cdot 11 \cdot 29 \cdot 353 \cdot 4657^2 \cdot w^2 \ = \ 5128\,58029\,09803 \cdot w^2$$

gibt. Es ist also die Lösung $(m, w) \in \mathbb{N} \times \mathbb{N}$ der Diophantischen Gleichung

$$\frac{1}{2} M(M+1) \ = \ 5128\,58029\,09803 \cdot W^2$$

zu finden, in der die Zahlen m und w möglichst klein sind – falls es überhaupt Lösungen aus natürlichen Zahlen gibt. Dazu ermittelt man, falls vorhanden, die Lösung $(x, w) \in \mathbb{N} \times \mathbb{N}$ der Gleichung

$$X^2 - 8 \cdot 5128\,58029\,09803 \cdot W^2 \ = \ 1, \tag{1}$$

in der x und w möglichst klein sind, sieht, daß darin x ungerade und größer als 1 ist, und setzt

$$m \ := \ \frac{1}{2}(x-1).$$

Die Pellsche Gleichung

Für Diophantische Gleichungen vom Typ der Gleichung (1) gibt es eine vollständige Theorie, die im wesentlichen auf Fermat, Euler und Lagrange zurückgeht, jedenfalls was die europäische Mathematikgeschichte betrifft. Euler nannte diese Gleichungen nach dem englischen Mathematiker John Pell, der sich wohl nicht weiter mit ihnen befaßt hat.

Es sei $d \in \mathbb{N}$ keine Quadratzahl. Die Pellsche Gleichung

$$X^2 - d \cdot Y^2 \ = \ 1 \tag{PELL}$$

besitzt unendlich viele Lösungen $(x, y) \in \mathbb{N} \times \mathbb{N}$. Es genügt, die Lösung $(x, y) \in \mathbb{N} \times \mathbb{N}$ von (PELL) mit minimalem y und daher auch minimalem x, die sogenannte Fundamentallösung von (PELL), zu finden. Definiert man nämlich für jedes $n \in \mathbb{N}$ die natürliche Zahlen x_n und y_n durch

$$x_n + y_n \sqrt{d} \ = \ \left(x + y\sqrt{d}\right)^n,$$

so ist $\{(x_n, y_n) \mid n \in \mathbb{N}\}$ die Menge aller Lösungen $(x, y) \in \mathbb{N} \times \mathbb{N}$ von (PELL).

Einen brauchbaren Algorithmus zur Berechnung der Fundamentallösung von (PELL) bekommt man mit Hilfe der Theorie der Kettenbrüche. Man berechnet den regelmäßigen Kettenbruch

$$\sqrt{d} \;=\; [\, a_0, a_1, a_2, \ldots, a_k, a_{k+1}, \ldots \,] \;=\; a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{a_3 + \cdots}}}$$

für $\sqrt{d}$. Dieser Kettenbruch ist periodisch mit einer Vorperiode der Länge 1, es gilt also, wenn l die Länge einer minimalen Periode ist,

$$\sqrt{d} \;=\; [\, a_0, \overline{a_1, a_2, \ldots, a_{l-1}, a_l}\,],$$

und die Periode $(a_1, a_2, \ldots, a_{l-1}, a_l)$ besitzt die folgende Symmetrie-Eigenschaft: Es ist $a_l = 2a_0$, und für jedes $i \in \{1, 2, \ldots, l-1\}$ ist $a_i = a_{l-i}$. Ein vergleichsweise schnelles Verfahren zur Berechnung der Periode des Kettenbruchs für $\sqrt{d}$, das diese Symmetrie-Eigenschaft ausnützt, wird in Perrons klassischer Darstellung der Theorie der Kettenbrüche beschrieben (vgl. [3]); in [4] sind die Verfahren zur Berechnung periodischer Kettenbrüche so beschrieben, daß sie leicht in *MuPAD*-Programme übersetzt werden können. Die Fundamentallösung (x, y) von (PELL) erhält man dann so (vgl. dazu zum Beispiel [4]):

Man berechnet den $(l-1)$-ten Näherungsbruch r_{l-1}/s_{l-1} des Kettenbruchs für $\sqrt{d}$ und setzt

$$\begin{array}{lllllll} x & := & r_{l-1} & \text{und} & y & := & s_{l-1}, \quad \text{falls } l \text{ gerade ist,} \\ x & := & r_{l-1}^2 + d s_{l-1}^2 & \text{und} & y & := & 2 r_{l-1} s_{l-1}, \quad \text{falls } l \text{ ungerade ist.} \end{array}$$

Zur Berechnung von (x, y) benötigt man somit die Näherungsbrüche r_0/s_0, $r_1/s_1, \ldots, r_{l-1}/s_{l-1}$ des Kettenbruchs

$$\sqrt{d} \;=\; [\, a_0, a_1, a_2, \ldots, a_k, a_{k+1}, \ldots \,].$$

Deren Zähler und Nenner berechnet man rekursiv: Man setzt

$$(2) \qquad \begin{cases} r_{-2} := 0, \quad r_{-1} := 1 \quad \text{und} \quad r_k := a_k r_{k-1} + r_{k-2} \quad \text{für jedes } k \in \mathbb{N}_0, \\ s_{-2} := 1, \quad s_{-1} := 0 \quad \text{und} \quad s_k := a_k s_{k-1} + s_{k-2} \quad \text{für jedes } k \in \mathbb{N}_0. \end{cases}$$

Diese klassischen Ergebnisse hat – etwa 100 Jahre nach Lagrange – A. Amthor in [5] verfeinert. Er zeigte: Ist p eine ungerade Primzahl, die d nicht teilt, so ist die kleinste natürliche Zahl n, für die in der Lösung (x_n, y_n) von (PELL) die Zahl y_n durch p teilbar ist, ein Teiler von $p-1$, falls d ein quadratischer Rest modulo p ist, bzw. ein Teiler von $p+1$, falls d ein quadratischer Nichtrest modulo p ist.

Der Lösung zweiter Teil

Wie bereits gezeigt wurde, ist zur Berechnung der Lösung der Aufgabe von Archimedes die Fundamentallösung der Pellschen Gleichung

$$(1) \qquad X^2 - 8 \cdot 5128\,58029\,09803 \cdot W^2 \;=\; 1$$

zu ermitteln, und wie das zu tun ist, wurde eben beschrieben. Aber *MuPAD* scheint nicht in der Lage zu sein, die Fundamentallösung dieser Pellschen Gleichung direkt zu berechnen. Man kommt mit folgendem Trick weiter, den Amthor in [5] verwendet: Es ist

$$8 \cdot 5128\,58029\,09803 \;=\; 2^3 \cdot 3 \cdot 7 \cdot 11 \cdot 29 \cdot 353 \cdot (2 \cdot 4657)^2 \;=\; 47\,29494 \cdot (2 \cdot 4657)^2.$$

Statt die Fundamentallösung (x, w) von (1) direkt zu berechnen, ermittelt man zuerst die Fundamentallösung (x, y) der Pellschen Gleichung

$$(3) \qquad X^2 - 47\,29494 \cdot Y^2 \;=\; 1.$$

Dazu berechnet man mittels *MuPAD* den periodischen Kettenbruch für $\sqrt{47\,29494}$. Seine Vorperiode besteht aus der einen Zahl 2174, und seine Periode ist

$$\begin{aligned}&(1, 2, 1, 5, 2, 25, 3, 1, 1, 1, 1, 1, 1, 15, 1, 2, 16, 1, 2, 1, 1, 8, 6, 1, 21, 1, 1, 3, 1, 1, 1, 2, 2, 6,\\ &\qquad 1, 1, 5, 1, 17, 1, 1, 47, 3, 1, 1, \mathbf{6}, 1, 1, 3, 47, 1, 1, 17, 1, 5, 1, 1,\\ &\qquad\qquad 6, 2, 2, 1, 1, 1, 3, 1, 1, 21, 1, 6, 8, 1, 1, 2, 1, 16, 2, 1, 15, 1, 1, 1, 1, 1, 1, 3, 25, 2, 5, 1, 2, 1, 4348).\end{aligned}$$

(Die fettgedruckte 6 markiert die Mitte der Periode). Die Periode dieses Kettenbruchs hat die Länge 92, also sind Zähler $x := r_{91}$ und der Nenner $y := s_{91}$ seines 91-ten Näherungsbruchs zu berechnen. Mit Hilfe von *MuPAD* ergibt sich aus (2)

$$\begin{aligned} x &= 10993\,19867\,32829\,73497\,98662\,32821\,43354\,39010\,88049 \quad \text{und}\\ y &= 5\,05494\,85234\,31503\,30744\,77819\,73554\,04089\,86340.\end{aligned}$$

Damit ist die Fundamentallösung (x, y) der Pellschen Gleichung (3) berechnet. Die Menge aller Lösungen aus natürlichen Zahlen von (3) ist, wie bereits vermerkt, $\{(x_n, y_n) \mid n \in \mathbb{N}\}$, wobei für jedes $n \in \mathbb{N}$

$$(4) \qquad x_n + y_n\sqrt{47\,29494} \;=\; \left(x + y\sqrt{47\,29494}\right)^n$$

gilt. Ist n die kleinste natürliche Zahl, für die y_n durch $2 \cdot 4657$ teilbar ist und ist $w := y_n/(2 \cdot 4657)$, so ist (x_n, w) offensichtlich die Fundamentallösung der Pellschen Gleichung (1).

Mit Hilfe der *MuPAD*-Funktion `isprime` zeigt man, daß 4657 eine Primzahl ist, und mit Hilfe der *MuPAD*-Funktion `numlib::jacobi`, daß $47\,29494$ ein quadratischer Nichtrest modulo 4657 ist. Daher ist nach dem Ergebnis von Amthor die kleinste natürliche Zahl n, für die y_n durch 4657 teilbar ist, einer der Teiler von $4657 + 1 = 4658 = 2 \cdot 17 \cdot 137$, also eine der Zahlen 1, 2, 17, 34, 137, 274, 548, 2329, 4658. (Man verwendet `numlib::divisors`). Mit Hilfe von `mod` zeigt man, daß $y_1 = y$ nicht durch 4657 teilbar ist. Aus (4) ergibt sich, daß für alle $m, n \in \mathbb{N}$

$$x_{m+n} \;=\; x_m x_n + 47\,29494\, y_m y_n \quad \text{und} \quad y_{m+n} \;=\; x_m y_n + x_n y_m$$

gilt, also für jedes $n \in \mathbb{N}$

$$x_{2n} \;=\; x_n^2 + 47\,29494\, y_n^2 \;=\; 2x_n^2 - 1 \quad \text{und} \quad y_{2n} \;=\; 2x_n y_n$$

und

$$x_{n+1} \;=\; x x_n + 47\,29494\, y y_n \quad \text{und} \quad y_{n+1} \;=\; x y_n + y x_n$$

gilt, und damit berechnet man der Reihe nach x_n und y_n für

$$n \;=\; 2,\ 4,\ 8,\ 16,\ 17,\ 34,\ 68,\ 136,\ 137,\ 274,\ 548,\ 1096,\ 2192,$$

stellt für $n = 2, 17, 34, 137, 274$ und 548 fest, daß y_n nicht durch 4657 teilbar ist, und berechnet schließlich

$$y_{2329} \;=\; y_{137+2192} \;=\; x_{137}\, y_{2192} + x_{2192}\, y_{137};$$

die Berechnung von x_{2329} ist nicht nötig. y_{2329} ist gerade und durch 4657 teilbar, und daher gilt für $w := y_{2329}/(2 \cdot 4657)$: (x_{2329}, w) ist die Fundamentallösung der Pellschen Gleichung (1). Man berechnet w^2 und mit den anfangs angegebenen Zahlen $a_1, a_2, \ldots, a_8$ und b für jedes $i \in \{1, 2, \ldots, 8\}$ die Zahl $N_i := a_i b w^2$. Dann ist $(N_1, N_2, \ldots, N_8)$ ein 8-tupel natürlicher Zahlen, das den Bedingungen (a), (b) und (c) der Aufgabe genügt, und es ist unter allen 8-tupeln, die dies tun, das mit minimalen Einträgen.

Die Herde des Sonnengottes besteht also aus

$$\begin{aligned} N \quad &:= \quad (a_1 + a_2 + \cdots + a_8) \cdot bw^2 \;=\; 7.76027\,14064 \ldots \cdot 10^{206544} \;= \\ &= \quad 77602\,71406\,48681\,82695\,30232\,83321 \;\ldots\; 88311\,89737\,23406\,62671\,94550\,81800 \end{aligned}$$

Rindern, ist also doch wohl ziemlich groß: Die Zahl N hat $2\,06545$ Stellen. Und doch kann *MuPAD* sie ohne Probleme berechnen.

Zur Geschichte

Der von Lessing 1773 veröffentlichte Text enthält eine aus der Antike stammende Lösung, die aber nicht korrekt ist; die darin angegebenen Anzahlen $N_1, N_2, \ldots, N_8$ erfüllen nur die Bedingungen in (a) (mit den hier verwendeten Bezeichnungen ist $N_i = 4a_i$ für jedes $i \in \{1, 2, \ldots, 8\}$), aber weder die Bedingung in (b), noch die in (c). Lessings Veröffentlichung enthält auch einige Bemerkungen eines Wolfenbütteler Mathematiklehrers Chr. Leiste, der zeigte, wie man alle Lösungen aus $\mathbb{N}^8$ des linearen Gleichungssystems in (a) berechnen und darunter die ermitteln kann, die auch der Bedingung in (b) genügen. Einen vollständigen Weg zur Lösung der Aufgabe veröffentlichte erstmals A. Amthor im zweiten Teil von [5] (diesem Weg folgt die vorliegende Darstellung); er gab die ersten vier Stellen und die Stellenzahl der Zahl N korrekt an. Im Jahr 1889 berechnete A. H. Bell, zusammen mit einem dazu gegründeten mathematischen Stammtisch, in vier Jahren die ersten 28 Stellen (davon die beiden letzten nicht korrekt) und die letzten 12 Stellen von N. Zum ersten Mal wurde N wohl im Jahr 1965 von H. C. Williams, R. A. German und C. R. Zarnke vollständig berechnet; sie benötigten dazu auf dem damaligen Computer der Universität von Waterloo 7 Stunden und 49 Minuten (vgl. [6]). Im Jahr 1981 wurde N erstmals veröffentlicht: H. L. Nelson berechnete N und fünf weitere, größere Lösungen der Aufgabe auf einem CRAY-1-Rechner, der dazu zehn Minuten brauchte, und druckte in [7] alle $2\,06545$ Stellen von N auf nicht ganz zwölf Druckseiten ab. Heute kann jeder, der *MuPAD* zu Verfügung hat, die Zahl N berechnen; wie lange sein Rechner dazu benötigt, mag jeder Leser selbst feststellen. So ist die alte Aufgabe von Archimedes auch eine Illustration zur Entwicklung des Scientific Computing geworden.

Die Aufgabe hat seit ihrer Veröffentlichung durch Lessing zu mancherlei Diskussionen geführt. Nach Meinung von B. Krumbiegel, der im ersten Teil von [5] neben einer Übersetzung ins Deutsche eine eingehende historische und philologische Untersuchung des Texts veröffentlicht hat, ist es “möglich, ja wahrscheinlich”, daß die Aufgabe wirklich von Archimedes stammt. Auch Heiberg schreibt in [2], daß es keinen Grund gibt, die Aufgabe nicht Archimedes zuzuschreiben. Der Wortlaut der Aufgabenstellung wurde im 19. Jahrhundert immer wieder diskutiert, wovon Krumbiegel im ersten Teil von [5] berichtet. So dachte man etwa darüber nach, wie die Bedingung (b) wirklich zu interpretieren sei, wo doch ein Rind im allgemeinen länger als breit ist.

