APL-Journal

A Programming Language

1-2/2020

APL-Germany e.V.

Doppelnummer

Nr. 1-2 2020

Jahrgang 39

ISSN - 1438-4531

RHOMBOS-VERLAG

Liebe Mitglieder von APL Germany,
liebe APL-Freunde,

heute erhalten Sie die neue Ausgabe des APL-Journals. Es erscheint in einer Zeit der Hoffnung auf einen besseren Umgang mit der Corona-Pandemie. Der Vorstand ist optimistisch, die Herbsttagung am 22. und 23. November in Berlin wieder als Präsenzveranstaltung durchführen zu können.

Die beiden letzten Tagungen von APL-Germany und der GSE Working Group APL fanden im Online-Modus mit teilweise über 40 Teilnehmern statt. Aus diesen Tagungen stammen auch die Artikel dieses Journals, bei deren Lektüre ich Ihnen viel Freude wünsche.

Mein Artikel befasst sich mit der Implementierung der Bruchrechnung mit ganzen Zahlen, ganzen Gauß-Zahlen und Polynomen. Michael Baas denkt über deutsche Ausdrücke für die APL-Symbole nach, Hanfeng Cheng und Wai-Mee Ching berichten über ihr ELI-Projekt. Adám Brudzewsky stellt Ideen zu einer Text-Notation für Datenfelder vor und Martin Barghoorn nennt Beispiele von Strukturen und Sprachelementen in R, die er in APLvermisst.

Auch unsere Mitgliederversammlung im Mai fand online statt. Zu dem verschickten Protokoll gab es keine Einwände, es ist damit beschlossen.

Nach über 50 Jahren Entwicklung und Betreuung von APL2 durch IBM endet diese Ära in diesem Jahr. Ich hoffe auf eine gute Weiterentwicklung der Sprache durch die Firma Log-On Software und freue mich, sie als institutionelles Mitglied des Vereins begrüßen zu dürfen!

Außerhalb von APL geht die Entwicklung natürlich auch weiter. Als besonders erwähnenswert halte ich die Einweihung des ersten Quantencomputers ,,im industriellen Kontext“ in Deutschland (mit 27 Qubits) durch IBM und Fraunhofer-Gesellschaft im Juni in Ehningen. Wann läuft APL zum ersten Mal auf einem Quantencomputer?

Ich wünsche Ihnen viel Gesundheit und freue mich auf ein Wiedersehen in Berlin!

Dieter Kilsch

Kontakt:

Prof. Dr. Dieter Kilsch
E-mail: d.kilsch@th-bingen.de

INHALT

Implementing Quotients of Euclidean Rings Using Nested Arrays

Dieter Kilsch

Dedicated to John Scholes[1]

Contents

This paper describes a general way how to implement the operations of quotient fields of Euclidean rings in `APL2`. Some examples in number fields will demonstrate this work:

(a) The quotient field of the ring of integers $\mathbb{Z}$ is the field of rational numbers.

(b) The quotient field of the ring of Gaussian integers $\mathbb{Z}[i]$ is the complex number field $\mathbb{Q}[i] = \mathbb{Q}(i)$ of rational points in the Gaussian plane.

(c) The quotient field of the polynomial ring $\mathbb{K}[x]$ over a field K is the field of rational functions $\mathbb{K}(x)$. This theoretically covers the polynomial rings $\mathbb{Q}[x]$, $\mathbb{R}[x]$ and $\mathbb{C}[x]$. But as all machine numbers have rational real and imaginary part, calculations with a computer always take place inside the field $\mathbb{Q}[i]$ and its ring of polynomials.

[1] I wrote some of the functions in the early 1990s and came back to this subject in the early 2010s when I wrote the first draft of this article as a letter to John Scholes. He thought about implementing calculations with fractions of integers into Dyalog APL and wanted to know more about the mathematics behind this subject. I am thankful to some really fruitful discussions on this subject with him.

As the polynomial ring $\mathbb{Z}[x]$ is not Euclidean, the results do not apply to this ring although its quotient field is $\mathbb{Q}(x)$.

The `APL2`-workspaces with the functions performing the ring, quotient field and matrix operations are contained in `KhAlZ, KhAlZi, KhAlPoly, KhAlEukl, KhAlQuot, KhAlMatr`.

1 Mathematical Background

To implement calculations with fractions we have to provide functions which do addition, subtraction, multiplication and division with fractions. The way to do it is well known from the definition of rational numbers and can be done with every integral domain ([4, p. 114]).

In this paper I am looking more closely to special integral domains, the Euclidean rings, which cover a wide area of rings with relevance in applications. These rings have a unique factorization into prime numbers, a greatest common divisor of two (or more) numbers and a normalized representation of the fractions in their quotient fields.

Examples are the integers, Gaussian integers and polynomial rings over any field which we will look at further down.

Definition 1.1 (Euclidean Ring or Euclidean Domain) *[4, 7.1, p. 120ff] A commutative ring $(R, +, \cdot)$ with identity element 1 is an* Euclidean ring *if*

(a) R does not have zero divisors: $\forall r, s \in R\backslash\{0\} : \; r \cdot s \neq 0$.

(b) There is a function $\varphi : R\backslash\{0\} \to \mathbb{N}_0$ with the following properties:[2]

(b1) $\forall r, s \in R\backslash\{0\} : \; \varphi(rs) \geq \max\{\varphi(r), \varphi(s)\}$ This function is denoted Euclidean function *or* Euclidean value.

(b2) $\forall p, q \in R\backslash\{0\} \; \exists r, s \in R : \; p = sq + r \;\wedge\; (r = 0 \;\vee\; \varphi(r) < \varphi(q))$ This algorithm is called Euclidean algorithm.

It is easy to show that all Euclidean rings are principle ideal rings and these are unique factorization domains, see [4]. On the other hand $\mathbb{Z}[x]$ is a unique factorization domain but has non-principal ideals, e.g. $\mathbb{Z}[p, x]$ for any prime number $p \in \mathbb{Z}$ ([4, Es. 7.2(2), 7.3.8]). The same is true for $\mathbb{Z}[x, y]$ and all rings of polynomials in two or more variables over a field.

There are non-Euclidean rings of integral elements[3] in a number field over $\mathbb{Q}$: [5] shows, that $\mathbb{Q}[\sqrt{-d}\,]$ with $d = 1, 2, 3, 7, 11$ are the only complex number fields, whose integral rings, which contain $\mathbb{Z}[-d]$, are Euclidean. For $d = 19, 43, 67, 163$ these rings are non-Euclidean but principle ideal domains. [4] states that $\mathbb{Z}[\sqrt{-3}\,]$ is not a unique factorization domain: $4 = 2 \cdot 2 = (1 + \sqrt{-3}) \cdot (1 - \sqrt{-3})$. Hence it is properly contained in the ring of integral elements of $\mathbb{Q}[-d]$.

When thinking of rational fractions one normally expects denominators to take positive values. In general Euclidean rings, an analogues restriction is achieved with the help of the character or Euclidean norm.

Definition 1.2 (Units and Euclidean Norm or Character) *A* unit *of a ring is an element through which one can divide any element of the ring or equivalently all elements which do divide 1. The set af all units of a ring R is denoted by $R^\times$, it is a multiplicative group.*

A function $\psi : R \mapsto R^\times$ which is $R^\times$-linear ($\forall r \in R \forall e \in R^\times \; \psi(e \cdot r) = e\psi(r)$) is called a character *or* Euclidean norm *of R.*

Definition 1.3 (Field of Quotients of an Integral domain) *([4, p. 114]) Let I be an integral domain. Then*

$$Q(I) := \left\{ \frac{a}{b} \;\middle|\; a \in I \wedge b \in I\backslash\{0\} \right\} \qquad \text{with} \qquad \frac{a_1}{b_1} = \frac{a_2}{b_2} \iff a_1 b_2 = a_2 b_1$$

with basic operations as defined in $\mathbb{Q}$ is a field, the field of fractions.

[2] $\mathbb{N}_0$ is the set of all non-negative integers.

[3] Zeros of integer polynomials with leading coefficient 1 in the given number field.

In the quotient field of an Euclidean ring with a character, shortening a fraction by the character of its denominator yields a fraction whose denominator has character 1.

2 General Implementation of Ring and Quotient Operations

Functions working with ring elements and their fractions use the functions `Δa` to add, `Δs` to subtract, `Δm` to multiply and `Δd` to divide fractions in quotient fields.

2.1 Ring Operations in Euclidean Rings

In an Euclidean ring we can add (`Δa`), subtract (`Δs`) and multiply (`Δm`). These are the ring operations addition (`RΔa`) and multiplication (`RΔm`) as given by a specific ring. Subtraction is addition of the negative value. These functions are provided by `KhAlEukl` and shown in A.4.

The greatest common divisor of two integers a and b is equal to the greatest common divisor of $\min(|a|, |b|)$ and the remainder of dividing $\max(|a|, |b|)$ by $\min(|a|, |b|)$, This division with remainder is calculated using the Euclidean algorithm of the given ring. If one of the numbers is zero, the greatest common divisor is set to the other. The function `ggT` calls itself recursively with $\min(|a|, |b|)$ and the remainder until the remainder is zero.

To get an effective algorithms it is necessary to get the absolute value of the remainder in the above algorithm as small as possible. The Euclidean algorithm thus must provide a result with minimal absolute value of the remainder, if the ring offers a chance to do so. The effect is shown in Figure 1 for $\mathbb{Z}$ which takes a remainder in the interval

(a) $[0, n-1]$ in the first example and

(b) $\left[-\lfloor\frac{n}{2}, \lfloor\frac{n}{2}\right]$ in the second.

when dividing through a number n.

The product of the lowest common multiple and the greatest common divisor is the product of the two numbers. So

$$\operatorname{lcm}(a, b) = \frac{ab}{\gcd(a, b)} \ .$$

The function `kgV` uses the fact, that if division of two integers results in zero remainder, the quotient equals the first component of the Euclidean algorithm.

All functions work on nested ring elements.

$$\begin{array}{rcl} 190 & = & 2 \cdot 68 + 54 \\ 68 & = & 1 \cdot 54 + 14 \\ 54 & = & 3 \cdot 14 + 12 \\ 14 & = & 1 \cdot 12 + 2 \\ 12 & = & 6 \cdot 2 + 0 \end{array} \qquad \begin{array}{rcl} 190 & = & 3 \cdot 68 + (-14) \\ 68 & = & (-5) \cdot (-14) + 2 \\ 12 & = & 6 \cdot 2 + 0 \end{array}$$

Figure 1: Calculating $\gcd(190, 68)$

2.2 Field Operations in Quotient Fields

Every Euclidean ring has a unique quotient field defined in the same way we are calculating in the field of rational numbers $\mathbb{Q}$. A quotient is represented by a nested two-element vector consisting of the *nominator* and *denominator*, so the depth of a quotient is one higher than the depth of a ring element. Zero is the quotient of ring-zero and ring-one; one is the quotient of ring-one and ring-one. These are initialized in the function `Δinit`.

Calculating with fractions one has to embed the ring into its quotient field. This map is given by `Δq`. At the end a quotient has to be displayed as a fraction, done by `Δdar`, see A.5.

The functions performing the fundamental arithmetic operations for quotient fields of Euclidean rings and are defined in `ΔKhAlQuot`:

`Δa`: The sum of two fractions is equal to the quotient of the sum of the crosswise products of nominator and denominator and the product of the two denominators.

`Δs`: The difference of two fractions is equal to the sum of the first with the negative of the second fraction. To get the negative of a fraction we change sign of the nominator.

`Δm`: The product of two fractions is the quotient of the product of the nominators and the product of the denominators.

`Δd`: The quotient of two fractions ist calculated as the product of the first and the inverse of the second fraction. The inverse of a fraction is achieved by interchanging nominator and denominator.

All results have to be shortened at the end. These functions are shown in A.5.

2.3 Matrix Calculations

The workspace `KhAlMatr` provides some functions to perform matrix calculations using the basic operations `Δa`, `Δs`, `Δm` and `Δd` of the quotient field as shown in A.6. It contains:

`Algaus`: one step of the Gauss-algorithm,

`Algaut`: complete Gauss-algorithm,

`Aldet`: Calculating the determinant of a square matrix: Checking data and calling `Aldet1`.

`Aldet1`: Calculating the determinant of a square matrix using Lagrange method.

`Alzeim`: displaying a matrix (with fractions) on the screen.

`Alchpo`: Calculating the characteristic polynomial using the determinant.

`Algaus` and `Algaut` provide an option to divide the pivot row by the pivot element and to multiply the next row with this pivot element. This leaves the determinant of the matrix unchanged and thus the bottom right element of the output of `Algaut` contains the determinant for an inserted square matrix. Using the standard Gauss-algotrithm the determinant of the matrix stays unchanged, hence it is the product of the diagonal of the final matrix.

3 Implementations of some Rings and their Quotient Fields

Table 1 contains the Euclidean rings and their quotient fields implemented and described in this paper. The field of rational functions $\mathbb{Q}(x)$ is not covered by these models, but it may be looked at as a subset of $\mathbb{R}(x)$. In these implementations all ring elements are simple or non-simple scalars

Table 1: Euclidean rings and their quotient fields

domain or field	workspace to be loaded	available operations
$\mathbb{Z}$	`KhAlZ`, `KhAlEukl`	`Δa, Δs, Δm`
$\mathbb{Q}$	`KhAlZ`, `KhAlEukl`, `KhAlQuot`	`Δa, Δs, Δm, Δd`
$\mathbb{Z}[i]$	`KhAlZi`, `KhAlEukl`	`Δa, Δs, Δm`
$\mathbb{Q}[i] = \mathbb{Q}(i)$	`KhAlZi`, `KhAlEukl`, `KhAlQuot`	`Δa, Δs, Δm`
$\mathbb{R}[x], \mathbb{C}[x]$	`KhAlPoly`, `KhAlEukl`	`Δa, Δs, Δm`
$\mathbb{R}(x), \mathbb{C}(x)$	`Polynom`, `KhAlEukl`, `KhAlQuot`	`Δa, Δs, Δm, Δd`

of a specified depth depending on the ring. A quotient is a nested vectors of two ring elements, the nominator and the denominator. Its depth is one higher than the depth of a ring element and the depth of a vector or array of elements of the quotient field is one higher again. The routine

`RΔinit` creates zero and unity (one) of the ring `rΔnull` and `rΔeins`. It also sets the variable `rΔt` which contains the depth of all ring elements.

The implementation of ring operations obviously depends on the ring. These functions are:

- `RΔinit`: Initializing the depth of a ring element `rΔt`, its zero element `rΔnull` and its identity element `rΔeins`.
- `rΔdar`: Formatting a ring element.
- `RΔa`: Addition
- `RΔm`: Multiplication
- `RΔef`: Euclidean function, calculating the Euclidean value of a ring element.
- `RΔe`: Euclidean algorithm.
- `RΔc`: Euclidean character or norm: The denominator in a final result should be of a special character, quite often 1.