Ob aber Archimedes oder einer seiner Zeitgenossen die Aufgabe gelöst hat, ist eine andere Frage; niemand kann sie nach den heutigen Kenntnissen über Archimedes und die Rechentechniken, über die er verfügte, mit Sicherheit beantworten. Es sind eben nicht alle mathematischen Schriften aus der Antike erhalten geblieben – wer rettet denn zuerst Mathematikbücher aus einer brennenden Bibliothek? Daher weiß man nicht, inwieweit den antiken Mathematikern Lösungsverfahren für Pellsche Gleichungen bekannt waren. Jedenfalls konnte man Pellsche Gleichungen schon vor Fermat lösen, wenn auch außerhalb des Rahmens der europäischen Mathematikgeschichte: Bereits um das Jahr 1000 lösten indische Mathematiker (Acarya Jayadeva und später Bhaskara Acharya) Pellsche Gleichungen, und manches spricht dafür, daß die von ihnen verwendeten Methoden schon 500 Jahre früher bekannt waren. Wie Clas-Olof Selenius in [8] zeigt, läßt sich übrigens auch das Lösungsverfahren der alten indischen Mathematiker mit Hilfe von Kettenbruchentwicklungen beschreiben; die dabei auftretenden Kettenbrüche sind, anders als bei Lagrange, halbregelmäßige Kettenbrüche und zwar solche, die besonders gute Approximationen liefern (vgl. Selenius [9]). Man darf selbstverständlich nicht meinen, daß die Verfahren der indischen Mathematiker in

Wirklichkeit aus der griechischen Welt stammen, aber man darf sich vorstellen, daß vielleicht auch dort schon früh ähnliche Methoden bekannt waren.

Man braucht nicht daran zu zweifeln, daß die Aufgabe, die Zahl der Rinder in der Herde des Gottes Helios zu berechnen, von Archimedes stammt. Von welchem anderen Mathematiker der griechischen Antike sollte sie schon stammen? Nicht sicher ist, ob Archimedes sie auch gelöst hat. Aber jeder, der sie selbst gelöst hat, darf darüber nachdenken und sich vorstellen, welche Aufgaben Archimedes gefunden und gelöst hätte, wenn er *MuPAD* zu Verfügung gehabt hätte.

Literatur

[1] G. E. Lessing, Zur Griechischen Anthologie. In: Gotthold Ephraim Lessings sämtliche Schriften, herausgegeben von K. Lachmann, Bd. 12, 99–115, G. J. Göschen'sche Verlagshandlung, Leipzig, 3. Auflage 1897, Nachdruck: de Gruyter, Berlin 1968

[2] Archimedes, Opera Omnia, herausgegeben von I. L. Heiberg (Bibliotheca Scriptorum Graecorum et Romanorum Teubneriana). Teubner, Stuttgart 1972, Nachdruck der Ausgabe von 1913

[3] O. Perron, Die Lehre von den Kettenbrüchen I und II. Teubner, Stuttgart, 2. Auflage 1954 und 1957

[4] F. Schwarz, Elementare Zahlentheorie mit *MuPAD*. Erscheint demnächst

[5] B. Krumbiegel, A. Amthor, Das Problema bovinum des Archimedes. Z. Math. Phys., Hist.-lit. Abt. **25** (1880) 121–136 und 153–171

[6] H. C. Williams, R. A. German, C. R. Zarnke, Solution of the cattle problem of Archimedes. Math. Comp. **19** (1965) 671–674

[7] H. L. Nelson, A solution to Archimedes' cattle problem. J. Recreational Math. **13** (1980/81) 162–176

[8] C.-O. Selenius, Rationale of the chakravāla process of Jayadeva and Bhāskara II. Historia Math. **2** (1975) 167–184

[9] C.-O. Selenius, Konstruktion und Theorie halbregelmäßiger Kettenbrüche mit idealer relativer Approximation. Acta Acad. Aboensis, Math. Phys. **22.2** (1960) 1–77

Symbolische Grenzwertberechnung mit Computer-Algebra-Systemen

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

Es wird ein Algorithmus zur symbolischen Bestimmung von Grenzwerten mit einem Computer-Algebra-System vorgestellt. Dieser Algorithmus wurde 1996 von Dominik Gruntz (ETH Zürich) im Rahmen seiner Promotion entwickelt und gilt zur Zeit als der beste Algorithmus zur symbolischen Bestimmung von Grenzwerten von exp-log-Funktionen. Er ist in MuPAD seit Version 1.2.2 implementiert.

Einführung

Grenzwertbestimmungen nehmen in vielen Bereichen symbolischer Berechnungen eine wichtige Rolle ein, so z.B. bei der Bestimmung von Unstetigkeiten und Singularitäten einer Funktion, bei bestimmter Summation und Integration, bei der Bestimmung einer geschlossenen Form für formale Reihenentwicklungen und bei der Bestimmung qualitativer Informationen über Funktionen wie Nullstellen, Maxima, Minima, Singularitäten, asymptotische Darstellungen und natürlich Grenzwerten.

Dieser Artikel beginnt mit einer kurzen Diskussion über zwei grundlegende Möglichkeiten zur symbolischen Bestimmung von Grenzwerten. Anschließend wird der Algorithmus von Dominik Gruntz für den Fall von exp-log-Funktionen vorgestellt.

Viele Beispiele dieses Artikels sind einem Preprint der Orginalarbeit von Dominik Gruntz entnommen. Ich möchte mich an dieser Stelle bei dem Autor für sein Entgegenkommen bedanken!

Diskussion verschiedener Lösungsansätze

Heuristische Verfahren

Verfahren zur symbolischen Bestimmung von Grenzwerten, die auf Anwenden bekannter mathematischer Regeln („Tricks“) basieren, wurden schon früh beispielsweise von Wang (1971) und Laurent (1973) entwickelt und implementiert.

Solche Methoden bedürfen mathematischer Intuition und bedienen sich bei der Implementation in ein Computer-Algebra-System diverser Heuristiken. Die Verfahren können fehlerhafte Resultate liefern, wenn Heuristiken falsch angewendet werden. Als Beispiel dazu diene die Regel

$$\lim_{x \to x_0} f(g(x)) = f\left(\lim_{x \to x_0} g(x)\right),$$

die nur dann gültig ist, wenn $g(x)$ stetig in x_0 ist. Für ein Computer-Algebra-System stellt jedoch das Testen auf Stetigkeit im Allgemeinen eine große Schwierigkeit dar.

Beispiel 1 *Der Grenzwert* $\lim\limits_{x \to 0} sign(x)$ *ist nicht definiert, wobei*

$$sign(x) := \begin{cases} -1 & : \quad x < 0 \\ 0 & : \quad x = 0 \\ 1 & : \quad x > 0 \end{cases}, \qquad (x \in \mathbb{R}).$$

Mathematica[1]*:*

```
In[3]:= <<Calculus/Limit.m
In[4]:= Limit[ Sign[x],x->0 ]

Out[4]= 0
```

MuPAD[2]*:*

```
>> limit( sign(x),x=0 );

          undefined
```

Ein weiteres Beispiel liefert die Regel von l'Hospital, die nur dann zur Berechnung von $\lim\limits_{x \to x_0} \dfrac{f(x)}{g(x)}$ angewendet werden darf, wenn der Grenzwert $\lim\limits_{x \to x_0} \dfrac{f'(x)}{g'(x)}$ existiert. Auch diese Voraussetzung ist im Allgemeinen für ein Computer-Algebra-System schwer zu überprüfen und ein naives Annehmen dieser Voraussetzung kann zu einem fehlerhaften Resultat führen (siehe [1]).

Darüberhinaus bieten solche Verfahren aus der Sicht des Benutzers kaum Möglichkeiten, die Grenzwertberechnung hinsichtlich der Behandlung neuer Funktionen zu erweitern.

Bei der Regel von l'Hospital ergeben sich noch weitere Schwierigkeiten, auf die hier kurz eingegangen sei. Zum einen sind asymptotische Verhalten der Form $\frac{\infty}{\infty}$ oder $\frac{0}{0}$ nicht immer auflösbar, wie das folgende Beispiel zeigt:

Beispiel 2

$$\lim_{x \to \infty} \frac{(e+1)^{x^2}}{e^x}.$$

Es ist a priori nicht entscheidbar, ob das iterative Anwenden der Regel von l'Hospital in endlichen Schritten zu einem Ergebnis führen wird. Daher werden weitere Heuristiken benötigt, wie oft die Iteration höchstens ausgeführt werden soll. Macsyma beispielsweise gibt eine obere Schranke in Form eines globalen Zählers (`lhospitallim`) vor, der standardmäßig auf den Wert 4 gesetzt ist. Bei Mathematica wird die Iteration nach 3 erfolglosen Schritten abgebrochen. Im Gegensatz zu Macsyma ist diese Schranke eine feste interne Größe, die durch den Benutzer nicht verändert werden kann.

---
[1] diese und weitere Aussagen beziehen sich auf Version 2.2
[2] diese und weitere Aussagen beziehen sich auf Version 1.2.9

Mathematica:

```
In[1]:= Limit[ x^3/Exp[x], x-> Infinity ]

Out[1]= 0

In[2]:= Limit[ x^4/Exp[x], x-> Infinity ]

                 4
                x
Out[2]= Limit[--, x -> Infinity]
                 x
                E
```

MuPAD:

```
>> assume( n>0 ):
>> limit( x^n/exp(x), x=infinity );

          0
```

Eine weitere Schwierigkeit in der Anwendung der Regel von l' Hospital findet sich darin, daß diese Regel sowohl auf den Quotienten $\frac{f(x)}{g(x)}$ als auch auf $\frac{f^{-1}(x)}{g^{-1}(x)}$ angewendet werden kann. Das kann jedoch in den Fällen: $f(x) \to 0$ oder $g(x) \to \infty$ (oder umgekehrt) zu einem exponentiellen Wachstum der Laufzeit führen. Daher ist eine weitere Heuristik nötig, um zu entscheiden, welche der Alternativen gewählt werden soll. Eine falsche Entscheidung kann zu einer Endlosrekursion führen, wie das folgende Beispiel zeigt:

$$\lim_{x\to 0} \ln(x)x = \lim_{x\to 0} \frac{x}{\ln^{-1}(x)} = \lim_{x\to 0} \frac{x}{-\ln^{-2}(x)} = \lim_{x\to 0} \frac{x}{2\ln^{-3}(x)} = \dots$$

Die Wahl der zweiten Alternative führt dagegen zu dem richtigen Ergebnis:

$$\lim_{x\to 0} \ln(x)x = \lim_{x\to 0} \frac{1}{x\ln(x)} = \lim_{x\to 0} -\frac{1}{x} = \lim_{x\to 0} \ln(x)x = 0.$$

MuPAD:

```
>> limit( ln(x)*x, x=0 );

                0
```

Reihenentwicklungen

Läßt sich die zu untersuchende Funktion in eine Potenzreihe um den Punkt, an dem der Grenzwert ermittelt werden soll, entwickeln, so kann in bestimmten Fällen das Grenzverhalten dem führenden Term der Entwicklung entnommen werden.

Solche Verfahren wurden u.a. von Zippel (1976) unter Verwendung von Taylorreihen, von Geddes and Gonnet (1988) unter Verwendung eines Modells für sogenannte „verallgemeinerte“ Potenzreihen oder von Salvy (1991) und Shakell (1990) verwendet. Auch der Algorithmus von Dominik Gruntz zählt zu der Klasse solcher Verfahren und verwendet Puiseuxreihenentwicklungen oder darüberhinaus verallgemeinerte Potenzreihenentwicklungen.

Beispiel 3 *Taylorreihenansatz.*

$$\lim_{x\to 0}\frac{(1+x)^s-1}{x}.$$

Intuitiv erhält man den unbestimmten Ausdruck $\frac{0}{0}$*, Reihenentwicklung aber liefert:*

$$\lim_{x\to 0}\frac{(1+x)^s-1}{x}=\lim_{x\to 0}\frac{1+s\,x+O(x^2)-1}{x}=\lim_{x\to 0}s+O(x)=s$$

Beispiel 4 *Verallgemeinerte Potenzreihen.*

$$\lim_{x\to\infty}\frac{\exp\left(\exp\left(\Psi\left(\Psi(x)\right)\right)\right)}{x},\ \textit{wobei}\ \Psi(x):=\frac{\Gamma'(x)}{\Gamma(x)}.$$

Man findet als Potenzreihenentwicklung von $\Psi(x)$ *um* $x=\infty$*:*

$$\Psi(x)=\ln x-\frac{1}{2x}+O\left(\frac{1}{x^2}\right). \tag{1}$$

Damit folgt weiter:

$$\begin{aligned}\exp\left(\Psi(x)\right) &= xe^{-\frac{1}{2x}}e^{O\left(\frac{1}{x^2}\right)}\\ &= xe^{-\frac{1}{2x}}\left(1+O\left(\frac{1}{x^2}\right)\right)\\ &= x\left(1-\frac{1}{2x}+O\left(\frac{1}{x^2}\right)\right)\\ &= x-\frac{1}{2}+O\left(\frac{1}{x}\right).\end{aligned}$$

Analog erhält man:

$$\exp\left(\exp\left(\Psi(x)\right)\right)=e^xe^{-\frac{1}{2}}e^{O\left(\frac{1}{x}\right)}=e^xe^{-\frac{1}{2}}\left(1+O\left(\frac{1}{x}\right)\right).$$

Ersetzt man darin x *durch die asymptotische Entwicklung von* $\Psi(x)$*:*

$$\exp\left(\exp\left(\Psi\left(\Psi(x)\right)\right)\right)=\left(x-\frac{1}{2}+O\left(\frac{1}{x}\right)\right)e^{-\frac{1}{2}}\left(1+O\left(\frac{1}{\Psi(x)}\right)\right),$$

so erhält man aus

$$\begin{aligned}\frac{\exp\left(\exp\left(\Psi\left(\Psi(x)\right)\right)\right)}{x} &= \frac{e^{-\frac{1}{2}}\left(x-\frac{1}{2}+O\left(\frac{1}{x}\right)+O\left(\frac{x}{\Psi(x)}\right)\right)}{x}\\ &\overset{(1)}{=} \frac{e^{-\frac{1}{2}}\left(x-\frac{1}{2}+O\left(\frac{x}{\ln(x)}\right)\right)}{x}\\ &= e^{-\frac{1}{2}}+O\left(\frac{1}{\ln x}\right)\end{aligned}$$

den Grenzwert zu $e^{-\frac{1}{2}}$.

Maple V[3]*:*

```
> limit( exp(exp(Psi(Psi(x))))/x, x=infinity );

        exp(-1/2)
```

Die Fähigkeiten solcher Grenzwertalgorithmen hängen entscheidend von dem verwendeten Reihenmodell ab. Die Funktion zur Reihenentwicklung liefert zugleich eine Schnittstelle zur Bestimmung von Grenzwerten neuer Funktionen.

Die bis dahin bekannten Verfahren waren jedoch im Allgemeinen nicht in der Lage, das Auslöschungsproblem (engl.: cancellation problem), in den Griff zu bekommen.

Die Bedeutung dieses Problems sei an der Berechnung des Grenzwertes

$$e^x \left(\sin\left(\frac{1}{x} + e^{-x}\right) - \sin\left(\frac{1}{x}\right)\right) \tag{2}$$

für $x \to \infty$ erläutert: Die Argumente der beiden Sinus-Terme streben für $x \to \infty$ gegen 0. Damit lassen sich diese Terme in eine Potenzreihe um $x = \infty$ entwickeln. Geht man hierbei getrennt vor und bildet anschließend die Differenz der beiden Entwicklungen, so erhält man:

$$\sin(1/x) = \frac{1}{x} - \frac{1}{6x^3} + \frac{1}{120x^5} + O\left(\frac{1}{x^6}\right)$$

sowie

$$\sin\left(1/x + e^{-x}\right) = \frac{1}{x} - \frac{1}{6x^3} + \frac{1}{120x^5} + O\left(\frac{1}{x^6}\right).$$

Ihre Differenz ergibt somit $O\left(\frac{1}{x^6}\right)$. Daraus läßt sich folgern, daß die Reihenentwicklungen der beiden Funktionen für die ersten n Terme identisch sind. Dabei kann n beliebige natürliche Werte annehmen. Die Information, daß die beiden Funktionen unterschiedliche Argumente besitzen, geht offensichtlich hierbei verloren.