3.1 Integers and Rational Numbers

All functions are included in the workspace `KhAlZ` and shown in A.1. The Euclidean function or value $\varphi(z)$ is the absolute value $|z|$ of the integer z, its character or norm is its sign. Hence we expect denominators to be positiv.

According to 2.1 the Euclidean algorithm of two numbers r and s to get $p = qs + r$ should yield a remainder r with smallest possible absolute value. Therefore rounding is to the nearest integer, the midpoint between two integers being rounded upwards. But this all can be calculated in two different ways using floor/ceiling or residue:

(a) `r←(p-q×r),r←⌈¯.5+p÷q`,

(b) `((p-r)÷q),r←(-q×(2×|r)>|q)+r←q|p`.

Figure 2 shows that the code in (b) is faster, so it is used in the Euclidean algorithm for $\mathbb{Z}$.

```
      (p q)←(2/1 ¯1×44)(1⌽2 2/1 ¯1×6)
      r←(p-q×r),[1.5]r←⌈¯.5+p÷q
      p,'=',r[;1],'×',q,'+',r[;2],'|',':',[1.5]q|p
 44 =  2 × 6 + 7 |: 2
 44 =  2 × ¯6 + ¯7 |: ¯4
¯44 = ¯2 × ¯6 + 7 |: ¯2
¯44 = ¯2 × 6 + ¯7 |: 4

      (p q)←(2/1 ¯1×45)(1⌽2 2/1 ¯1×6)
      r←((p-r)÷q),[1.5]r←(-q×(2×|r)>|q)+r←q|p
      p,'=',r[;1],'×',q,'+',r[;2],'|',':',[1.5]q|p
 45 =  7 × 6 + 3 |: 3
 45 = ¯8 × ¯6 + ¯3 |: ¯3
¯45 =  7 × ¯6 + ¯3 |: ¯3
¯45 = ¯8 × 6 + 3 |: 3

      (p q)←(6E6⍴44)(6)
      t←⎕ts⋄r←(p-q×r),r←⌈¯.5+p÷q⋄1000⊥¯2↑⎕ts-t
187
      t←⎕ts⋄r←((p-r)÷q),r←(-q×(2×|r)>|q)+r←q|p⋄1000⊥¯2↑⎕ts-t
129
```

Figure 2: The Euclidean algorithm for integers

Example 3.1

(a) After loading the work spaces `KhAlZ, KhAlEukl, KhAlQuot` and initializing the parameters we calculate, see fig. 3

- $\frac{1}{4} + \frac{1}{12}$ and get $\frac{1}{3}$, and display the result using `Δdar`,

- the scalar product $\left\langle \begin{pmatrix} \frac{1}{2} \\ \frac{3}{4} \end{pmatrix}, \begin{pmatrix} \frac{1}{5} \\ \frac{2}{7} \end{pmatrix} \right\rangle$ and get $\dfrac{11}{35}$.

(b) We calculate the determinant of a matrix of rational numbers, see Figure 4:

$$\begin{vmatrix} 1 & -1 & \frac{5}{3} \\ \frac{-5}{4} & \frac{-8}{5} & \frac{3}{2} \\ \frac{9}{7} & \frac{3}{8} & \frac{16}{9} \end{vmatrix} = \frac{-2357}{480} .$$

(c) Using the Gauss-algorithm we solve the system of linear equations

$$\begin{pmatrix} 1 & -2 & 5 \\ -5 & -8 & 9 \\ 9 & 3 & 16 \end{pmatrix} \cdot \vec{x} = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}$$

with the solution by hand

$$\left(\begin{array}{ccc|c} 1 & -2 & 5 & 1 \\ 0 & -18 & 34 & 7 \\ 0 & 21 & -29 & -6 \end{array}\right) \rightarrow \left(\begin{array}{ccc|c} 1 & 0 & \frac{11}{9} & \frac{2}{9} \\ 0 & 1 & \frac{-17}{9} & \frac{-7}{18} \\ 0 & 0 & \frac{32}{3} & \frac{13}{6} \end{array}\right) \rightarrow \left(\begin{array}{ccc|c} 1 & 0 & 0 & \frac{-5}{192} \\ 0 & 1 & 0 & \frac{-1}{192} \\ 0 & 0 & 1 & \frac{13}{64} \end{array}\right) .$$

Fig. 5 contains the solution with APL2.

(d) We also solve the system of linear equations $\begin{pmatrix} 1 & -1 & \frac{5}{3} \\ \frac{-5}{4} & \frac{-8}{5} & \frac{3}{2} \\ \frac{9}{7} & \frac{3}{8} & \frac{16}{9} \end{pmatrix} \cdot \vec{x} = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}$, the solution from fig. 6 is $\begin{pmatrix} \frac{-7054}{7071} \\ \frac{73880}{49497} \\ \frac{34551}{16499} \end{pmatrix}$.

```
      ∆init      ⍝ KHAlZ -Eukl -Quot
      1 4 ∆a 1 12
1 3
      ∆dar 1 4 ∆a 1 12
 1/3
      ∆dar (1 2)(3 4)∆a.∆m(1 5)(2 7)
  11/35
```

Figure 3: Calculating with fractions of integers

```
      ∆init ⍝KHAlZ -Eukl -Quot -Matr
      mat←3 3⍴1 ¯2 5 ¯5 ¯8 9 9 3 16
      ∆dar mat←(∆q¨mat)∆d 3 3⍴∆q¨⍳9
   1     ¯1    5/3
 ¯5/4   ¯8/5    3/2
  9/7    3/8   16/9
      ∆dar Aldet mat   ⍝ Det. Laplace
 ¯2357/480
```

Figure 4: Determinant of a rational matrix

```
      ∆init ⍝ KHAlZ -Eukl -Quot -Matr
      mat←3 3⍴1 ¯2 5 ¯5 ¯8 9 9 3 16
      2⊃2 Algaut mat,1 2 3
 1 1  0 1   0 1   ¯5 192
 0 1  1 1   0 1   ¯1 192
 0 1  0 1   1 1   13 64
      1 Alzeim 2⊃2 Algaut mat,1 2 3
┌ 1 0 0 │ ¯5/192 ┐
│ 0 1 0 │ ¯1/192 │
└ 0 0 1 │  13/64 ┘
```

Figure 5: System of linear eq. with rational solution

```
      ∆init ⍝ KHAlZ -Eukl -Quot -Matr
      mat←3 3⍴1 ¯2 5 ¯5 ¯8 9 9 3 16
      ∆dar mat←(∆q¨mat)∆d 3 3⍴∆q¨⍳9
   1     ¯1    5/3
 ¯5/4   ¯8/5    3/2
  9/7    3/8   16/9
      1 Alzeim 2⊃2 Algaut mat,∆q 1 2 3
┌ 1 0 0 │  ¯7054/7071 ┐
│ 0 1 0 │ 73880/49497 │
└ 0 0 1 │ 34551/16499 ┘
```