Entwickelt man jedoch die Funktion $\sin\left(1/x + e^{-x}\right)$ in $w = e^{-x}$ um 0 ($w \to 0$ für $x \to \infty$):

$$\sin\left(1/x + e^{-x}\right) = \sin\left(\frac{1}{x}\right) + \cos\left(\frac{1}{x}\right) e^{-x} + O\left(e^{-2x}\right).$$

und anschließend die Sinus- und Cosinus-Terme um $x = \infty$, so erhält man für (2) die Entwicklung:

$$\begin{aligned} e^x \Bigg(&\left\{ \left(\frac{1}{x} - \frac{1}{6x^3} + \frac{1}{120x^5} + \ldots\right) + e^{-x}\left(1 - \frac{1}{2x^2} + \frac{1}{24x^4} - \frac{1}{720x^6} + \ldots\right) + \ldots \right\} \\ &- \left(\frac{1}{x} - \frac{1}{6x^3} + \frac{1}{120x^5} + \ldots\right)\Bigg). \end{aligned}$$

Somit ist der Grenzwert von (2) für $x \to \infty$ gleich 1.

MuPAD:

```
>> limit( exp(x)*(sin(1/x+exp(-x)) - sin(1/x)), x=infinity );

                             1
```

[3]diese und weitere Aussagen beziehen sich auf Release 3

Das obige Problem kann nur dann auftreten, wenn die Funktion in einem Term entwickelt wird, der im Sinne „asymptotischen Wachstums“ nicht hinreichend „groß“ genug ist (im obigen Beispiel wurde die Funktion in x anstatt in e^{-x} entwickelt). Diese Tatsache wird von dem Algorithmus von Dominik Gruntz berücksichtigt, indem eine Funktion immer in einem ihrer „größt“ möglichen asymptotischen Teilausdrücke entwickelt wird.

Der mrv-Algorithmus

Grundlagen

Es werden im folgenden reellwertige Funktionen einer reellen Variablen x betrachtet, die auf einem Intervall der Form $x > x_0 \in \mathbb{R}$ definiert sind.

Es sei $\mathcal{K}$ die Menge aller solcher Funktionen. Auf $\mathcal{K}$ kann nun wie folgt eine Äquivalenzrelation eingeführt werden: Seien f_1 und f_2 zwei Funktionen aus $\mathcal{K}$ definiert für $x > x_1$ bzw. $x > x_2$. Dann heißt f_1 *äquivalent zu* f_2, wenn ein $x_0 > \max(x_1, x_2)$ existiert, so daß $f_1(x) = f_2(x)\ \forall x > x_0$ gilt.

Die Äquivalenzklassen von $\mathcal{K}$ (bezüglich dieser Relation) werden als die *Keime* von Funktionen um ∞ bezeichnet.

Definition 1 *Ein* Hardy-Körper *ist eine Menge von Keimen reellwertiger Funktionen um ∞, die abgeschlossen unter Differentation ist und mit der gewöhnlichen Addition und Multiplikation von Keimen ein Körper ist.*

Satz 1 *Es sei $\mathcal{H}$ ein Hardy-Körper und $0 \neq f \in \mathcal{H}$. Dann gilt:*

1. *$\mathcal{H}$ enthält das Element $\frac{1}{f}$.*
2. $\exists x_0 \in \mathbb{R}\ \forall x > x_0 : f(x) \neq 0.$
3. $\exists x_0 \in \mathbb{R}\ \forall x > x_0 : f(x)$ *ist stetig und entweder immer positiv oder negativ.*
4. $\exists x_0 \in \mathbb{R}\ \forall x > x_0 : f(x)$ *ist streng monoton.*
5. *Für jedes $f \in \mathcal{H}$ existiert $c := \lim\limits_{x\to\infty} f(x)$ und es gilt: $c \in \mathbb{R}$ oder $c = \pm\infty$.*

Beispiele für Hardy-Körper sind $\mathbb{R}$ und $\mathbb{Q}$.

Satz 2 *Es sei $\mathcal{H}$ ein Hardy-Körper und $e \in \mathcal{H}$. Dann gilt:*

1. *$\mathcal{H}(y)$ mit $y' = y \cdot e'$ ist ein Hardy-Körper (Adjunktion der Exponentialfunktion).*
2. *$\mathcal{H}(y)$ mit $y' = \frac{e'}{e}$ ist ein Hardy-Körper (Adjunktion der Logarithmenfunktion).*

Inbesondere ist die Menge der exp-log-Funktionen ein Hardy-Körper.

Es wird nun ein Maß, das das Wachstum von Funktionen eines Hardy-Körpers charakterisiert, eingeführt.

Definition 2 *Es seien f und g zwei Elemente eines Hardy-Körpers $\mathcal{H}$, beide $\neq 0$ mit $f \to \infty$ und $g \to \infty$. Dann schreibt man:*

$$f(x) \succ g(x) \ :\Longleftrightarrow\ \forall p \in \mathbb{N} : \lim_{x\to\infty} \frac{g(x)^p}{f(x)} = 0.$$

Man sagt: $f(x)$ ist stärker wachsend *als $g(x)$.*

Analog lassen sich die Schreibweisen $f(x) \prec g(x)$ und $f(x) \asymp g(x)$ einführen. Letztere führt auf $\mathcal{H}$ eine Äquivalenzrelation ein, die wie folgt auf ganz $\mathcal{H}$ fortgesetzt ist:

- Für jedes $f \in \mathcal{H}$ sind $\pm f$ und $\pm\frac{1}{f}$ in der gleichen Äquivalenzklasse.
- Jedes $f \in \mathcal{H}$ mit einem endlichen Grenzwert liegt in derselben Äquivalenzklasse und es gilt: $f \preceq g$, wobei $g \in \mathcal{H}$ und $g \to \infty$ oder $g \to 0$.

Diese Erweiterung bedingt, daß die Terme e^x und e^{-x} in derselben Äquivalenzklasse enthalten sind.

Beispiel 5 *1.* $e^x \succ x^m$

2. $e^{x^2} \succ (e^x)^2$

3. $e^x \asymp e^{x+e^{-x}}$

4. $\ln f(x) \prec f(x)$

5. $e^{g(x)} \succ g(x)$

Satz 3 *Es sei $\mathcal{H}$ ein Hardy-Körper und f und g Elemente von $\mathcal{H}$. Dann gilt:*

1. $f(x) \succ g(x) \iff \lim\limits_{x\to\infty} \dfrac{\ln f(x)}{\ln g(x)} = \pm\infty.$

2. $f(x) \prec g(x) \iff \lim\limits_{x\to\infty} \dfrac{\ln f(x)}{\ln g(x)} = 0.$

3. $f(x) \asymp g(x) \iff \lim\limits_{x\to\infty} \dfrac{\ln f(x)}{\ln g(x)} = c \neq 0.$

Der Algorithmus

Es sei $\mathcal{H}$ der Körper der exp-log-Funktionen und $f \in \mathcal{H}$.

Algorithmus 1 $c := \lim\limits_{x\to\infty} f(x) =$

1. *Finde die Menge $\mathcal{S}$ der „am stärksten wachsenden" Teilausdrücke (s.u.) von $f(x)$.*
2. *Wähle ein Element ω aus $\mathcal{S}$ mit $\omega > 0$ und $\omega \to 0$. Schreibe alle Elemente von $\mathcal{S}$ in der Form $\omega^p h(x)$ mit $h(x) \prec \omega$ und betrachte alle Ausdrücke unabhängig von ω als konstant.*
3. *Entwickle $f(x)$ in eine Puiseux-Reihe in ω um $\omega = 0^+$, d.h. in eine Darstellung der Form*

$$a_0\omega^{e_0} + a_1\omega^{e_1} + \ldots + O\left(w^{e_n}\right)$$

   mit $e_i \in \mathbb{R}$, $e_i < e_{i+1}$ und $a_i \prec \omega$ für alle i.

4. *Ist $e_0 > 0$, so ist $c = 0$.*
   Ist $e_0 < 0$, so ist $c = sign(a_0) \cdot \infty$.
   Anderenfalls wende den Algorithmus auf $f(x) := a_0 \prec \omega$ an.

Dominik Gruntz zeigt in [1] die Terminierung des Algorithmus für den Körper der exp-log-Funktionen. Desweiteren läßt sich die Klasse der Funktionen, für die der Algorithmus den Grenzwert bestimmen kann, erweitern. Das führt zu dem sogenannten *mrv-Körper*, für dessen Elemente weitere Bedinungen erfüllt sein muß. Für den interessierten Leser sei erneut auf [1] verwiesen.

Abschließend wird die folgende Definition nachgereicht:

Definition 3 *Die* Menge der am stärksten wachsenden Teilausdrücke *einer Funktion $f(x) \in \mathcal{H}$ ist definiert als:*

$$\mathit{mrv}\left(f(x)\right) := \begin{cases} \{\}\text{, wenn } x \not\dashv f(x) \\ \{g(x) \mid g(x) \dashv f(x) \wedge (\nexists\, h(x) \dashv f(x) :\; h(x) \succ g(x))\} \end{cases}$$

Dabei bedeutet $h(x) \dashv g(x)$: $h(x)$ ist Teilausdruck von $g(x)$.

Damit ist auch der Name des mrv-Algorithmus motiviert. Einen Algorithmus zur Bestimmung von mrv $(f(x))$ findet sich in [1].

Es sei noch darauf hingewiesen, daß die Menge mrv $(f(x))$ von dem Ausdruck, der die Funktion $f(x)$ repräsentiert, abhängt. Für unterschiedliche Ausdrücke, die mathematisch zueinander äquivalent sind, können diese Mengen verschieden sein. Der Algorithmus liefert jedoch für solche Ausdrücke, von denen er ein Ergebnis berechnen kann, die gleichen Resultate.

Beispiel 6 *1. mrv* $\left(e^{x+\frac{1}{x}}\right) = \{e^{x+\frac{1}{x}}\}$

2. mrv $\left(e^{x+\frac{1}{e^x}}\right) = \{e^{x+\frac{1}{e^x}}, e^x\}$

3. mrv $\left(x + e^{-x^2}\right) = \{e^{-x^2}\}$

Abschließend wird der mrv-Algorithmus an einem Beispiel erläutert.

Beispiel 7

$$\lim_{x\to\infty} e^x \left(e^{\frac{1}{x}+e^{-x}} - e^{\frac{1}{x}}\right) =: \lim_{x\to\infty} f(x).$$

Es ist mrv$(f(x)) = \{e^x, e^{-x}\}$*. Man ersetzt e^{-x} durch ω, e^x durch $\frac{1}{\omega}$ und erhält den Ausdruck*

$$f(x) = \frac{1}{w}\left(e^{\frac{1}{x}+\omega} - e^{\frac{1}{x}}\right).$$

Die Puiseux-Reihe von $f(x)$ um $\omega = 0$ lautet:

$$e^{\frac{1}{x}} \cdot \omega^0 + \frac{1}{2} e^{\frac{1}{x}} \cdot \omega^1 + O\left(\omega^2\right).$$

Daher ist $\lim_{x\to\infty} f(x) = \lim_{x\to\infty} e^{\frac{1}{x}}$.

Nun ist mrv $\left(e^{\frac{1}{x}}\right) =$ *mrv* $\left(\frac{1}{x}\right) = \{x\}$*. Man wählt $\omega = \frac{1}{x}$ und erhält als zugehörige Puiseux-Reihe in $\omega = 0$:*

$$e^{\omega} = 1 + \omega + O\left(\omega^2\right).$$

Daraus folgt: $\lim_{x\to\infty} f(x) = 1$.

MuPAD:

```
>> limit( exp(x)*( exp(1/x+exp(-x)) - exp(1/x) ), x=infinity );

                1
```

Generelle Schwierigkeiten

Nulltest (zero-equivalence problem)

Aus der Unentscheidbarkeit dieses Problems im Allgemeinen (Richardson) folgt, daß das Problem der Bestimmung von Grenzwerten für bestimmte Klassen von Funktionen unlösbar ist. Denn ist das Problem $E = 0$ unentscheidbar, so ist damit der folgende Grenzwert nicht berechenbar:

$$\lim_{\epsilon \to 0} \frac{\epsilon}{\epsilon + E} = \begin{cases} 1 & : \quad E = 0, \\ 0 & : \quad \text{sonst.} \end{cases}$$

Es wird daher die Existenz eines Orakels zur Bestimmung auf Nullgleichheit angenommen.

Das Problem tritt auch dann auf, wenn die zugrundeliegende Funktion zur Bestimmung auf Nullgleichheit nicht in der Lage ist, die Entscheidung durchzuführen, obwohl das Problem entscheidbar ist.

Beispiel:

```
>> f:= w/((1+w)*2^(1/4)*(2^(1/2)+2)-(8+6*2^(1/2))^(1/2);

                      w
  -------------------------------------------
        1/2       1/2     1/4            1/2
  - (6 2    + 8)     + 2    (w + 1)  (2    + 2)

>> limit(f, w=0, Right);

                     0
```

Dieses Ergebnis ist falsch. Das Problem liegt darin, daß die Funktion zur Reihenentwicklung (`series`) nicht in der Lage ist, die Äquivalenz $2\sqrt[4]{2} + \sqrt[4]{8} - \sqrt{6\sqrt{2}+8} = 0$ festzustellen.

Mit einer „kleinen" Hilfestellung wird das richtige Ergebnis berechnet:

```
>> iszero(2^(1/4)*2+2^(3/4)-(2^(1/2)*6+8)^(1/2)):=TRUE:
>> limit(f, w=0, Right);

                 1
           -------------
              1/4    3/4
           2 2    + 2
```

Rechnen mit Parametern

Beispiel:

$$\lim_{x\to 0} x^n = \begin{cases} 1 & : \quad n \geq 0, \\ \text{nicht definiert} & : \quad n < 0 \end{cases}, \qquad (n \in \mathbb{R}).$$

Hier benötigt man einen Mechanismus zur Definition sogenannter „Eigenschaften" von Parametern (reellwertig, positiv, negativ, rational, ...).

Maple V:

```
> assume( n>0 ):
> limit( x^n, x=0 );

                0
> assume( n<0 ):
> limit( x^n, x=0 );

             undefined
```

MuPAD bietet derzeit nur eine prinzipielle Möglichkeit, mit Hilfe der Funktion `assume` einige wenige Eigenschaften zu spezifizieren, die von anderen Funktionen (`int`, `sign` und andere) berücksichtigt werden.

Dieser Mechanismus soll später von einer umfangreichen Library abgelöst werden, die Möglichkeiten zur Abfrage von Eigenschaften und eine Schnittstelle zur Erstellung von Eigenschaften und Regeln bereitstellen soll.

Ausblick

Die Implementation des mrv-Algorithmus in *MuPAD* ist noch nicht vollständig durchgeführt. Spezielle Erweiterungen hinsichtlich der verallgemeinerten Potenzreihenentwicklung (siehe `series` und `asympt`) fehlen. Weiterhin werden spezielle Funktionen wie beispielsweise `gamma` oder `psi` an ihren wesentlichen Singularitäten nicht berücksichtigt. Eine Ausnahme bildet die Fehlerfunktion `erf`, die bei $x = \infty$ eine wesentliche Singularität besitzt. Für sie ist eine solche Erweiterung bereits implementiert:

```
>> limit(
       (erf(n-exp(-exp(n)))-erf(n))*exp(exp(n))*exp(n^2)
   ,n=infinity);

                              2
                          - -----
                              1/2
                            PI
```

Für zukünftige Entwicklungen ist eine Implementation der Funktionen `limit` und `series` unter Verwendung der Library `domains` geplant. Damit lassen sich u.a. gezielte Normalisierungen von Koeffizienten einer Reihenentwicklung vornehmen.

Zudem soll die Funktion `limit` erweitert werden, um auch Funktionen mit oszillierenden wesentlichen Singularitäten (z.B. $\sin(x)$ bei $x = \infty$) behandeln zu können.

Literatur

[1] D. Gruntz. *On Computing Limits in a Symbolic Manipulation System.* phd thesis (preprint). ETH Zürich, 1996.

[2] G. H. Gonnet and D. Gruntz. *Limit Computation in Computer Algebra.* Technical Report No. 187. ETH Zürich, 1992.

A Model for Object Based Systems (Part 1)

Holger Naundorf
Universität GH Paderborn, snoo@uni-paderborn.de

There is no commonly accepted general model for object based systems. Therefore different object based systems cannot be combined because their models are not compatible. We introduce a general model for timed and untimed object based systems. The use of this model for all object based systems enables the combination of independently developed systems.

Introduction

To understand object based languages and systems it is important to have a model for these systems. Though there are some models, none of them is commonly accepted and general enough for modeling all object based systems.

Well known models are the actor model [2] and the layered semantics of POOL [3]. These models have the advantage that they model object based systems in an intuitive way, i.e. a system consists of objects that communicate with the help of a transmission unit. However, they have the disadvantage that timed object based systems cannot be modeled and that they are not hierarchical, i.e. object based systems do not represent objects and hence cannot be used as an object in an object based system. To avoid this problem, in [1] the concept of open systems is used, but this can only be viewed as a work-around complicating the theory.

Other well known models are process calculi like CSP [4], CCS [8] or the π-calculus [10]. There are timed variants like RtCCS or DtCCS [12]. These models are hierarchical, but the objects communicate via handshaking which contradicts the encapsulation of objects – for the author the most important property of objects –, because two objects must know the internal state of each other before being able to communicate. The same is true for models for asynchronous communication based on process calculi like [5].

This paper introduces a general hierarchical model for object based systems, called *gMobS*. To call a model general one first has to define what the model is used for. The only basic property of object based systems is, that they consist of objects that communicate by messages only. Obviously, apart from systems usually called object based, circuits and neural nets meet this condition, too. This implies that the model must not only be a timed model but must be able to model continuous sending of messages. At first glance this seems awkward but in fact it does not really complicate the theory and unifies two different theories. Moreover, in the future some objects of object based systems might be implemented in hardware and continuous sending of messages could become important. Obviously systems without time can be modeled with timed models.

We concentrate mainly on the important results of this theory – many proofs are omitted or sketched only. A more detailed study can be found in [9].

This article consists of two parts. The first part describes a very general formalism for objects, called I/O-descriptions, and investigates some properties. The second part, which will appear in the next mathPAD, describes a formalism for general object based systems. A canonical way to view an object based system as an object is introduced. Then it is investigated under which conditions an object, defined by an object based system, behaves well, e.g. respects a kind of causality. Finally an object based system representing a stack (LIFO) is presented.

Definitions

1. In this paper $\mathcal{T} \in \{\mathbb{N}_0, \mathbb{R}_+\}$ models time.
2. For $t_1, t_2 \in \mathcal{T}$ let $\mathcal{T}_{t_1}^{t_2} = \{t \in \mathcal{T} \mid t_1 \leq t < t_2\}$ be the set of all times between t_1 and t_2.
3. For an arbitrary set A and $f \in (\mathcal{T} \to A)$ let $f_{t_1}^{t_2} \in (\mathcal{T}_{t_1}^{t_2} \to A)$ with: $\forall t \in \mathcal{T}_{t_1}^{t_2} : f(t) = f_{t_1}^{t_2}(t)$
4. For an arbitrary set A let $f, g \in (\mathcal{T} \to A)$ and $t_0 \in \mathcal{T}$, then f equals g up to time t_0, $f =_{t_0} g :\Leftrightarrow$ $\forall t \in \mathcal{T}, t < t_0 : f(t) = g(t)$
5. Consider a relation $R \subset A \times B$, then $Do(R) = \{a \in A \mid \exists b \in B : (a, b) \in R\}$ is the domain of R.
6. For any set A, $f \in (\mathcal{T} \to A)$ and $\tau \in A$, $\mathcal{MT}_\tau(f) = \{t \in \mathcal{T} \mid f(t) \neq \tau\}$.

Objects

A universal model for object based systems requires a universal model for objects. Moreover, a model for objects should respect their most basic property – they hide their implementation and their state (they even hide, if they have a state).

The actor-model [2] and the layered semantics of POOL [3] can only describe a very limited set of objects. Systems of objects are not objects themselves. Process calculi like CSP [4], CCS [8] or the π-calculus [10] do not respect encapsulation of objects.

A suitable mechanism are (abstract) time systems introduced in system- and control theory [7]. Time systems are a generalization of finite traces [4], infinite traces [6], and timed traces [13].

I/O-descriptions

Definition:

1. A *time system* (see [7] Def. 5.2) is a triple $\mathcal{D} = (\mathcal{I}, \mathcal{O}, \mathcal{U})$ where $\mathcal{I} \neq \emptyset$ is an arbitrary set representing the set of messages possibly received, $\mathcal{O} \neq \emptyset$ is an arbitrary set representing the set of messages possibly sent and $\mathcal{U} \subset (\mathcal{T} \to \mathcal{I}) \times (\mathcal{T} \to \mathcal{O})$ is a set representing the behavior of the system.
2. An object provides a behavior for every input. An *I/O-description* is a time system $\mathcal{D} = (\mathcal{I}, \mathcal{O}, \mathcal{U})$ where $\mathcal{U} = \emptyset$ or $\mathcal{U}$ is a *total* relation: $\forall I \in (\mathcal{T} \to \mathcal{I})\ \exists O \in (\mathcal{T} \to \mathcal{O}) : (I, O) \in \mathcal{U}$

   If $\mathcal{U}$ is total, then $\mathcal{D}$ is called *real*.
3. $\mathcal{DS}_{\mathcal{I}}^{\mathcal{O}}$ is the set of all I/O-descriptions $(\mathcal{I}, \mathcal{O}, \mathcal{U})$.

   The I/O-description $\top := \top_{\mathcal{I}}^{\mathcal{O}} := (\mathcal{I}, \mathcal{O}, \emptyset)$ is needed for $\mathcal{DS}_{\mathcal{I}}^{\mathcal{O}}$ forming a lattice.

$\square$

This formalism implies that an I/O-description receives and sends messages all the time, while objects used in object based systems send and receive messages sometimes only. Non-receiving (non-sending) of messages can be interpreted as receiving (sending) messages with no information. Most formalisms simply neglect these empty messages since they are considered as useless.

An important example of a timeless object is a buffer:

Example: Let N be an arbitrary set and $\tau \notin N$. $N_\tau = N \cup \{\tau\}$. Let $\phi \subset N \times N$.

Then $\mathcal{D}_{Buf(N_\tau,\phi)} = (N_\tau, N_\tau, \mathcal{U}_{Buf(N_\tau,\phi)})$ with
$\mathcal{U}_{Buf(N_\tau,\phi)} = \{(I,O) \in (\mathcal{T} \to N_\tau) \times (\mathcal{T} \to N_\tau) \mid \exists f : \mathcal{MT}_\tau(I) \to \mathcal{MT}_\tau(O)$ with

1. f is bijective [no messages are lost or duplicated]
2. $\forall t \in \mathcal{MT}_\tau(I) : t \leq f(t)$ [a message is only sent after being received]
3. $\forall t \in \mathcal{MT}_\tau(I) : I(t) = O(f(t))$ [no message is changed]
4. $\forall t_1, t_2 \in \mathcal{MT}_\tau(I) : (f(t_1) < f(t_2) \Rightarrow t_1 < t_2 \vee (I(t_1), I(t_2)) \in \phi$ [a message is only sent earlier than another message, if it was received earlier or it was allowed to pass (by ϕ)]

$\mathcal{D}_{Buf(N_\tau,\phi)}$ is a buffer, that stores messages not equal to τ and sends them (without any changes) at a time which is not earlier than the receiving of them. A message n_1 is sent earlier than a message n_2 only if n_1 arrived earlier than n_2 or if n_1 is allowed to pass n_2, which is determined by $(n_1, n_2) \in \phi$.

This means that $\mathcal{D}_{Buf(N_\tau,\emptyset)}$ is a buffer which sends messages in exactly the same order as they arrive. In contrast $\mathcal{D}_{Buf(N_\tau,N\times N)}$ is a buffer, which sends all received messages in an arbitrary order.

A buffer can receive infinite many messages not equal to τ in a finite time interval. □

Determinism

An I/O-description that can answer with exactly one answer to every input is a special case. In reality this is almost never the case, because objects do not possess an exact clock. The condition for determinism is very restrictive, because time is considered. Objects, that are determined intuitively, usually are not determined according to the following definition. It is possible to define this kind of determinism in this model, too. But for that some messages representing empty messages must be marked and then determinism can be defined by the possible orders of the remaining messages.

Definition: A time system $\mathcal{D} = (\mathcal{I}, \mathcal{O}, \mathcal{U})$ is called *determined* or *functional*, if $\mathcal{U}$ is functional:
$\forall I \in (\mathcal{T} \to \mathcal{I})\, \forall (I, O_1), (I, O_2) \in (\mathcal{T} \to \mathcal{O}) : O_1 = O_2$.

In all other cases $\mathcal{D}$ is called not determined. □

In contrast to [7] throughout this paper the notion determined is used, because it seems more intuitive than functional.

Usually an observer is not interested, if an object is really able to answer with all the messages, (s)he expects. Moreover the observer does not have the chance to decide, if an object is able to answer with all messages (s)he expects.

Definition: Let $\mathcal{D}_1 = (\mathcal{I}_1, \mathcal{O}_1, \mathcal{U}_1)$ and $\mathcal{D}_2 = (\mathcal{I}_1, \mathcal{O}_1, \mathcal{U}_2)$ be I/O-descriptions. $\mathcal{D}_1$ is called more determined than $\mathcal{D}_2$, $\mathcal{D}_1 \succeq \mathcal{D}_2$, if: $\mathcal{U}_1 \subset \mathcal{U}_2$. □

Though the relation 'more determined' is a canonical relation, it seems not to be introduced in system theory. Probably this is due to the fact, that this relation is only meaningful, if the definition sets of the compared time systems are equal.

Remark: Let $\mathcal{D}_1$ and $\mathcal{D}_2$ be real I/O-descriptions with $\mathcal{D}_1 \succeq \mathcal{D}_2$. If $\mathcal{D}_2$ is determined, then $\mathcal{D}_1 = \mathcal{D}_2$. □

Lattice-Structure

Remark: The relation $\succeq$ is transitive, reflective and antisymmetric and hence a partial order. □

Definition: Let S be a set and let $\mathcal{D}_s = (\mathcal{I}_s, \mathcal{O}_s, \mathcal{U}_s) \in \mathcal{DS}_{\mathcal{I}}^{\mathcal{O}}$ for $s \in S$.

1. Let $\mathcal{D} = (\mathcal{I}, \mathcal{O}, \mathcal{U})$ with: $\mathcal{U} = \bigcup_{s \in S} \mathcal{U}_s$. Then $\mathcal{D} = \bigcup_{s \in S} \mathcal{D}_s$ is the union of the $\mathcal{D}_s$.
2. Let $\mathcal{D} = (\mathcal{I}, \mathcal{O}, \mathcal{U})$ with:
$$\mathcal{U} = \begin{cases} \bigcap_{s \in S} \mathcal{U}_s & : \bigcap_{s \in S} \mathcal{U}_s \text{ is total} \\ \emptyset & : \text{otherwise} \end{cases}$$
   Then $\mathcal{D} = \bigcap_{s \in S} \mathcal{D}_s$ is the intersection of the $\mathcal{D}_s$.

□

Lemma: Let S be a set and $D = \{\mathcal{D}_s \mid s \in S\} \subset \mathcal{DS}_{\mathcal{I}}^{\mathcal{O}}$.

1. $\bigcap_{s \in S} \mathcal{D}_s$ is the least upper bound of D.
2. $\bigcup_{s \in S} \mathcal{D}_s$ is the greatest lower bound of D.

□

Remark: $(\mathcal{DS}_{\mathcal{I}}^{\mathcal{O}}, \succeq)$ is a complete lattice with:

1. $\bigsqcup = \bigcap$
2. $\sqcap = \bigcup$
3. $\bot = (\mathcal{I}, \mathcal{O}, (\mathcal{T} \to \mathcal{I}) \times (\mathcal{T} \to \mathcal{O}))$
4. $\top = (\mathcal{I}, \mathcal{O}, \emptyset)$

□

The fact that $\mathcal{DS}_{\mathcal{I}}^{\mathcal{O}}$ is a complete lattice is interesting because the object based systems introduced in the next part of this article can be viewed as a monoton function — hence a recursive object based system has a least fixed point, which can be used as the I/O-description defined by this object based system.

Properties

The definition of I/O-descriptions is very general. There exist many I/O-descriptions, that do not describe a meaningful object. This section defines classes of I/O-descriptions, which are meaningful – especially in context with object based systems.

An object based system consists of a set of objects – defined by I/O-descriptions –, which can communicate with the help of a transmission unit. The transmission unit itself is defined by an I/O-description. The input into the transmission unit consists of the output of the objects and an input coming from the outside. The output of the transmission unit consists of the input into the objects and an output directed to the outside.

To develop a general model, the input from and the output to the outside should be considered independently from the input from and the output to the objects.

Definition: Let X be a set and for $x \in X$ let $W(x)$ be a set. Let $Z \subset X$, then

1. $Int_Z(\prod_{x\in X} W(x)) = \prod_{x\in X\setminus Z} W(x)$
2. $Int_Z : (\prod_{x\in X} W(x)) \to \prod_{x\in X\setminus Z} W(x), (s_x)_{x\in X} \mapsto (s_x)_{x\in X\setminus Z}$
3. $Ext_Z(\prod_{x\in X} W(x)) = \prod_{x\in Z} W(x)$
4. $Ext_Z : (\prod_{x\in X} W(x)) \to \prod_{x\in Z} W(x), (s_x)_{x\in X} \mapsto (s_x)_{x\in Z}$

$\square$

Pre-Causal

Let $\mathcal{D}_{NP} = (\mathcal{I}_{NP}, \mathcal{O}_{NP}, \mathcal{U}_{NP})$ with $\mathcal{T} = \mathbb{N}_0$, $\mathcal{I}_{NP} = \mathcal{O}_{NP} = \{0,1\}$ and $\mathcal{U}_{NP} = \{(I,O) \mid \forall n \in \mathbb{N}_0 : O(n) = I(n+1)\}$. The output of $\mathcal{D}_{NP}$ at time n is the input at time $n+1$. Every time $\mathcal{D}_{NP}$ must know what message it will receive next. In reality such an object cannot exist.

An I/O-description is called pre-causal, if at any time the output only depends on the received input up to that time. Since the behavior is not defined by calculations, it is not possible to find a causal connection between input and output. But one can say that the output of an I/O-description up to a time t_0 only depends on the input up to t_0, if for all inputs, that are equal up to t_0, the same output up to t_0 is possible (see [7] Def. 2.4).

Causal

Though the pre-causal I/O-descriptions are a quite nice class, they are too general to be handled. Therefore this notion is strengthened. An I/O-description is called causal, if it is the union of determined pre-causal I/O-descriptions. In [7] Def. 2.1 causal is defined by completely different means. However, both definitions are equivalent.

Pre-Delayed

For investigating object based systems, it is important to identify a huge class of object based systems, that themselves represent real pre-causal I/O-descriptions. If the objects and the transmission unit are only guaranteed to be causal, the I/O-description represented by this object based system can neither be guaranteed to be real nor pre-causal.

But if the transmission unit can continue its output for a short time without knowing the input coming from the objects during this time, the transmission unit can always continue a calculation a bit and by this produce a calculation.

The transmission unit must only be able to continue its output to the objects in this way – the transmission unit needs to be internally pre-delayed only.