Figure 6: System of linear eq. with rational coefficients

Example 3.2 (Hilbert Matrix[4]) Fig. 7 defines the Hilbert Matrix $mat_{ij} = \frac{1}{i+j}$ and calculates its determinant and its product with its inverse, which is the identity matrix – in theory. Precise calculations with fractions are shown in fig. 8.

```
      ∆init    ⍝ KHAlZ -Eukl -Quot Khmatrix
      11 ¯4⍕mat←÷¯1+(⍳6)∘.+⍳6
 1.000E0    5.000E¯1   3.333E¯1   2.500E¯1   2.000E¯1   1.667E¯1
 5.000E¯1   3.333E¯1   2.500E¯1   2.000E¯1   1.667E¯1   1.429E¯1
 3.333E¯1   2.500E¯1   2.000E¯1   1.667E¯1   1.429E¯1   1.250E¯1
 2.500E¯1   2.000E¯1   1.667E¯1   1.429E¯1   1.250E¯1   1.111E¯1
 2.000E¯1   1.667E¯1   1.429E¯1   1.250E¯1   1.111E¯1   1.000E¯1
 1.667E¯1   1.429E¯1   1.250E¯1   1.111E¯1   1.000E¯1   9.091E¯2
      Det mat
5.367299886E¯18
      11 ¯4⍕mat+.×⌹mat
 1.000E0    0.000E0    0.000E0    2.910E¯11  0.000E0    0.000E0
 1.137E¯13  1.000E0    0.000E0    1.455E¯10  5.821E¯11  0.000E0
¯5.684E¯14  0.000E0    1.000E0    8.731E¯11 ¯2.910E¯11  0.000E0
¯1.137E¯13 ¯1.819E¯12  7.276E¯12  1.000E0    2.910E¯11  1.455E¯11
 1.137E¯13 ¯1.819E¯12  2.910E¯11  0.000E0    1.000E0   ¯1.455E¯11
 0.000E0    0.000E0    7.276E¯12  2.910E¯11  0.000E0    1.000E0
```

Figure 7: Hilbert matrix using standard operations

```
      ∆init    ⍝ KHAlZ -Eukl -Quot -Matr
      ∆dar mat←1,¨¯1+(⍳6)∘.+⍳6    ⍝ with fractions
    1  1/2  1/3  1/4   1/5   1/6
  1/2  1/3  1/4  1/5   1/6   1/7
  1/3  1/4  1/5  1/6   1/7   1/8
  1/4  1/5  1/6  1/7   1/8   1/9
  1/5  1/6  1/7  1/8   1/9  1/10
  1/6  1/7  1/8  1/9  1/10  1/11
      Aldet mat                   ⍝ determinant: Laplace
1 1.863134203E17
      ∆dar inv←0 6↓2⊃2 Algaut mat,∆q 6 6⍴7↑1   ⍝ Gauss
     36      ¯630      3360      ¯7560      7560      ¯2772
   ¯630     14700    ¯88200     211680   ¯220500      83160
   3360    ¯88200    564480   ¯1411200   1512000    ¯582120
  ¯7560    211680  ¯1411200    3628800  ¯3969000    1552320
   7560   ¯220500   1512000   ¯3969000   4410000   ¯1746360
  ¯2772     83160   ¯582120    1552320  ¯1746360     698544
      ∆dar mat ∆a.∆m inv          ⍝ product with inverse
  1  0  0  0  0  0
  0  1  0  0  0  0
  0  0  1  0  0  0
  0  0  0  1  0  0
  0  0  0  0  1  0
  0  0  0  0  0  1
```

Figure 8: Hilbert matrix with fractions

Fig. 9 contains an example with a solution of a system of linear equations. First its determinant is calculates using Laplace's method (`Aldet matq`) and then using the method described in 2.3: During the Gauss algorithm the pivot row is divided by the pivot element and the following line multiplied by that pivot element. Next it shows the system of equations in matrix form and its solution.

[4]The relevance of the Hilbert matrix to demonstrate the effect of accurate computation by fractions has been pointed out by Roger Hui in his talk on his implementation of fractions of integers presented during the Dyaolg'20 conference.

```
      mat←3 3⍴1 ¯2 5 ¯5 ¯8 9 9 3 16
      Δdar mat←(Δq¨mat)Δd 3 3⍴Δq¨⍳9
   1    ¯1   5/3
 ¯5/4  ¯8/5   3/2
  9/7   3/8  16/9
      Δdar Aldet mat                  ⍝ Det. with Laplace
 ¯2357/480
      Δdar Δm/1 1⍉2⊃0 Algaut mat      ⍝ Det. with Gauss
 ¯2357/480
      Δdar ¯1 ¯1↑2⊃0 0 1 Algaut mat ⍝ .. special
 ¯2357/480
      1 Alzeim mat, Δq 1 2 3          ⍝ linear equations
┌    1    ¯1   5/3 │ 1 ┐
│  ¯5/4  ¯8/5   3/2 │ 2 │
└   9/7   3/8  16/9 │ 3 ┘
      1 Alzeim 2⊃2 Algaut mat,Δq 1 2 3    ⍝... solution
┌ 1 0 0 │   ¯7054/7071 ┐
│ 0 1 0 │  73880/49497 │
└ 0 0 1 │  34551/16499 ┘
```

Figure 9: Solving a system of linear equations

3.2 Gaussian Integers and their Quotients

The Euclidean function or value in $\mathbb{Z}[\mathrm{i}]$ is the square of the absolute value of the ring element.[5] Fig. 10 shows that `r×+r` is the quickest way to calculate the Euclidean value. The norm or character depends on the argument angle of the complex number:

```
      r←1000 1000 ⍴(4J17)*⍳9
      t←⎕ts⋄s←10○r*2⋄1000⊥¯2↑⎕ts-t
265
      t←⎕ts⋄s←(10○r)*2⋄1000⊥¯2↑⎕ts-t
94
      t←⎕ts⋄s←(|r)*2⋄1000⊥¯2↑⎕ts-t
47
      t←⎕ts⋄s←r×+r⋄1000⊥¯2↑⎕ts-t
15
```

Figure 10: The Euclidean function in $\mathbb{Z}[\mathrm{i}]$

$$\psi(a+\mathrm{i}\,b) = \left\{\begin{array}{ll} 1 & 0 \le \arg(a+\mathrm{i}\,b) < 90^\circ \\ \mathrm{i} & 90^\circ \le \arg(a+\mathrm{i}\,b) < 180^\circ \\ -1 & 180^\circ \le \arg(a+\mathrm{i}\,b) < 270^\circ \\ -\mathrm{i} & 270^\circ \le \arg(a+\mathrm{i}\,b) < 360^\circ \end{array}\right\} = \left\{\begin{array}{ll} 1 & a > 0 \wedge b \ge 0 \\ \mathrm{i} & a \le 0 \wedge b > 0 \\ -1 & a < 0 \wedge b \le 0 \\ -\mathrm{i} & a \ge 0 \wedge b < 0 \end{array}\right\}$$

As in $\mathbb{Z}$ we have to get an Euclidean algorithm with small remainder and compare different algorithm with different ways of rounding:

(a) According to the APL2 language reference manual we have `q|p is p-q×⌊p÷q+q=0`. So we have to check the usage of the complex floor which assigns a complex number to a neighbouring integral lattice point according to fig. 11. Setting `s←⌊p÷q` and `r←p-q×s` we have $p = q \cdot s + r$ and from $\left|\frac{p}{q} - \lfloor\frac{p}{q}\right| < 1$ we get

$$\varphi(r) \;=\; |p - q\cdot s|^2 \;=\; |q|^2 \left|\frac{p}{q} - s\right|^2 \;<\; |q|^2 \;=\; \varphi(q)\,. \tag{1}$$

This shows that these formulas could be used as Euclidean algorithm but Figure 11 shows that the distance between a point and its related lattice point could be closed to 1, so $\varphi(r)$

[5] I do not see any reason why one always takes the square in the literature instead of just the absolute value.

could be closed to $\varphi(q)$ in (1) an the algorithm to calculate the greatest common divisor converges too slowly, see 2.1.