Delayed

Similar to pre-causal I/O-descriptions the class of pre-delayed I/O-descriptions is not restricted enough to be handled. Therefore this notion is strengthened, too. An I/O-description is called delayed if it is the union of determined pre-delayed I/O-descriptions. Obviously the definition is analogous to the definition of causal. An analogous definition for delayed based on the definition of causal given in [7] is not known to the author.

Definition: Let $\mathcal{D} = (\mathcal{I}, \mathcal{O}, \mathcal{U})$ be a time system. $\mathcal{D}$ is called

1. *pre-causal*, if: $\forall (I, O) \in \mathcal{U}\, \forall t \in \mathcal{T}\, \forall \hat{I} \in Do(\mathcal{U})$ with $I =_t \hat{I}\ \exists (\hat{I}, \hat{O}) \in \mathcal{U} : O =_t \hat{O}$
2. *causal*, if there are a set S and determined pre-causal time systems $\mathcal{D}_s$ for $s \in S$ with: $\mathcal{D} = \bigcup_{s \in S} \mathcal{D}_s$
3. internally *pre-delayed* with respect to (Z_I, Z_O), if $\mathcal{D}$ is pre-causal with:
   $\forall (I, O) \in \mathcal{U}\, \forall t \in \mathcal{T}\, \exists t < t_0 \in \mathcal{T}\, \forall \hat{I} \in Do(\mathcal{U})$ with $I =_t \hat{I}$ and $Ext_{Z_I}(I) = Ext_{Z_I}(\hat{I})\ \exists (\hat{I}, \hat{O}) \in \mathcal{U}$:
   $O =_t \hat{O}$ and $Int_{Z_O}(O) =_{t_0} Int_{Z_O}(\hat{O})$

   $\mathcal{D}$ is called pre-delayed if it is internally pre-delayed with respect to $(\emptyset, \emptyset)$.
4. internally *delayed* with respect to (Z_I, Z_O), if there are a set S and determined pre-delayed time systems $\mathcal{D}_s$ for $s \in S$ with: $\mathcal{D} = \bigcup_{s \in S} \mathcal{D}_s$

   $\mathcal{D}$ is called delayed if it is delayed with respect to $(\emptyset, \emptyset)$.

□

Remarks on Properties

Remark: Let S be a set and $\mathcal{D}_s$ be time systems for $s \in S$. Let $\mathcal{D} = \bigcup_{s \in S} \mathcal{D}_s$.

1. $\forall s \in S : \mathcal{D}_s$ is pre-causal $\Rightarrow \mathcal{D}$ is pre-causal
2. $\forall s \in S : \mathcal{D}_s$ is causal $\Rightarrow \mathcal{D}$ is causal
3. $\forall s \in S : \mathcal{D}_s$ is internally pre-delayed with respect to $(Z_I, Z_O) \Rightarrow \mathcal{D}$ is internally pre-delayed with respect to (Z_I, Z_O)
4. $\forall s \in S : \mathcal{D}_s$ is internally delayed with respect to $(Z_I, Z_O) \Rightarrow \mathcal{D}$ is internally delayed with respect to (Z_I, Z_O)

□

Proof: Let $\mathcal{D} = (\mathcal{I}, \mathcal{O}, \mathcal{U})$ and for $s \in S$ let $\mathcal{D}_s = (\mathcal{I}_s, \mathcal{O}_s, \mathcal{U}_s)$.

1. Consider $(I, O) \in \mathcal{U}$, $t \in \mathcal{T}$, and $\hat{I} \in (\mathcal{T} \to \mathcal{I})$ with $I =_t \hat{I}$. Then there exists a $s \in S$ with $(I, O) \in \mathcal{U}_s$. Since $\mathcal{D}_s$ is pre-causal, there exists $(\hat{I}, \hat{O}) \in \mathcal{U}_s \subset \mathcal{U}$ with $O =_t \hat{O}$. Hence $\mathcal{D}$ is pre-causal.
2. The other cases are obvious or can be proved analogously.

□

Remark: Let $\mathcal{D}$ be a time system.

1. $\mathcal{D}$ causal $\Rightarrow \mathcal{D}$ pre-causal
2. $\mathcal{D}$ pre-delayed $\Rightarrow \mathcal{D}$ pre-causal
3. $\mathcal{D}$ delayed $\Rightarrow \mathcal{D}$ pre-delayed and causal

□

Proof: Union preserves the properties and a pre-delayed I/O-description is pre-causal by definition. □

Example: The buffer $\mathcal{D}_{Buf(N_\tau,\phi)}$ is causal and hence pre-causal.

While it is easy to show, that $\mathcal{D}_{Buf(N_\tau,\phi)}$ is pre-causal, it is a lot more work to show, that $\mathcal{D}_{Buf(N_\tau,\phi)}$ is causal. [7] Prop. 2.1 provides conditions for an I/O-description to be causal. In [7] it is said that these conditions are fulfilled in most cases. But even the slight generalization [9] theorem 4.38 of [7] Prop. 2.1 is not strong enough for $\mathcal{D}_{Buf(N_\tau,\phi)}$ and for most untimed objects. This illustrates the difference between system theory and the theory for object based systems. □

Remark: Let $\mathcal{T} = \mathbb{N}_0$, $\mathcal{D}_1 = (\{0,1\},\{0,1\},\{(I,O) \mid O(0) = 0 \wedge \forall n \in \mathbb{N}_0 : O(n+1) = I(n)\})$ and $\mathcal{D}_2 = \perp_{\{0,1\}}^{\{0,1\}}$. $\mathcal{D}_1$ and $\mathcal{D}_2$ are pre-causal, causal, pre-delayed and delayed.

Let $\mathcal{D}_3 = \mathcal{D}_1 \cup (\{0,1\},\{0,1\},\{(I,O) \mid \forall n \in \mathbb{N}_0 : O(n) = I(n+1)\})$. Then $\mathcal{D}_1 \succeq \mathcal{D}_3$ and $\mathcal{D}_3 \succeq \mathcal{D}_2$, but $\mathcal{D}_3$ is neither pre-causal, causal, pre-delayed nor delayed.

It seems that the $\succeq$ relation does not possess any interrelations with the defined properties. □

Remark:

1. Let $\mathcal{T} = \mathbb{N}_0$ and $\mathcal{D}_1 = (\{0,1\},\{0,1\},\mathcal{U}_1)$ with $\mathcal{U}_1 = \{(I,O) \mid O \equiv 0 \vee O = (1,1,0,0,0,\ldots)\}$.
   Let $\mathcal{D}_2 = (\{0,1\},\{0,1\},\mathcal{U}_2)$ with $\mathcal{U}_2 = \{(I,O) \mid O \equiv 0 \vee (O(0) = 1 \wedge \forall n \in \mathbb{N}_0 : O(n+1) = I(n)\}$. Both $\mathcal{D}_1$ and $\mathcal{D}_2$ are constantly delayed, i.e. causal, but the intersection $\mathcal{D}_1 \cap \mathcal{D}_2 = (\{0,1\},\{0,1\},\{(I,O) \mid O \equiv 0 \vee (I,O) = ((1,0,0,0,\ldots),(1,1,0,0,0,\ldots))\}$ is not pre-causal and hence neither causal nor pre-delayed.
2. Let $\mathcal{T} = \mathbb{N}_0$. For $n \in \mathbb{N}_0$ let $\mathcal{D}_n = (\{0,1\},\{0,1\},\mathcal{U}_n)$ with
   $\mathcal{U}_n = \{(I,O) \mid (\forall k \in \mathbb{N}_0$ with $k < n : O(k) = 0) \wedge (I,O) \not\equiv (0,0)\}$.

   For $n \in \mathbb{N}_0$ $\mathcal{D}_n$ is constantly delayed (with arbitrary delay) and causal.

   Additionally $\{\mathcal{D}_n \mid n \in \mathbb{N}_0\}$ is a chain with respect to (linearly ordered by) $\succeq$.

   But: $\bigcap_{n\in\mathbb{N}_0} \mathcal{D}_n = \top_{\{0,1\}}^{\{0,1\}}$

□

Other Characterizations of Causal

The definition of causal seems to be arbitrary and it is interesting to know if there exist other characterizations.

[7] Def. 2.1 defines causal in a completely different way. It is not obvious that both definitions have something in common. In [7] Prop. 2.8 it is shown that a time system is causal if and only if it has a state space representation ([7] Def. 2.8). The same is true for causal I/O-descriptions and consequently both definitions are equivalent.

Lemma: Let $\mathcal{D} = (\mathcal{I}, \mathcal{O}, \mathcal{U})$ be an I/O-description. $\mathcal{D}$ is causal if and only if there exist

1. a set $States_t$ of states for every $t \in \mathcal{T}$
2. a set $SStates \subset States_0$ of starting states
3. a state transition function $TransFunc_{t_1}^{t_2} \in (States_{t_1} \times (\mathcal{T}_{t_1}^{t_2} \to \mathcal{I}) \to States_{t_2})$ for all $t_1, t_2 \in \mathcal{T}$ with $t_1 \leq t_2$.
4. an output function $OutFunc_t \in ((States_t \times \mathcal{I}) \to \mathcal{O})$ for all $t \in \mathcal{T}$

such that

1. $\forall t_1, t_2, t_3 \in \mathcal{T}, t_1 \leq t_2 \leq t_3 \; \forall c \in States_{t_1} \; \forall I \in (\mathcal{T} \to \mathcal{I}) :$
   $TransFunc_{t_2}^{t_3}(TransFunc_{t_1}^{t_2}(c, I_{t_1}^{t_2}), I_{t_2}^{t_3}) = TransFunc_{t_1}^{t_3}(c, I_{t_1}^{t_3})$
2. $\forall t \in \mathcal{T} \; \forall c \in States_t \; \forall I \in (\mathcal{T}_t^t \to \mathcal{I}) : TransFunc_t^t(c, I) = c$
3. $\forall (I, O) \in (\mathcal{T} \to \mathcal{I}) \times (\mathcal{T} \to \mathcal{O}) :$
   $(I, O) \in \mathcal{U} \Leftrightarrow \exists c \in SStates \; \forall t \in \mathcal{T} : OutFunc_t(TransFunc_0^t(c, I_0^t), I(t)) = O(t)$

□

A similar characterization of causal is the notion of oraclizable I/O-descriptions (similar to [11]).

Remark: Let $\mathcal{D} = (\mathcal{I}, \mathcal{O}, \mathcal{U})$ be an I/O-description. $\mathcal{D}$ is causal if and only if there exist sets OS and $OI \subset (\mathcal{T} \to OS)$ and a determined, pre-causal I/O-description $\mathcal{D}_{\mathcal{O}r} = (\mathcal{I} \times OS, \mathcal{O}, \mathcal{U}_{\mathcal{O}r})$ such that:
$\mathcal{U} = \{(I, O) \mid \exists I_0 \in OI : ((I, I_0), O) \in \mathcal{U}_{\mathcal{O}r}\}$. □

Partial Objects

It is very laborious to define the complete behavior of an I/O-description. Moreover in literature objects cannot answer to every input, i.e. they are partially defined only.

Definition: Let $\mathcal{D}^P = (\mathcal{I}, \mathcal{O}, \mathcal{U}^P)$ be a time system. An I/O-description $\mathcal{D} = (\mathcal{I}, \mathcal{O}, \mathcal{U})$ is called the canonical I/O-description for the time system $\mathcal{D}^P$, if $\mathcal{U}^P \subset \mathcal{U}$ and $\mathcal{D} = \bigcup_{\mathcal{D}_d \in D} \mathcal{D}_d$ with
$D = \{\mathcal{D}_1 = (\mathcal{I}_1, \mathcal{O}_1, \mathcal{U}_1) \in \mathcal{DS}_{\mathcal{I}}^{\mathcal{O}} \mid$

1. $\mathcal{D}_1$ is pre-causal
2. $\forall (I, O) \in \mathcal{U}_1 \; \forall t \in \mathcal{T} \cup \{\infty\} : (\; \exists \hat{I} \in Do(\mathcal{U}^P)$ with $I =_t \hat{I} \Rightarrow \exists (I_1, O_1) \in \mathcal{U}^P : (I_1, O_1) =_t (I, O))$

$\}$.
If there does not exist such an I/O-description, then $\top_{\mathcal{I}}^{\mathcal{O}}$ is called the canonical I/O-description for $\mathcal{D}^P$. □

Remark: Let $\mathcal{D}^P$ be a time system with canonical I/O-description $\mathcal{D}$.

1. If $\mathcal{D}^P$ is pre-causal, then $\mathcal{D}$ is real and pre-causal.
2. If $\mathcal{D}^P$ is causal, then $\mathcal{D}$ is causal.
3. If $\mathcal{D}^P$ is not pre-causal, then $\mathcal{D} = \top$.
4. If $\mathcal{D}^P$ is a pre-causal I/O-description, then $\mathcal{D} = \mathcal{D}^P$.

□

Remark: Let $\mathcal{D}_1^P = (\mathcal{I}_1, \mathcal{O}_1, \mathcal{U}_1^P)$ and $\mathcal{D}_2^P = (\mathcal{I}_1, \mathcal{O}_1, \mathcal{U}_2^P)$ be pre-causal time systems with canonical I/O-descriptions $\mathcal{D}_1$ and $\mathcal{D}_2$. If $Do(\mathcal{U}_1^P) = Do(\mathcal{U}_2^P)$ and $\mathcal{U}_1^P \subset \mathcal{U}_2^P$, then $\mathcal{D}_1 \succeq \mathcal{D}_2$. □

Example: To illustrate the usefulness but perhaps even more the limitations of partial objects and the whole formalism introduced until now, an I/O-description $\mathcal{D}_{Stack}$ representing a very easy stack (LIFO) is defined.

Let X be an arbitrary set. Besides the empty message τ the stack can get messages $push(x)$ with $x \in X$ and the message $pop()$ (the non-empty messages are syntactically function calls to remind of the syntax used in most object based languages to send a message). A message $push(x)$ pushs the value x onto the stack. A message $pop()$ takes the value from the top of the stack and returns it. If $\mathcal{D}_{Stack}$ receives infinite many non-empty messages in a finite time or receives more $pop()$ messages than $push(x)$ messages, then $\mathcal{D}_{Stack}$ behaves chaotically, i.e. can produce any output.

For defining $\mathcal{D}_{Stack} = (\{\tau, pop()\} \cup \{push(x) \mid x \in X\}, \{\tau\} \cup X, \mathcal{U}_{Stack})$ some further formalism is required.

Since $\mathcal{U}_{Stack}$ will be defined explicitly only for inputs $I \in (\mathcal{T} \to \{\tau, pop()\} \cup \{push(x) \mid x \in X\})$ with only finite many non-empty messages in finite time and no more $pop()$ messages than $push(x)$ messages at every time, in the following only such inputs are considered.

Let $PushTime(I) = \{t \in \mathcal{T} \mid \exists x \in X : I(t) = push(x)\}$ and $PopTime(I) = \{t \in \mathcal{T} \mid I(t) = pop()\}$.
Then let $g_I : PopTime(I) \to PushTime(I)$ with $g_I(t) = \max\{t_0 \in PushTime(I) \mid t_0 < t \wedge |\{\hat{t} \in PushTime(I) \mid t_0 < \hat{t} < t\}| = |\{\hat{t} \in PopTime(I) \mid t_0 < \hat{t} < t\}|\}$.

The function g_I exists (due to the special inputs considered) and for $t \in PopTime(I)$ $g_I(t)$ is the time, when the value that is now on top of the stack, was pushed onto the stack.

Then for an input I the pair $(I, O) \in \mathcal{U}_{Stack} \Leftrightarrow$ it exists a function $f : PopTime(I) \to \mathcal{MT}_\tau(O)$ with

1. f is bijective [every $pop()$-message is answered exactly one time]
2. $\forall t \in PopTime(I) : t \leq f(t)$ [only $pop()$-messages already received are answered]
3. f is monotone [the $pop()$-messages are answered in the order they are received]
4. $\forall t \in PopTime(I) : Push(O(f(t))) = I(g_I(t))$ [a $pop()$-message is answered with the current top element of the stack]

The function f was introduced, because $\mathcal{D}_{Stack}$ need not answer a $pop()$-message immediately but can delay the answer by an arbitrary amount of time.