(b) So we check the function which maps every point to its closest integral lattice point: $\frac{p}{q} = u + \mathrm{i}\,v \in \mathbb{Q}[i]$ has 4 neighbouring lattice points $\lfloor u + \mathrm{i}\lfloor v$, $\lfloor u + \mathrm{i}\lceil v$, $\lceil u + \mathrm{i}\lfloor v$, $\lceil u + \mathrm{i}\lceil v$. If $s = x + \mathrm{i}\,y$ is the closest lattice point we set $r := p - q \cdot s$ and have

$$|x - u| \le \frac{1}{2} \;\wedge\; |y - v| \le \frac{1}{2} \quad \Rightarrow \quad \varphi(r) \le \frac{1}{2}\varphi(q) \qquad \text{(see [4])} \qquad .$$

This method provides a better algorithm to calculate the greatest common divisor. The APL code is

```
s←(1⌷r)+0J1×2⌷r←⌈¯0.5+9 11○p÷q⋄r←s,p-q×s
```

All functions are listed in A.2.

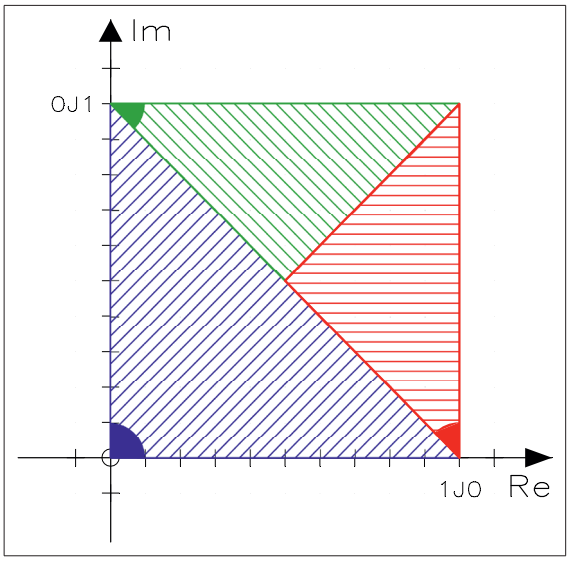


Figure 11: ⌊ - mapping

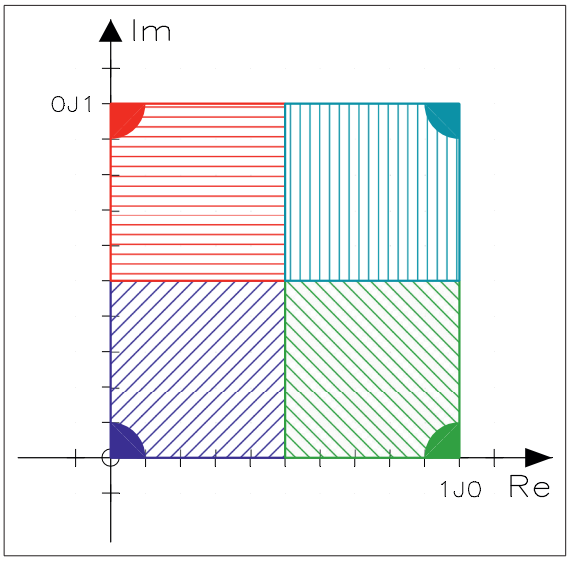


Figure 12: Mapping to nearest lattice point

Examples As a first example fig. 13 shows an addition of Gaussian fractions, showing that calculations with fractions finally yield the same result as traditional operations.

```
      ⍙init          ⍝ KhAlZi, -Eukl, -Quot
      (a b)←(1J5 4J¯7)(¯1J12 7J17)
      ⍙dar c←a ⍙a b
 ¯16J25/31J27
      (÷/c)≡(÷/a)+(÷/b)
1
```

Figure 13: Addition of fractions in $\mathbb{Q}[i]$

Fig. 14 contains methods to calculate the determinant of a matrix of Gaussian fractions:

(a) `Det ÷/¨mat` divides the fractions to rational machine numbers and calculates the determinant using the LU-decomposition.

(b) `÷/⍙ldet mat` calculates the determinant of fractions using Laplace method.

(c) `÷/⊃⍙m/1 1⍉2⊃1 ⍙lgaut mat` computes the determinant using the Gauss algorithm and multiplying all diagonal elements of the result.

(d) `÷/⊃det←¯1¯1↑2⊃0 0 1 ⍙lgaut mat` computes the determinant using the special Gauss algorithm explained in 2.3.

(e) `⍙dar det` displays the determinant as a fraction of Gaussian integers.

```
      Δinit          ⍝ KhAlZi, -Eukl, -Quot, -Matr, KhMatrix
      n←4⋄mat←(¯4J¯7+(⍳n))∘.,(2J1×⍳n)
      (1 1⍉mat)←(1 1⍉mat)+n↑,(⍳3)∘.×2J¯5×⍳3
      Δdar mat
  ¯1J¯12/4J¯4   ¯3J¯7/4J2     ¯3J¯7/6J3    ¯3J¯7/8J4
    ¯2J¯7/2J1  2J¯17/8J¯8     ¯2J¯7/6J3    ¯2J¯7/8J4
    ¯1J¯7/2J1   ¯1J¯7/4J2  5J¯22/12J¯12    ¯1J¯7/8J4
     0J¯7/2J1    0J¯7/4J2      0J¯7/6J3  4J¯17/12J¯6
      Det +/¨mat                        ⍝ real LU-decompos.
¯11.23053108J¯40.56266719
      +/Aldet mat                       ⍝ Laplace
¯11.23053108J¯40.56266719
      +/⊃Δm/1 1⍉2⊃1 Algaut mat          ⍝ Gauss
¯11.23053108J¯40.56266719
      +/⊃det←¯1 ¯1↑2⊃0 0 1 Algaut mat ⍝   = " =
¯11.23053108J¯40.56266719
      Δdar det
 4914541J¯4781776/78336J142848
```

Figure 14: Determinant of a matrix in $\mathbb{Q}[i]$

In fig. 15 a 4 by 4 matrix is inverted using the Gauss algorithm. In fig. 16 I try to do the same with a 5 by 5 matrix. This fails as shortening fails. This issue may be corrected a bit by improving addition and multiplication, but then we the same problem arises with larger matrices or numbers. To avoid this we have to use more storage for the numbers using an Accu-technic as proposed by Hahn/Mohr in their book [1] or by Roger Hui in his talk at Dyalog'20.

```
      Δdar inv←0 n↓2⊃2 Algaut mat,Δq n nρ(n+1)↑1
   41516J9548/236276J9001   ¯3344J¯1712/14743J41358   ¯2928J¯1584/14743J41358   ¯2376J¯2628/14743J41358
 ¯32128J21536/236276J9001   73832J10136/236276J9001  ¯25664J15328/236276J9001   ¯31188J6276/236276J9001
 ¯45360J20880/236276J9001  ¯40080J17040/236276J9001   100452J6996/236276J9001   ¯41040J1620/236276J9001
 ¯70056J¯3528/236276J9001  ¯60984J¯4872/236276J9001  ¯53928J¯5544/236276J9001  133656J30018/236276J9001
      ⎕pp←6⋄(+/¨mat)+.×+/¨inv
 1.00000E0                ¯5.55112E¯17J¯5.55112E¯17 8.32667E¯17              9.02056E¯17J¯2.77556E¯17
¯1.11022E¯16J1.11022E¯16  1.00000E0                 5.55112E¯17J¯4.16334E¯17 0.00000E0J 1.38778E¯17
 5.55112E¯17              ¯5.55112E¯17J 1.38778E¯17 1.00000E0                2.77556E¯17J 5.55112E¯17
 0.00000E0J1.11022E¯16    ¯5.55112E¯17J 4.16334E¯17 0.00000E0                1.00000E0
      Δdar mat Δa.Δm inv
  1  0  0  0
  0  1  0  0
  0  0  1  0
  0  0  0  1
```