For any input not considered the possible outputs are defined implicitly. Problems with this definition are:

1. The definition is not very intuitive — it is not obvious, that $\mathcal{D}_{Stack}$ represents a stack.
2. Though it is very easy to verify, that $\mathcal{D}_{Stack}$ is pre-causal it is hard to show, that it is causal (but if the stack did not have the chance to delay its answers arbitrarily but had to answer immediately the defined partial object would be determined and hence causal).

Both problems will not arise if the formalism introduced in the second part of this article is used — exactly the same I/O-description will be defined in the second part with means developped there. □

Conclusion and Preview

In this first part of this article a very general model for objects was introduced — the already very general concept of traces is generalized to continous communication. Very intutive properties like pre-causal and pre-delayed as well as technical important properties (especially for the second part of this article) like causal and delayed were defined and their interplay was investigated.

The next part of this article will introduce systems consisting of objects that communicate by messages only. This implies that no new objects can be created. To obtain the power needed for object based system defined by object based languages, such a system can consist of infinite many objects. Creating a new object means activating an object. An object based system can be viewed as an object again and it will be shown, that systems consisting of causal objects with delayed communication represent causal objects. Finally an object based system with infinte many objects representing a stack (LIFO) will be presented.

References

[1] Gul Agha, Ian A. Mason, Scott Smith, and Carolyn Talcott. Towards a theory of actor computation. In W. R. Cleaveland, editor, *CONCUR '92, Third International Conference on Concurrency Theory*, volume 630 of *Lecture Notes in Computer Science*, pages 565–579, Berlin, August 1992. Springer-Verlag.

[2] Gul A. Agha. *Actors: A Model of Concurrent Computation in Distributed Systems*. The MIT Press series in artificial inteligence. MIT Press, 1986.

[3] Pierre America and Jan Rutten. A layered semantics for a prallel object-oriented language. In J.W. de Bakker, W. P. de Roever, and G. Rozenberg, editors, *Foundations of Object-Oriented Languages*, volume 489 of *Lecture Notes in Computer Science*, pages 91–123, Berlin, May 1990. REX School, Springer-Verlag.

[4] C. A. R. Hoare. *Communicating Sequential Processes*. Series in Computer Science. Prentice Hall, 1985.

[5] Kohei Honda and Mario Tokoro. On asynchronous communication semantics. In M. Tokoro, O. Nierstasz, and P. Wegner, editors, *Object-Based Concurrent Computing, ECOOP '91 Workshop*, volume 612 of *Lecture Notes in Computer Science*, pages 21–51. Springer-Verlag, July 1991.

[6] Bengt Jonsson. A fully abstract trace model for dataflow networks. In *Sixteenth Annual ACM Symposium on Principles of Programming Languages*, pages 155–165. ACM Press, 1989.

[7] M.D. Mesarović and Y. Takahara. *Abstract Systems Theory*, volume 116 of *Lecture Notes in Control and Information Science*. Springer-Verlag, 1989.

[8] Robin Milner. *Communication and Concurrency*. International Series in Computer Science. Prentice Hall, 66 Wood Labe End, Hemel Hempstead Hertfordshire, HP2 4RG, 1989.

[9] Holger Naundorf. *Ein denotationales Domain für parallele objekt-basierte Systeme (to appear)*. Phd thesis, Universität Paderborn, Germany, 1995.

[10] Nielson and Engberg. A calculus of communicating systems with label passing. Research Report DAIMI PB-208, Computer Science Department, University of Aarhus, 1986.

[11] James R. Russel. On oraclizable networks and kahn's principle. In *Seventeenth Annual ACM Symposium on Principles of Programming Languages*, pages 320–328, January 1990.

[12] Mario Tokoro and Kazunori Takashio. Toward languages and formal systems for distributed computing. In Rachid Guerraoui, Oscar M. Nierstrasz, and Michel Riveill, editors, *ECOOP '93 – Object-Based Distributed Programming*, volume 791 of *Lecture Notes in Computer Science*, pages 93–110, Berlin, July 1993. Springer-Verlag.

[13] Robert Kim Yates. Networks of real-time processes. In Eike Best, editor, *CONCUR '93, 4th International Conference on Concurrency Theory*, volume 715 of *Lecture Notes in Computer Science*, pages 384–397, Berlin, August 1993. Springer-Verlag.

Arbeiten mit Listen

Klaus Drescher
Universität GH Paderborn, kg@uni-paderborn.de

Listen sind eines der wichtigsten strukturierten Datentypen in MuPAD. Dieser Artikel gibt Hinweise zur Verwendung von Listen bei der MuPAD-Programmierung.

Eine Liste ist in *MuPAD* eine geordnete Sequenz von in eckigen Klammern eingeschlossenen Ausdrücken. Die Ausdrücke können beliebige Typen haben, die Liste darf natürlich auch leer sein. Zwei Elemente der Liste können identisch sein. Eine Liste hat den Basistyp `DOM_LIST`, dieses Domain wird als Typ einer Liste von den Funktionen `domtype` und `type` zurückgeliefert:

```
>> l := [ 1, x, a + b, [2/3, []], {a}, x ];

              [1, x, a + b, [2/3, []], {a}, x]

>> domtype(l);

                          DOM_LIST
```

Die Elemente einer Liste können durch beliebige Ausdrucks-Sequenzen definiert werden. Diese können natürlich auch mit dem Sequenz-Operator erzeugt werden:

```
>> s:= a, b, c: [ s ];

                          [a, b, c]

>> l:= [ x_.i $ i=1..6 ];

               [x_1, x_2, x_3, x_4, x_5, x_6]
```

Die Anzahl der Elemente einer Liste kann mit der „generischen“ Funktion `nops` erfragt werden:

```
>> nops(l);

                              6
```

Eine Liste wird evaluiert, indem zuerst alle ihre Elemente evaluiert werden, danach wird die Liste vereinfacht:

- Ein „leeres“ Datum vom Typ `DOM_NULL`, wie es zum Beispiel von der Funktion `null` zurückgeliefert wird, wird aus der Liste entfernt.

- Elemente, die Ausdrucks-Sequenzen sind, werden „geflattet“. Soll heißen, das statt der Sequenz die Elemente der Sequenz in die Liste eingetragen werden.

Ein Beispiel für die Vereinfachung:

```
>> s:= a, b, []:
>> l:= [ 1, null(), 2, s, 3 ];

                    [1, 2, a, b, [], 3]

>>nops(l);

                              6
```

Intern werden Listen durch Verweise auf ihre Elemente repräsentiert. Diese Verweise sind alle in einem gemeinsam alloziierten Speicherbereich enthalten. Das ist anders als in den meisten anderen funktionalen Programmiersprachen, in denen Listen als verkettete einzelne Verweise repräsentiert werden. Die theoretische Grenze für die Anzahl der Listenelemente ist $2^{32} - 1$, aber die wenigsten Systeme dürften genügend Hauptspeicher haben, um eine solch lange Liste zu erzeugen.

Mit der Funktion `op` können Elemente aus einer Liste herausgeholt werden, die Operanden sind dabei von 1 bis `nops(l)` durchnummeriert. Werden keine Indizes angegeben, so liefert `op` alle Elemente als Ausdrucks-Sequenz:

```
>> op(l, 3);

                              a

>> op(l, 4..nops(l));

                          b, [], 3

>> op(l);

                     1, 2, a, b, [], 3
```

Bei einer leeren Liste liefert `op` das leere Datum zurück:

```
>> op([]);


>> type(op([]));

                          DOM_NULL
```

Bei „leeren“ Bereichen liefert `op` ebenfalls das leere Datum zurück:

```
>> type(op(l, 3..1));

                          DOM_NULL
```

Die Funktion `op` liefert bei ungültigen Indizes übrigens keinen Fehler, sondern `FAIL`:

```
>> op(l, 7), op(l, 0);

                                FAIL, FAIL
```

Auf einzelne Listenelemente kann man nicht nur mittels `op`, sondern auch per Index-Operator zugreifen. Dieser liefert bei falschen Indizes allerdings einen Fehler:

```
>> l[3];

                                     a

>> l[0];

Error: Invalid index [list]
```

Der indizierte Zugriff ist übrigens schneller als der Zugriff mittels `op`.

Mit einer „for-in"-Schleife kann man bequem alle Elemente einer Liste durchlaufen:

```
>> S:= {}:
>> for e in l do S:= S union { e } end_for;

                          {[], a, b, 1, 2, 3}
```

Das Ergebnis des letzten Beispiels kann übrigens viel eleganter erreicht werden:

```
>> S:= { op(l) };

                          {[], a, b, 1, 2, 3}
```

Listenelemente können mittels indizierter Zuweisung verändert werden:

```
>> l[1]:= XX:
>> l;

                          [XX, 2, a, b, [], 3]
```

Daneben kann auch die Funktion `subsop` verwendet werden, um Elemente zu ändern:

```
>> l:= subsop(l, 1=AA, 2=BB);

                         [AA, BB, a, b, [], 3]
```

Meist dürfte die indizierte Zuweisung aber bequemer sein.

Ein mögliches Verfahren, einzelne Listenelement zu löschen, ist die indizierte Zuweisung des Wertes `NIL` an ein Listenelement:

```
>> l[1]:= NIL:
>> l;

                     [BB, a, b, [], 3]
```

Das bedeutet aber nicht, daß man `NIL` nicht als Listenelement verwenden dürfte. Nur die indizierte Zuweisung von `NIL` löscht Elemente:

```
>> l:= subsop(l, 2=NIL);

                    [BB, NIL, b, [], 3]
```

Eine andere Möglichkeit, Elemente aus einer Liste zu löschen, ist deren Substitution durch das leere Datum. Das leere Datum wird nach der Substitution durch die Vereinfachung aus der Liste entfernt und bewirkt dadurch das Löschen der ursprünglichen Elemente:

```
>> l:= subsop(l, 2=null(), 4=op([]));

                         [BB, b, 3]
```

Mit der Funktion `append` kann man ein oder mehrere Elemente an eine Liste anhängen:

```
>> l:= append(l, x);

                       [BB, b, 3, x]

>> l:= append(l, y $ 3);

                   [BB, b, 3, x, y, y, y]
```

Weiter kann man Listen mit dem „Punkt-Operator" (`_concat`) konkatenieren:

```
>> l:= [a, b] . [2, 3] . [c, d];

                    [a, b, 2, 3, c, d]
```

Das Anhängen von Elementen mittels `append` ist allerdings effizienter als die Konkatenation mit dem Punkt-Operators, falls dazu die Liste für die anzuhängenden Elemente erst neu erzeugt werden muß.

Eine Liste kann als Funktion aufgefaßt werden. Hierbei wird jedes Listenelement als Funktion auf die Argumente angewandt, die Ergebnisse werden als Liste zurückgeliefert:

```
>> ([ sin, cos, f ])(x);

                   [sin(x), cos(x), f(x)]
```

Die Liste muß hier aus syntaktischen Gründen geklammert werden. (Diese Funktionalität wird übrigens beim Start von *MuPAD* durch die System-Initialisierungs-Datei definiert, indem für das Domain `DOM_LIST` eine `func_call`-Methode definiert wird.)

Andererseits kann man mit der Funktion `map` eine Funktion auf alle Elemente einer Liste anwenden, die Ergebnisse werden wieder als Liste zurückgeliefert:

```
>> l:= [a+b, 2/3, 2.0]:
>> map(l, type);

                 ["_plus", DOM_RAT, DOM_FLOAT]
```

Zusätzliche Argumente von `map` werden jeweils als weitere Argumente der Element-Funktion übergeben:

```
>> map(l, f, x, y);

         [f(a + b, x, y), f(2/3, x, y), f(2.0, x, y)]

>> map(l, _plus, 3);

                     [a + b + 3, 11/3, 5.0]
```

Die Funktion `select` dient dazu, Elemente aus einer Liste „herauszufiltern“. Hierzu wird eine Funktion als Prädikat auf alle Elemente der Liste angewandt. Die Funktion muß sich in einem booleschen Kontext zu `TRUE` oder `FALSE` evaluieren. Elemente, für die die Funktion `FALSE` liefert, werden aus der Liste entfernt. Wie bei `map` können zusätzliche Argumente für die Selektions-Funktion angegeben werden:

```
>> select(l, has, a);

                            [a + b]

>> select(l, fun(degree(args(1)) = 0));

                           [2/3, 2.0]
```

Die Funktionen `map` und `select` können nicht nur auf Listen, sondern auch auf viele andere Datentypen angewandt werden, nähere Informationen liefern die Help-Seiten.

Manchmal möchte man die Elemente zweier Listen paarweise verknüpfen, etwa paarweise addieren. Hierzu dient die Funktion `zip`:

```
>> zip([a, b, c], [1, 2, 3], _plus);

                     [a + 1, b + 2, c + 3]
```

Haben beide Listen eine unterschiedliche Länge, so hängt das Verhalten von `zip` vom Vorhandensein eines 4ten Argumentes ab. Fehlt ein 4tes Argument, so wird die Funktion nur auf die vorhandenen Paare angewandt:

```
>> zip([a, b, c, d], [1, 2], _plus);

                         [a + 1, b + 2]
```

Wird dagegen ein 4tes Argument angegeben, so wird dieses als Ersatz für die „fehlenden“ Elemente der kürzeren Liste verwendet:

```
>> zip([a, b, c, d], [1, 2], _plus, 0);

                    [a + 1, b + 2, c, d]
```

Diese Funktionen können oft wirkungsvoll kombiniert werden. Im folgenden Beispiel wird die Liste `l2` elementweise von der Liste `l1` subtrahiert, danach werden die negativen Elemente entfernt:

```
>> l1:= [1, 2, 3]:
>> l2:= [-2, 1, 4]:
>> select( zip(l1, map(l2, -id), _plus), not _less, 0);

                                  [3, 1]
```

Die Funktion `sort` sortiert eine Liste aufsteigend. Man kann beim Aufruf von `sort` eine Ordnungsfunktion angeben oder die Default-Sortierung verwenden. Die Default-Sortierung ist folgende:

- Listen von reelen Zahlen werden gemäß der „natürlichen" Ordnung „<" sortiert.
- Listen von Zeichenketten werden lexikographisch sortiert.
- Alle anderen Listen werden gemäß einer „internen" Ordnung sortiert. Dies ist eine totale Ordnung für alle *MuPAD*-Ausdrücke, sie ist aber nicht näher spezifiziert.

Einige Beispiele:

```
>> sort([1, 13, 2/3, -0.2]);

                     [-0.2, 2/3, 1, 13]

>> sort(["a", "z", "ax"]);

                      ["a", "ax", "z"]
```

Wird eine Ordnungsfunktion `f` angegeben, so muß diese 2 Argumente akzeptieren. Der Aufruf `f(a,b)` muß `TRUE` liefern, falls die Relation $a \prec b$ bezüglich der gewünschten Ordnung $\prec$ gilt. Die Ordnungsfunktion wird in einem booleschen Kontext evaluiert. Um zum Beispiel eine Liste von reelen Zahlen absteigend zu sortieren, kann man `not _less` als Ordnungsfunktion verwenden:

```
>> sort([1, 13, 2/3, -0.2], not _less);

                     [13, 1, 2/3, -0.2]
```

Intern wird von `sort` ein modifizierter Quicksort-Algorithmus verwendet.

Wie man sieht, ist die Liste eine sehr flexible Datenstruktur, für die mächtige Manipulations-Funktionen in *MuPAD* existieren.

Mancher Leser mag sich fragen, welche Stellung der Typ Ausdrucks-Sequenz zum Typ Liste hat. Eine Ausdrucks-Sequenz ist ja auch — ähnlich wie eine Liste — eine geordnete Folge von Elementen.

Der erste Unterschied ist, daß eine Ausdrucks-Sequenz kein eigener Basistyp ist. Eine Ausdrucks-Sequenz hat den Basistyp `DOM_EXPR` und den Ausdruckstyp `"_exprseq"`:

```
>> s:= 1, a, 3:
>> domtype(s), type(s);

                        DOM_EXPR, "_exprseq"
```

Der „vornehmste“ Zweck einer Ausdrucks-Sequenz ist ihre Verwendung als Folge von Argumenten für einen Funktionsaufruf, dazu wurde sie geschaffen:

```
>> _plus(s);

                                   a + 4
```

Die allermeisten *MuPAD*-Funktionen „flatten“ Ausdrucks-Sequenzen wie im obigen Beispiel. Um ein „natürliches“ Verhalten von Ausdrucks-Sequenzen als Argumente eines Aufrufs zu erzielen, führt die Funktion `_exprseq` noch zwei wesentliche Vereinfachungen durch:

- Die leere Sequenz `_exprseq()` evaluiert sich zum leeren Datum.
- Eine Sequenz mit einem Element, wie etwa `_exprseq(x)`, evaluiert sich zu `x`.

Diese Vereinfachungen sind der wesentliche Unterschied zur Liste: Eine Sequenz ohne Elemente löst sich in Wohlgefallen auf. Eine Sequenz mit nur einem Element wird zu diesem Element evaluiert. Im Gegensatz dazu kann es durchaus Listen mit einem oder auch garkeinem Element geben, eine Liste wird nie zu einem neuen Typ evaluiert.

Die Vereinfachung der Ausdrucks-Sequenzen kann zu manchen unliebsamen Überaschungen führen. Man betrachte folgendes Beispiel:

```
>> l:= [ 2/3 ]:
>> S:= 0:
>> for e in op(l) do S:= S + e end_for;

                                      5
```

Es ist klar, was passiert: `op(l)` liefert `2/3`. Diese rationale Zahl hat wiederum die Operanden `op(2/3) = 2, 3`. Die „for-in“-Schleife durchläuft diese Operanden und berechnet als Summe `2+3 = 5`.

Was bei der interaktiven Eingabe leicht zu sehen ist, kann in einer Prozedur recht verwirrend sein, wenn nämlich die Operanden eines Ausdrucks „generisch“ mit `op` erfragt werden und man den Spezialfall, daß nur ein Operand vorhanden ist, nicht extra berücksichtigt. Oft ist es in solchen Fällen angebracht, die Operanden durch eine Liste zu „schützen“, wie in folgendem Beispiel:

```
for e in [ op(l) ] do ... end_for;
```

MuPAD in Schulen und Hochschulen

Frank Postel
Universität GH Paderborn, `frankp@uni-paderborn.de`

Dieser Artikel stellt eine Zusammenfassung von Aktivitäten und Projekte der MuPAD-Gruppe im Bereich Schulen und Hochschulen dar. Er besteht zum überwiegenden Teil aus Artikeln der Rubrik „The Educational Site“ der MuPAD Web Seite[1], auf die für weitere aktuelle Informationen verwiesen sei.

Ein Interaktives Buch zur Linearen Algebra

Es gibt zahlreiche Lehr- und Übungsbücher zur Linearen Algebra, wie sie an Hochschulen gelehrt wird. Doch trotz des starken Aufkommens von Computer Algebra Systemen in den letzten Jahren auch in der Lehre der Mathematik gibt es nur sehr wenig Lehr- und Übungsmaterial, das ein Gebiet der Mathematik in geschlossener Form unter Einbeziehung eines Computer Algebra Systems behandelt.

Für die Verwendung eines Computer Algebra Systems bietet sich die Lineare Algebra in hervorragender Weise an. Das Matrizenkalkül erleichtert das Formulieren von Ergebnissen der Linearen Algebra und ist in ihr das zentrale Werkzeug für viele Berechnungen. Doch die Komplexität dieser Berechnungen birgt die Gefahr in sich, die theoretischen Zusammenhänge der Linearen Algebra aus den Augen zu verlieren.

Hier können Computer Algebra Systeme das Erkennen und Verstehen von grundlegenden Aussagen unterstützen, indem sie dem Lernenden einmal geschulte praktische Tätigkeiten (wie beispielsweise das Multiplizieren von Matrizen) abnehmen. Sie sind in der Lage, sekundenschnell solche Berechnungen auch symbolisch durchzuführen.

Das interaktive Lehrbuch [5] soll ein erster Beitrag dazu sein. Es enthält ausgewählte Gebiete eines einsemestrigen Kurses zur Linearen Algebra und richtet sich an Studenten der Informatik, der Physik, der Ingenieurwissenschaften und der Angewandten Mathematik.

Das Buch versteht sich als ein begleitendes *Übungsbuch* zu einer vollständigen Abhandlung der Grundlagen der Linearen Algebra. Diese kann in Form eines Lehrbuches oder eines Vorlesungsskriptes einer Anfängervorlesung über Lineare Algebra vorliegen.

Die Auswahl der Beispiele und Übungen geschah in ihrem überwiegenden Teil unter dem Aspekt der Verwendbarkeit eines Computer Algebra Systems.

Darüberhinaus finden sich eine Fülle von Definitionen und mathematischen Aussagen, die meist ohne Angabe eine Beweises *zitiert* werden. Somit kann dem Leser das Buch ebenso als ein Nachschlagewerk zur Linearen Algebra dienen, das einen zusätzlichen Wert durch die interaktiven Nutzungsmöglichkeiten des Textes gewinnt.

Die *MuPAD* spezifischen Teile basieren auf der *MuPAD* Version 1.2.2a vom Februar 1996. Für diese ist zur Benutzung des Buches die Library `student` entwickelt worden (siehe [5]), die auf den Libraries `domains` [1, 2] und `linalg` [4] basiert.

[1] `http://math-www.uni-paderborn.de/MuPAD/`

Teile der Library `linalg` werden im Buch vorgestellt und der Umgang mit ihren Funktionen geübt. Am Ende des Buches wird der Leser jedoch nach dem Lösen der Übungsaufgaben auf ein kleines, selbstgeschriebenes Lineare Algebra Paket zurückblicken können.

2-dimensionale Geometrie mit *MuPAD* – Die Library `geometrie`

Dieser Abschnitt beschreibt eine erste Version der *MuPAD* Library `geometrie`, die unter Leitung von Herrn Dr. F. Schwarz[2] im Rahmen einer Examensarbeit (Sekundarstufe II) von Frau Klein entwickelt wird.

Die Library `geometrie` enthält Funktionen, die im Rahmen einer Vorlesung mit dem Titel „Geometrie und Computer" (Sekundarstufe I) geschrieben worden sind. Die Examensarbeit besteht zum einen aus einer Weiterentwicklung dieser Library und zum anderen aus einer Beschreibung des Aufbaus der zweidimensionalen Elementar- und Koordinatengeometrie, die im Rahmen dieser Vorlesung behandelt worden ist. Dieser Text könnte später in Form eines interaktiven Lehrmaterials aufbereitet werden.

Zum Konzept der Library

Die Library `geometrie` dient dazu, Geometrie in der Ebene mit Hilfe eines Computer Algebra Systems zu veranschaulichen. Das kann einerseits mit Hilfe von Visualisierungen der geometrischen Sachverhalte als auch durch symbolisches Beweisen durch Anwendung von algebraischen Umformungen realisiert werden.

Innerhalb dieser Library sind verschiedene Datentypen mit Hilfe von Domains (siehe z.B. [3]) implementiert. Die wichtigsten dieser Datentypen sind:

- `Punkt` – dient zur Beschreibung eines Punktes,
- `Gerade` – dient zur Beschreibung einer Gerade,
- `Kreis` – dient zur Beschreibung eines Kreises,
- `Dreieck` – dient zur Beschreibung einer Dreieckes, und
- `Kurve` – dient zur Beschreibung einer Kurve.

Allen diesen Datentypen sind Methoden zugeordnet, die angeben, wie eine Instanz dieses Typs erzeugt, ausgegeben oder gezeichnet werden soll.

Beispiele

Beispiel 1 *Konstruktion des Umkreises eines Dreiecks.*

Ein Dreieck wird hierbei durch drei Punkte beschrieben, die wir zunächst definieren:

```
>> A:= Punkt(-4,2); B := Punkt(6,-4); C := Punkt(4,8);
```

Das entsprechende Dreieck erhalten wir dann mittels des Befehles

[2] Universität GH Paderborn, `fritz@uni-paderborn.de`

```
>> D:= Dreieck(A,B,C):
```

Um den Umkreis eines Dreieckes zu berechnen, benötigt man bekannterweise den Schnittpunkt der drei Mittelsenkrechten sowie den Abstand zwischen diesem Schnittpunkt und einem Eckpunkt des Dreiecks. Das wollen wir anhand der Dreiecksseite, welche die Punkte A und B des Dreiecks verbindet, explizit vorführen. Dazu bilden wir nun zunächst die Dreiecksseite s:

```
>> s:= Strecke(A,B):
```

Dann berechnen wir den Mittelpunkt dieser Strecke durch den Befehl:

```
>> Mittelpunkt:= Geometrie::Mittelpunkt(s):
```

Die Mittelsenkrechte der Strecke s ergibt sich wie folgt:

```
>> Senkrechte:= Geometrie::Mittelsenkrechte(s):
```

die wir uns mit Hilfe des folgenden Befehles visualisieren lassen können:

```
>> Geometrie::zeichne([A],[B],[Mittelpunkt,rot],[s,gelb],[Senkrechte,gruen]);
```

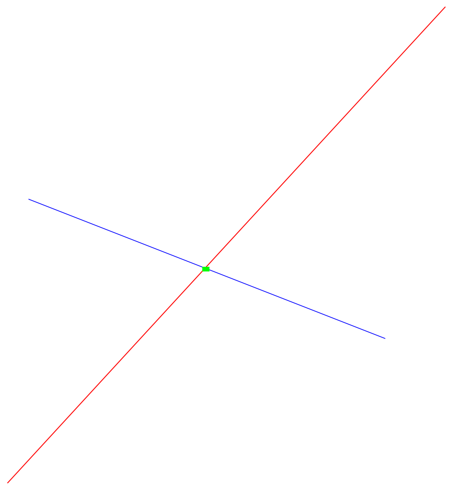

Abbildung 1: Konstruktion der Mittelsenkrechte eines Dreiecks

Um nun den Umkreis des gegebenen Dreiecks D zu berechnen, führt man diese Berechnungen für alle Seiten des Dreiecks durch. Die entsprechenden Befehle sind in einer Prozedur `zeichneUmkreis()` zusammengefaßt.

```
>> zeichneUmkreis(D):
```

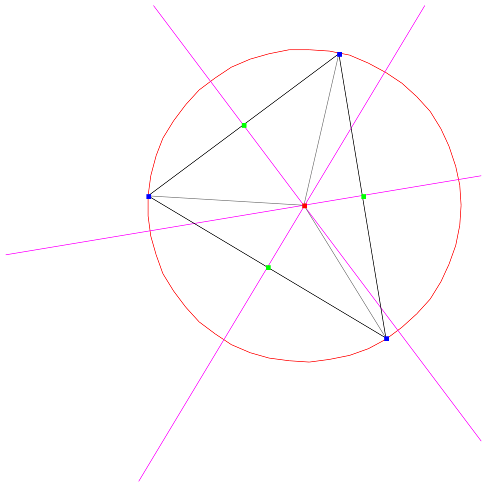

Abbildung 2: Konstruktion des Umkreises eines Dreiecks

Beispiel 2 *Konstruktion des Inkreises eines Dreiecks.*

Wiederum definieren wir zunächst das Dreieck D durch seine drei Eckpunkte A, B und C:

```
>> A:= Punkt(-4,2); B:= Punkt(6,-4); C:= Punkt(4,8);
>> D:= Dreieck(A, B, C):
```

Der Mittelpunkt des Inkreises eines Dreiecks ergibt sich gerade als Schnittpunkt der drei Winkelhalbierenden des Dreiecks D. Wir wollen hier explizit die Konstruktion der Winkelhalbierenden zwischen den Dreiecksseiten AB und AC vorführen:

```
>> b:= Halbgerade(A,B):
>> c:= Halbgerade(A,C):
>> x:= Winkelfeld::Winkelhalbierende(Winkelfeld(b,c)):
>> Geometrie::zeichne([A,rot],[B,rot],[C,rot],[b,gruen],[c,gruen],[x,gelb]):
```

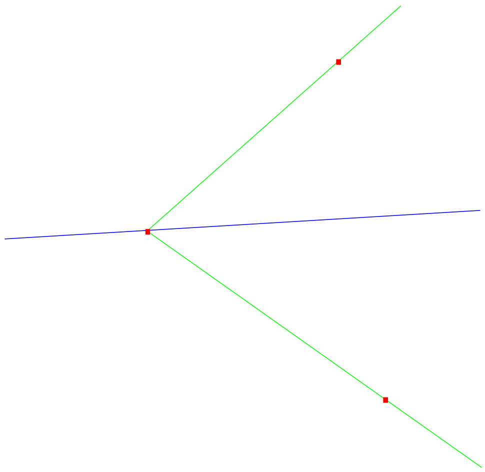

Abbildung 3: Konstruktion der Winkelhalbierenden

Führt man diese Berechnungen wiederum für alle Dreiecksseiten durch, so erhält man schließlich die drei Winkelhalbierenden, deren Schnittpunkt man berechnen kann. Ferner ist es möglich, den Abstand des Mittelpunktes zu den Seiten des Dreiecks zu berechnen. Diese Berechnungen sind in einer Prozedur `zeichneInkreis()` zusammengefaßt:

```
>> zeichneInkreis(D):
```

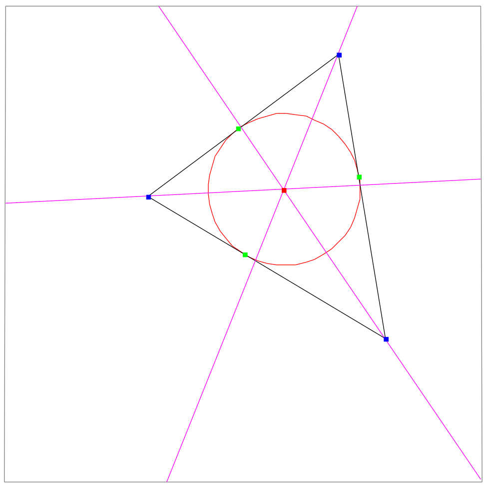

Abbildung 4: Konstruktion des Inkreises eines Dreiecks

Vorträge und Kurse über MuPAD

Seit dem letzten Jahr wurden folgende Vorträge und Präsentationen von *MuPAD* in Schulen und auf Lehrerfortbildungen gegeben:

Institut für Lehrerfortbildung Mühlheim / Ruhr:

- Vortrag und Software-Präsentation von *MuPAD*.
- Vortrag und Software-Präsentation von *MuPAD* im Rahmen des EDV-Benutzer-Seminars 95.
- Dreitägiger MuPAD Kurs mit dem Titel: *MuPAD – Eine methodische Bereicherung des Mathematikunterrichts*. Dieser Kurs richtete sich an Gymnasiallehrer der Sekundarstufe II und hatte zum Ziel, die Bedienung und Programmierung von *MuPAD* zu lernen. Wiederholungen und Fortsetzungen dieses Kurses sind geplant.

Landesinstitut für Schule und Weiterbildung, Soest: Vortrag und Software-Präsentation von *MuPAD*.

Hessisches Institut für Lehrerfortbildung, Weilburg: Vortrag und Software-Präsentation von *MuPAD* auf einer Mitarbeitertagung.

Lichtenberggymnasium, Kassel: Vortrag über Computer-Algebra vor einer 13. Schulklasse mit den Leistungskursen Mathematik und Informatik.