Figure 15: A matrix and its inverse in $\mathbb{Q}[i]$

```
      n←5⋄mat←(¯4J¯7+(⍳n))∘.,(2J1×⍳n)
      (1 1⍉mat)←(1 1⍉mat)+n↑,(⍳3)∘.×2J¯5×⍳3
      Δdar mat
  ¯1J¯12/4J¯4   ¯3J¯7/4J2     ¯3J¯7/6J3    ¯3J¯7/8J4    ¯3J¯7/10J5
    ¯2J¯7/2J1  2J¯17/8J¯8     ¯2J¯7/6J3    ¯2J¯7/8J4    ¯2J¯7/10J5
    ¯1J¯7/2J1   ¯1J¯7/4J2  5J¯22/12J¯12    ¯1J¯7/8J4    ¯1J¯7/10J5
     0J¯7/2J1    0J¯7/4J2      0J¯7/6J3  4J¯17/12J¯6     0J¯7/10J5
     1J¯7/2J1    1J¯7/4J2      1J¯7/6J3     1J¯7/8J4  9J¯27/18J¯15

      Δdar inv←0 n↓2⊃2 Algaut mat,Δq n nρ(n+1)↑1
Kurzen: Cancellation fails
Δa[7]  r←Kurzen(RΔa/a RΔm¨⍋b),⊂(2⊃a)RΔm 2⊃b
            ^
Δs[6]  b[1]←-b[1]⋄r←a Δa b
              ^ ^
Δs[5]  →(∧/Δt=≡¨a b)/1+⎕LC⋄r←a Δs¨b⋄→0                ⍝ Subtraction in each cell
       ^                ^
Algaus[39]  mat[m;]←mat[m;]Δs(mat[m;j]Δd mat[j;j])∘.Δm mat[j;]
              ^        ^
Algaut[35]  →(0≠(1⊃r)←1⊃mat←(j,3↑ip)Algaus mat)/FX⋄mat←2⊃mat                    ⍝3.1
                         ^        ^
      Δdar inv←0 n↓2⊃2 Algaut mat,Δq n nρ(n+1)↑1
                ^ ^
      x←2*52 53 54⋄x+1-x ⋄(-x)+1+x
1 0 0
```

Figure 16: Cancellation fails: numbers get too big

3.3 Polynomials and their Quotients.

For a field K the polynomial ring $K[x]$ with $\varphi(p(x)) = \operatorname{grad} p$ is an Euclidean ring with $K[x]^\times = K$. Addition and multiplication of polynomials are standard algorithms. The Euclidean algorithm is polynomial division with remainder. The character or norm of a polynomial is its main coefficient. All functions are given in A.3.

Example 3.3 (Division of polynomials) After loading the work spaces and initializing the parameters we calculate, see fig. 17,

(a) $$\frac{x^2-2x}{2x-2} = \frac{x^2-2x}{1} \div \frac{2x-2}{1} = 0.5x - 0.5$$

(b) $$\frac{x^4-1}{x-1} = \frac{x^4-1}{1} \div \frac{x-1}{1} = x^3 + x^2 + x + 1$$

(c) $$\frac{x^5+x^4+x^3+x^2+x+1}{x+1} = x^4 + x^2 + 1$$

(d) $$\frac{x^2-2x+1}{x+1} \div 5x^2 + 2x - 2x - 1 = \frac{.2x^2 - .4x + .2}{x^3 + 1.4x^2 - .4}$$

This result shows the effect of the character: the main coefficient of the denominator is 1.

```
      ∆init   ⍝ KhAlPoly, -Eukl, -Quot
⍝     (x^2-2x+1)÷(2x-2):
      1 ∆dar (1 ¯2 1)1 ∆d (¯2 2)1
 -.5+.5x
⍝     (x^5-1)÷(x-1):
      ∆dar (¯1 0 0 0 1)1 ∆d (¯1 1)1
 1+x+x^2+x^3
⍝     (1+x+x^2+x^3+x^4+x^5)÷(1+x):
      ∆dar (∆q 1 1 1 1 1 1) ∆d (∆q 1 1)
 1+x^2+x^4
      1 ∆dar (1 ¯2 1)(1 1)∆d (¯2 2 5)1
 .2-.4x+.2x^2/-.4+.0x+1.4x^2+x^3
```

Figure 17: Computing with fractions of polynomials

Example 3.4 (Rational and complex polynomials) To calculate the characteristic polynomial of a matrix *mat* we use the workspace `KhAlPoly` and calculate the determinant of $mat - x \cdot \mathrm{Id}$. This is done by the function `Alchpo` in the ring of polynomials. It also can be calculated using the Gauss algorithm. This is all shown in Figure 18. Figure 19 shows an example with a complex matrix.

4 Concluding Remarks

I wonder whether it is good idea to increase the depth of an element of the quotient field by one. It might be better to increase it by two so one can distinguish between vectors or matrices of ring elements whose depth would be one above and fractions whose depth would be two above the depth of a scalar ring element.

```
      ∆init   ⍝ KhAlPoly, -Eukl, -Matr
      mat←4 4⍴1 ¯2 5 ¯5 ¯8 9 9 3 16,⍳11
      R∆dar Alchpo,¨mat      ⍝ determinant in ring
-4604+1699x-5x^2-19x^3+x^4
      ∆init   ⍝ KhAlPoly, -Eukl, -Quot, -Matr
      (1 1⍉mat)←(1 1⍉mat),¨¯1
      ∆dar ∆q ,¨mat          ⍝ determinant quotient field
 1-x   ¯2    5   ¯5
  ¯8  9-x    9    3
  16    1  2-x    3
   4    5    6  7-x
      ∆dar Aldet mat                ⍝ Laplace
 -4604+1699x-5x^2-19x^3+x^4
      ∆dar ∆m/1 1⍉2⊃0 Algaut mat    ⍝ Gauss
 -4604+1699x-5x^2-19x^3+x^4
      ∆dar 2⊃0 0 1 Algaut mat       ⍝ Gauss special
  1  2/-1+x           ¯5/-1+x                              5/-1+x
  0       1  49-9x/-7-10x+x^2                  -37-3x/-7-10x+x^2
  0       0                 1  -640+107x-3x^2/1071-76x-12x^2+x^3
  0       0                 0         -4604+1699x-5x^2-19x^3+x^4
      ∆dar ¯1 ¯1↑2⊃0 0 1 Algaut mat ⍝   = " =
 -4604+1699x-5x^2-19x^3+x^4
```

Figure 18: Computing with matrices of polynomials

```
      ∆init   ⍝ KhAlPoly, -Eukl, -Quot, -Matr
      ∆var←'z'
      mat←2 2⍴1 0J1 0J¯1
      (1 1⍉mat)←(1 1⍉mat),¨¯1
      ∆dar mat←∆q mat
  1-z  0J1
 0J¯1  1-z
      ∆dar Aldet mat                ⍝ Laplace
 -2z+z^2
      ∆dar ∆m/1 1⍉2⊃0 Algaut mat    ⍝ Gauss
 -2z+z^2
      ∆dar 2⊃0 0 1 Algaut mat       ⍝ special Gauss
  1  0J¯1/-1+z
  0    -2z+z^2
      ∆dar ¯1 ¯1↑2⊃0 0 1 Algaut mat ⍝   = " =
 -2z+z^2
```

Figure 19: Computing with complex polynomials

Appendix Implemented Functions

A.1 KhAlZ: The Operations on ℤ

```
r←a R∆a b
⍝ V1.1  04.01.1993  D.Kilsch    KhAlZ                    (04.01.1993)
⍝ Addition of integers
⍝ a,b,r  S,V    integers
⍝═══════════════════════════════════════════════════════════════════
r←a+b

r←a R∆m b
⍝ V1.1  04.01.1993  D.Kilsch    KhAlZ                    (04.01.1993)
⍝ Multiplication of integers
⍝═══════════════════════════════════════════════════════════════════
r←a×b
```

```
r←p RΔe q
⍝ V1.2  22.11.2020  D.Kilsch     KhAlZ                      (04.01.1993)
⍝ Euclidean algorithm for integers: a = r[1]×b + r[2]
⍝ p,q  S      integers
⍝ r    V      p=r[1]×q+r[2]
⍝═
⍝  Method: (2×|r)>|q: subtract q
⍝══════════════════════════════════════════════════════════════════════
r←((p-r)÷q),r←(-q×(2×|r)>|q)+r←q|p
```

```
r←RΔef z
⍝ V1.1  04.01.1993  D.Kilsch    KhAlZ                      (04.01.1993)
⍝ Euclidean value or function    Z: absolute value
⍝ a,b,r  S,V    integers
⍝═════════════════════════════════════════════════════════════════════
r←|z
```

```
r←RΔinit
⍝ V1.1  08.01.1993  D.Kilsch     KhAlZ                     (08.01.1993)
⍝ Initialisations for the ring of integers.
rΔt←0           ⍝ depth of a ring element
rΔnull←0        ⍝ zero in Z
rΔeins←1        ⍝ unity in Z
⍝════════════════════════════════════════════════════════════════════
```

```
r←RΔc a
⍝ V1.1  04.01.1993  D.Kilsch    KhAlZ                      (04.01.1993)
⍝ Character (Euclidean Norm) of an integer: sign
⍝ a    S,V    integer
⍝ r    S,V    +1, -1
⍝═════════════════════════════════════════════════════════════════════
r←×a
```

```
r←f RΔdar a
⍝ V1.1  04.01.1993  D.Kilsch     KhAlZ                    (04.01.1993)
⍝ Formatting integers
⍝ a    S,V    integers
⍝ f    TV     formatting parameter
⍝════════════════════════════════════════════════════════════════════
r←(f⍕a)~' '
```

A.2 `KhAlZi`: The Operations on $\mathbb{Z}[\mathrm{i}]$

Addition, multiplication and all initializations are identical to the operations on $\mathbb{Z}$. Hence the functions `RΔa`, `RΔm`, `RΔinit` and `RΔdar` may be taken from `KhAlZ`

```
r←a RΔe b
⍝ V1.1  21.11.2020  D.Kilsch     KhAlZi                       (21.11.2020)
⍝ Euclidean algorithm for Gaussian integers with nearest lattice point.
⍝ The middle points move to smaller real and/or imaginary values
⍝ a,b  S      Gaussian integers
⍝ r    V      Gaussian integers: a = r[1]×b + r[2]
⍝═
⍝  Method: r[1] is chosen to be the lattice point closest to a÷b.
⍝════════════════════════════════════════════════════════════════════════
r←(1⌷r)+0J1×2⌷r←⌈¯0.5+9 11○a÷b
r←r,a-b×r
→(∧/0=1|,9 11∘.○r)/0⋄'Runden: ',⍕r⋄r←1 0J1+.×⌊0.5+9 11∘.○r
```

```
r←RΔef z
⍝ V1.1  20.11.2020  D.Kilsch     KhAlZi                      (20.11.2020)
⍝ Euclidean value or function, Z[i]: square of absolute value
⍝ a,b  S,V    integers
⍝═══════════════════════════════════════════════════════════════════════
r←z×+z
```

```
r←RΔc a
⍝ V1.1  20.11.2023  D.Kilsch     KhAlZi                        (20.11.2020)
⍝ Character (Euclidean Norm) of a Gaussian integer: sign
⍝ a     S,V    integer
⍝ Method: Remainder of the argument wrt ○.5 (calculated by the residue)
⍝ and applying ¯12○ (multiply by 0J1 and take the exponential)
⍝═══════════════════════════════════════════════════════════════════════
→(0=r←a)/0
r←¯12○r-(○0.5)|r←12○r
```