Im Oktober dieses Jahres wird an der Universität GH Paderborn ein einwöchiger Kurs zur Einführung in die Programmierung von *MuPAD* angeboten. Der Kurs besteht aus einer täglichen ca. zweistündigen Vorlesung und praktischen Übungen am Computer. Die Kursteilnehmer sollen in die Benutzung und die Programmierung von *MuPAD* anhand von Anwendungsbeispielen eingeführt werden.

Internetadressen

Interaktive Materialien:

- The Brooks/Cole Home Page: `http://www.thomson.com/brookscole.html`
- MathSource Home Page: `http://www.wri.com/mathsource/`
- Maple's Share Library: `ftp://ftp.inf.ethz.ch/pub/maple/`

Projekte zur Erstellung interaktiver Materialien:

- Transitional Maths Project at Imperial College London: `http://othello.ma.ic.ac.uk/`

Adressen zu Computer-Algebra und Bildung:

- CAIN Germany - Educational Archive: `http://math-www.uni-paderborn.de/CAIN/EDU/index.html`
- CAIN Europe - Educational Archive: `http://www.can.nl/Education/`
- Zentrale für Unterrichtsmedien: `http://132.230.36.11/cgi-bin/hoturls?cas`

Literatur

[1] K. Drescher. *Axioms, Categories and Domains.* Technischer Report Nr. 1. Universität GH Paderborn, 1995.

[2] K. Drescher. *The constructors of the domains package.* Technischer Report Nr. 12. Universität GH Paderborn, 1995.

[3] H. Naundorf. *Domains.* mathPAD Vol. 4, No. 1. Universität GH Paderborn, 1994.

[4] F. Postel. *The Linear Algebra Package "linalg".* Technischer Report Nr. 9. Universität GH Paderborn, 1995.

[5] F. Postel. *Ein interaktives Lehrbuch zur Linearen Algebra.* Bestandteil der Diplomarbeit. Universität GH Paderborn, 1996.

The Computer Algebra Information Network

Thierry Van Effelterre
Riaca, Computer Algebra Nederland, `vaneff@can.nl`[3]

CAIN*, the Computer Algebra Information Network, is a World-Wide-Web server providing all kinds of information about Symbolic and Algebraic Computation which is set up by eight research teams in the framework of the European SAC Network and maintained by* Riaca — Computer Algebra Nederland *in Amsterdam.*
CAIN*'s Home page is at the URL:* `http://www.can.nl/`

What can you find in CAIN ?

CAIN contains the following categories of information:

- **Announcements**:
  - Conferences and Workshops:
    A list of the conferences and workshops that will take place during the next months as well as information about the past conferences. The conferences and workshops are also sorted by month and by country.
  - Software releases.
  - Job opportunities.
  - New Books.
- **The News**:
  Links pointing towards various *electronic newsletters* and *CAIN's Bulletin Board*: a place where *CAIN* users may freely exchange their remarks, questions or suggestions about *Symbolic and Algebraic Computation.*
- **Newcomers in Computer Algebra**:
  Tutorials and courses, interactive demos and examples of outputs from *Computer Algebra* programs.
- **Teams and People**:
  Links pointing towards the Home pages of the teams dealing with *Symbolic and Algebraic Computation* in the world (by country) and an address list of individuals. The entries of the address list contain general as well as scientific information. A search engine gives the means to find entries in the address list via the name or a keyword.
- **Education**:
  Courses and educational material.

[3] Authors address: Thierry Van Effelterre, Riaca, Computer Algebra Nederland, Universiteit van Amsterdam, Nieuwe Achtergracht 170, 1018 WV Amsterdam, Nederland, e-mail: `vaneff@can.nl`

- **Applications**:
  Links pointing towards various applications of *Symbolic and Algebraic Computation.* The fields that are presently covered are
  - Artificial Intelligence
  - Chemistry
  - Economics and Finance
  - Mathematics
  - Physics
- **Systems and Packages**:
  Contains information about the softwares dealing with *Symbolic and Algebraic Computation*. There are presently information about 20 *general purpose* softwares, 17 *experimental softwares* and 79 *special purpose* softwares about
  - Commutative, non commutative algebra and algebraic geometry
  - Differential equations
  - Finite elements analysis
  - Group theory
  - High energy physics
  - Number theory
  - Tensor calculus
- **Literature**:
  Information about articles, books, journals, preprints, PhD theses and technical reports, proceedings and electronic journals. This section also includes a search engine that gives the means to find books via a keyword and a facility to create new bibliographic entries in the *BibTex* format.
- **The Virtual Computer Algebra Library**:
  Last but not least, *The Virtual Computer Algebra Library* is a new facility to help you finding information about a few important topics in *Computer Algebra*. For each topic, you will find links pointing towards tutorials, algorithms, software implementations, applications and bibliographies. Presently, the topics covered are
  - Groebner bases
  - Real solutions of polynomial systems

  The Virtual Computer Algebra Library will soon incorporate more topics like Robotics and polynomial system solving, Parametrization of curves, Quantifier elimination, Symbolic integration and Factorization.

Building up CAIN together

Keeping information up to date is a crucial issue on the World-Wide-Web. This is the reason why we try to keep the information on the servers of the teams and the individuals that provide it. Little databases will be maintained in a few central locations for announcements only. *CAIN* gives the means to efficiently and quickly get the information one is looking for but those who provide the information (more specifically the URL's of the corresponding pages) maintain their own Web pages. We are presently implementing an easy way to gather the URL's of Web pages via fill-in forms by category of information (announcements, bibliographies, ...). Some fill-in forms are already available in *CAIN*. The gathering process via forms will be completely implemented in the Fall 1996.

Eugene Fiume: Scientific Computing

Holger Naundorf
Universität GH Paderborn, snoo@uni-paderborn.de

Das Buch von Eugene Fiume, „Scientific Computing, Eine Einführung in numerische, graphische und symbolische Methoden mit Beispielen in Maple und C“ (364 Seiten, 169 Abb, DM 59,00, ISBN 3-920993-22-5), hat als Ziel, die Wechselwirkung zwischen mathematischer Modellierung und Berechnung an einfachen aber aussagekräftigen Beispielen aufzuzeigen.

In dem oben genannten Buch geht es insbesondere darum, die diskrete Darstellung kontinuierlicher Phänomene zu untersuchen und verständlich zu machen. Maple wird dabei hauptsächlich dazu benutzt, Sachverhalte zu visualisieren, aber auch die symbolischen Möglichkeiten werden zum Beweis einiger Zusammenhänge angewandt.

Das Ziel dieses Buches ist nicht die Erläuterung mathematischer Formalismen, sondern die Vermittlung von Intuitionen und das Aufzeigen von (für den Laien nicht immer offensichtlichen) Zusammenhängen zwischen verschiedenen Bereichen der Mathematik und den Ingenieurwissenschaften.

Zunächst werden bekannte mathematische Konzepte wie Zahlensysteme, Vektoren und Funktionen eingeführt. Danach wird auf die nicht so bekannte Darstellung von Funktionen, vor allem auch auf die für die Visualisierung besonders wichtige Parameterdarstellung eingegangen.
Nachfolgend werden verschiedene Konzepte der Interpolation, der Approximation und der Abtastung von Funktionen vorgestellt sowie die damit verbundenen Probleme aufgezeigt. Insbesondere wird dabei auch das fundamentale Abtasttheorem erläutert. Dies führt auf natürliche Weise zur Untersuchung von Filtern.
In der anschließenden Untersuchung von rechnerischer Integration und Reihenapproximation werden sehr schön die Zusammenhänge mit der Interpolation und der Approximation ausgeführt.
Das Buch wird durch eine Untersuchung der Nullstellenbestimmung von Funktionen abgerundet.

Insgesamt ist das Buch sehr interessant, flüssig und anschaulich geschrieben, und die Zusammenhänge zwischen den einzelnen Themengebieten werden sehr gut aufgezeigt. Durch eine Vielzahl von Aufgaben verschiedener Schwierigkeitsstufen wird das Buch sinnvoll ergänzt — eine Angabe von Musterlösungen allerdings hätte sicher vielen Lesern weitere sinnvolle Einsichten gewährt. Wegen des fehlenden Formalismus wendet sich das Buch hauptsächlich an Studierende der ersten Semester, für diese ist das Buch jedoch eine sinnvolle Ergänzung.
Aber auch für Hochschullehrer ist das Buch interessant als Anregung zum Einsatz von Computeralgebra innerhalb von Vorlesungen.

Kritisch anzumerken ist, daß die Anzahl der in C geschriebenen Beispiele verschwindend gering ist, so daß ein Verzicht auf diese Programmiersprache angebracht gewesen wäre.

Da dieses Buch großenteils von dem spielerischen Umgang des Lesers mit den Beispielen und den Aufgaben lebt, würde der Nutzen und die Qualität noch erhöht werden, wenn dem Buch eine elektronische Version beiläge, so daß Beispiele und Musterlösungen einfach durch Mausklick in das Computeralgebra System übertragen, dort ausprobiert und verändert werden könnten.

Surfin' the Web

Interessante, nützliche und kuriose Plätze im „global village“.

Für die Tips in dieser Ausgabe danken wir Dr. Oliver Kluge[1] (kl), Frank Postel[2] (po) und Ralf Hillebrand[3] (hi).

Computer Algebra

`http://math-www.uni-paderborn.de/CAIN/index.html`
Computer Algebra Information Network. (kl)

`http://symbolicnet.mcs.kent.edu/`
Symbolic Mathematical Computation Information Center. (kl)

`http://info.risc.uni-linz.ac.at:70/1/local`
Research Institute for Symbolic Computation (RISC-Linz). (kl)

`http://www.uni-karlsruhe.de/~CAIS/`
Computer Algebra Fachgruppe Karlsruhe. (kl)

`http://www.can.nl/~abbott/OpenMath/index.html`
Home page of OpenMath. (po)

Computer Algebra Systeme

`http://www.nag.co.uk`
Axiom's home page. (kl)

`http://www.macsyma.com`
Macsyma's home page. (kl)

`http://www.maplesoft.com`, `http://daisy.uwaterloo.ca`
Maple's home page. (kl)

`http://info.risc.uni-linz.ac.at:70/0h/sca/software/pmaple`
||MAPLE||'s home page. (kl)

`http://www.wolfram.com`
Mathematica's home page (kl)

`http://math-www.uni-paderborn.de/MuPAD`
MuPAD's home page. (kl)

`http://www.rrz.uni-koeln.de/REDUCE`
Home page of REDUCE (kl)

[1] olli@uni-paderborn.de
[2] frankp@uni-paderborn.de
[3] tonner@uni-paderborn.de

Mathematik

http://math.nist.gov/HotGAMS/
GAMS, the Guide to Available Mathematical Software, is a cross-index and virtual repository of mathematical and statistical software useful in science and engineering. For more information see background on the Gams project. (kl)

http://www.math.psu.edu/OtherMath.html
Mathematics Information Server. (kl)

Nützliches

http://www.springer.de/catalog/catalog-files/catalog-db.html
Index-Suche im Katalog des Springer Verlag's. (po)

http://lusk1.mines.edu/hjjou/linux_prg.html
Scientific Applications on Linux: Eine Übersicht über wissenschaftliche Software (nicht nur für Linux). (hi)

http://www.uni-paderborn.de/~SciTeXt/
SciTeXt ist ein freier Wordprocessor, der an der Univ. Paderborn von einigen Studenten entwickelt wird. (hi)

Entspannung

http://www.sandiegozoo.org/
Virtueller Rundgang durch den Zoo von San Diego (kl)

http://www.metmuseum.org/
Metropolitain Museum of Art (kl)

http://www.techexpo.com/
TechExpo — Hi-Tech in Engineering and Medical/Life Sciences. (po)

http://www.uni-paderborn.de/paderborn/kino/kino.html
Aktuelle Kino-Übersicht für Paderborn und Umgebung. (po)

http://www.ruhr-uni-bochum.de/~schaelcz/dinner.html
Dinner for two: Eine feine Menü-Sammlung für gesellige Stunden zu Zweit. (hi)

http://www.atcom.co.at/atcom/sref.htm
Der ultimative Smiley-Server. (hi)

Kalender

DISCO '96

International Symposium on Design and Implementation of Symbolic Computation Systems, Karlsruhe, 18.–20. September 1996

Arbeiten sind einzureichen bis:
19. Februar 1996

Kontakt:
WWW: `http://iaks-www.ira.uka.de/iaks-calmet/conf/disco.html`

EuroAIED

European Conference on AI in Education, Fundacao Caloustre Gulbenkian, Lisabon, Portugal, 30. September–2. Oktober 1996

Arbeiten sind einzureichen bis:
29. Februar 1996

Kontakt:
WWW: `http://www.cbl.leeds.ac.uk/conferences/EuroAIED`

INFORMATIK'96

Joint Annual Meeting of the German Gesellschaft für Informatik and the Austrian Computergesellschaft, Klagenfurt, Östereich, 25.–27. September 1996

Letzte Meldungen

Kooperation zwischen der MuPAD-Gruppe und INRIA

Kürzlich hat die MuPAD-Gruppe eine förmliche Vereinbarung mit INRIA (Institute Nationale de Recherche en Informatique et en Automatique), vertreten durch den Präsidenten von INRIA-Lorraine, über die Lizensierung des Computeralgebrasystems MuPAD für alle Bereiche von INRIA getroffen.

Das Abkommen regelt die Zusammenarbeit im Bereich der Computeralgebra. Der neue Kooperationspartner verpflichtet sich, die Entwicklung von Algorithmen für das MuPAD-System zu fördern und zu koordinieren.

MuPAD at the University H. Poincare, Nancy

Alain Mirgaux[1] von der Universität H. Poincare in Nancy benutzte eine sehr frühe Version von MuPAD für Windows für einen Kurs. Er notierte:

> During the first semester of 1996 we used MuPAD with students in their second year of scientific studies.
>
> About 100 students were interested in this optional course of 60 hours. We began to learn MuPAD's syntax and in the second part we made an introduction to numerical calculation:
>
> - influence of rounding errors,
> - Gauss's elimination for linear systems,
> - fixed point algorithm for non linear equations,
> - Euler's methods for differential equations.
>
> We had an incomplete beta-version of MuPAD for Windows 3.11 (special thanks to P. Zimmermann and to S.O. Ivanov for their help and for their good advices). With our PC-networks we can't use the linear algebra package, so we have to program with MuPAD like with Turbo-PASCAL. In spite of these inconveniences, the students have learned the differences between a programming langage and a formal calculus software.

Beta Test für MuPAD Version 1.3

Am 6. Mai 1996 wurde ein offizielles Beta-Release für MuPAD Version 1.3 veröffentlicht. Die CA-Gemeinde war für drei Monate aufgerufen, dieses Beta-Release, welches sehr viele Verbesserungen gegenüber der Version 1.2.2 aufweist, zu testen.

Um die Tester anzuspornen, wurden für die drei „erfolgreichsten" (sprich längsten) Fehlerlisten Preise ausgelobt: Ein „MuPAD User Manual", eine MuPAD CD sowie ein MuPAD T-Shirt.

Die Teilnahme war so rege, daß wir Mühe hatten, mit den Rückmeldungen einigermaßen auf dem Laufenden zu bleiben. (Einige aus der MuPAD-Gruppe stöhnten doch etwas unter der Last der hunderte von Mails, die zu beantworten waren.)

[1] `Alain.Mirgaux@loria.fr`

mathPAD

Den ersten Preis errang Michael Ulbrich, den zweiten Francois Maltey und den dritten „last but not least“ Michael Wester.

Dank der regen Teilnahme konnten wir zahllose Fehler beseitigen, was der Version 1.3 sehr zugute kommen wird. Wir möchten uns an dieser Stelle nochmals bei all den begeisterten Testern herzlich bedanken!

Internet Explorer

Eine kleine Firma Namens Synet aus Chicago kämpft gegen Microsoft um die Rechte am Produktnamen „Internet Explorer“. Angeblich will Netscape in Synet investieren. *(Quelle: Die Zeit vom 23.8.96)*