A.3 KhAlPoly: The Operations on Polynomials over a Field

```
r←a RΔa b
⍝ V1.3  11.11.2020  D.Kilsch     KhAlPoly                      (04.01.1993)
⍝ Addition of polynomials, also with vectors
⍝ a,b,r   V    polynomials; a[1], b[1]: constant coefficient
⍝=
⍝  global variables
⍝ rΔt     depth of a ring element
⍝═══════════════════════════════════════════════════════════════════════
→(rΔt≥⌈/≡¨a b)/1+⎕LC⋄r←a RΔa¨b⋄→0
r←(⍴,a)⌈⍴,b⋄r←(1,1↓⍬∨\0≠⍬r)/r←r×(⎕CT×⌈/|∊a,b)<|r←(r↑a)+(r↑b)

r←a RΔm b;i
⍝ V1.1  04.10.1993  D.Kilsch     KhAlPoly                      (04.01.1993)
⍝ Multiplication of polynomials, or arrays of polynomials
⍝ a, b  V,M,AV  polynomials; a[1], b[1]: constant coefficient
⍝ r     V,M,AV  product
⍝=
⍝  global variables
⍝ rΔt     depth of a ring element
⍝═══════════════════════════════════════════════════════════════════════
→(rΔt≥⌈/≡¨a b)/1+⎕LC⋄r←a RΔm¨b⋄→0
→(~∨/∧/¨0=a b)/1+⎕LC⋄r←,0⋄→0
r←b+.×((⍴,b),r-1)⍴(r←⍴a,b)↑a


r←p RΔe q;gr;hk;rr
⍝  V1.2  28.10.1997  D.Kilsch     KhAlPoly                      (31.12.1992)
⍝ Euclidean algorithm for polynomials
⍝ p, q    V         polynomial
⍝ r       AV[2]     p÷q = r[1] R r[2]  or  p = q × r[1] + r[2]
⍝=
⍝  local variables
⍝ gr      V  degree of the polynomials
⍝ hk      S  quotient of main coefficient
⍝ rr      S  local: p-hk×q
⍝
⍝ called routines from the ring model:
⍝ RΔef Euclidean function
⍝ RΔc  norm
⍝═══════════════════════════════════════════════════════════════════════
r←0⍴0
DO:→(<⌊/gr←∊RΔef¨p q)/UNDO
rr←¯1↓p-(-1+↑gr)↑q×hk←÷/∊RΔc¨p q              ⍝ no main coefficient
((⎕CT×⌈/∊|p q)>|rr)/rr)←0                     ⍝ closed to zero
p←(1⌈1+RΔef rr)↑rr
r←((-gr[1]-(gr[2]-1)⌈RΔef p)↑hk),r⋄→DO
UNDO:r←((1⌈⍴r)↑r)p
```

```
r←RΔef a
⍝ V1.1  04.01.1993  D.Kilsch     KhAlPoly                    (04.01.1993)
⍝ Euclidean function for polynomials: degree of the polynomial
⍝ a       V    polynomial
⍝ r       S    degree of the polynomial, zero polynomial: -infinity
⍝═══════════════════════════════════════════════════════════════════════
⍎(¯1=r←¯1++/∨\⌽(a≠0))/'r←⌈/0⍴0'


r←RΔinit
⍝ V1.1  25.11.2020  D.Kilsch     KhAlPoly                 (08.01.1993)
⍝ Initialisation for the ring of polynomials.
⍝=
rΔt←1         ⍝ depth of a ring element
rΔnull←,0     ⍝ zeor of the ring
rΔeins←,1     ⍝ unity of the ring
Δvx←0 1       ⍝ variable
Δvar←'x'      ⍝ name of variable
⍝═══════════════════════════════════════════════════════════════════


r←RΔc a
⍝ V1.2  28.10.1997  D.Kilsch     KhAlPoly                    (04.01.1993)
⍝ Character -- Euclidean norm of a polynomial: main coefficient
⍝ a       V    polynomial
⍝ r       S    main coefficient; 0 for zero polynomial
⍝═══════════════════════════════════════════════════════════════════════
r←↑¯1↑(0≠a)/a


r←f RΔdar a;lind;aa;var
⍝ V1.5  25.11.2010  D.Kilsch     KhAlPoly                    (20.12.1992)
⍝ Textual representation of a polynomial
⍝ f    S,V[2] format for outpout of coefficients
⍝      AV[1]  S,V[2] see above                            [DEF.: (,0)(1)]
⍝        [2]  1: representation starts with constant term
⍝             2: representation starts with main term
⍝ a    V      coefficients of the polynomial, upwards
⍝ r    TV     textual representation of the polynomial
⍝=
⍝  Global variables
⍝ Δvar  TS,TV name of variable
⍝
⍝  Variables
⍝ aa    V     real / complex inpit
⍝ lind  LS    logical index
⍝ var   TV    name of variable
⍝═══════════════════════════════════════════════════════════════════════
⍎(2≠⎕NC 'f')/'f←,0'
→(2≤≡f)/1+⎕LC⋄f←(⊂f)(1)
→(1<⍴,a)/1+⎕LC⋄r←⍕a⋄r←⍎∊(1 0=0=11⎕a)/'(∊↑f)⍕a' '⍕a'⋄→0⍝ zero polyn.
⍝
→(~∨/lind←0≠11⎕a)/ENDIFC
(lind/a)←'+',¨⍕¨lind/a
ENDIFC:→(~∨/lind←~lind)/ENDIFR
r←(r\(r←1,1↓1≠|aa)/(↑f)⍕¨|aa←lind/a)~¨' '    ⍝ const. coeff. always
(lind/a)←((⊂1+aa≥0)⌷'-+'),¨r
ENDIFR:
r←(~a≡¨⊂'+0')/a,¨(⍴a)↑('' Δvar),(⊂Δvar,'^'),¨⍕¨1↓¯1↓⍳⍴a
→(2≠2⊃f)/1+⎕LC⋄r←⌽r                      ⍝ downwards
r←('+'=↑r)↓r←∊r                          ⍝ no starting '+'
```

A.4 KhAlEukl: The Operations on an Euclidean ring

```
r←a Δa b
⍝ V1.1  08.01.1993  D.Kilsch     KhAlEukl                (1.1 08.01.1993)
⍝ Addition in an Euclidean ring is ring addition
⍝═══════════════════════════════════════════════════════════════════════
→(∧/rΔt=≡¨a b)/1+⎕LC⋄r←a Δa¨b⋄→0
r←a RΔa b

r←a Δm b
⍝ V1.1  08.01.1993  D.Kilsch     KhAlEukl                 (1.1 08.01.1993)
⍝ Multiplication in an Euclidean ring is ring multiplication
⍝═══════════════════════════════════════════════════════════════════════
→(∧/rΔt=≡¨a b)/1+⎕LC⋄r←a Δm¨b⋄→0
r←a RΔm b

r←a Δs b
⍝ V1.1  08.01.1993  D.Kilsch     KhAlEukl                 (1.1 08.01.1993)
⍝ Subtraction in an Euclidean ring is adding of the negative
⍝═══════════════════════════════════════════════════════════════════════
→(∧/rΔt=≡¨a b)/1+⎕LC⋄r←a Δs¨b⋄→0
r←a RΔa-b


r←a ggT b;rr
⍝ V1.3  20.11.2020  D.Kilsch  KhAlEukl:                       (20.12.1992)
⍝ ggT calculates the greatest common devisor.
⍝ a      element of Euklidian ring
⍝ b      element of Euklidian ring
⍝ r      gcd(a,b): depth and structure according to input
⍝
⍝  global variable:
⍝ rΔt    S E  depth of a ring elemnt
⍝ rΔnull   E  zero of the ring
⍝=
⍝  called routines from the ring model:
⍝ RΔe  Euclidean algorithm
⍝ RΔef Euclidean function
⍝ RΔc  Character -- Euclidean Norm
⍝
⍝  method: |a|: grad a: euklidsche Funktion
⍝          ggT(a,b) = ggT(min(|a|,|b|),remainder max(|a|,|b|)/min(|a|,|b|)
⍝          ggT(a,0) = a;  ggT(0,b) = b
⍝═══════════════════════════════════════════════════════════════════════
→(∧/rΔt=≡¨a b)/1+⎕LC⋄r←a ggT¨b⋄→0
→(~∨/r←(⊂rΔnull)≡¨a b)/1+⎕LC⋄r←r÷RΔc r←1⊃((~r),1)/a b rΔnull⋄→0 ⍝ fou 1.3
⍝⍎(∨/∊0.1>|(⊂rΔnull)-a b)/'xxx'
r←b ggT 2⊃⊃RΔe/(a b)←(</RΔef¨a b)⌽a b⋄→0                          ⍝1.3


r←a kgV b
⍝ V1.1  20.12.1992  D.Kilsch  KhAlEukl:                       (20.12.1992)
⍝ ggT calculates the lowest common multiple
⍝ a      element of Euclidean ring
⍝ b      element of Euclidean ring
⍝ r      lcm(a,b): depth and structure according to input
⍝
⍝ global variable:
⍝ rΔt    S E  depth of a ring element
⍝=
⍝ called routines
⍝ EUKLRING: ggT
⍝
⍝ called routines from the ring model:
⍝ RΔe  Euclidean algorithm
⍝ RΔm  ring multiplication
```

```
⍝
⍝ method: 1st component of Euklidian algorithm applied to a×b and gcd(a,b)
⍝═══════════════════════════════════════════════════════════════════════════
→(∧/rΔt=≡¨a b)/1+⎕LC⋄r←a kgV¨b⋄→0
r←1⌷(a RΔm b)RΔe a ggT b
```

A.5 KhAlQuot: The Operations on a Quotient Field

```
r←Δinit
⍝ V1.1  08.01.1993  D.Kilsch     KhAlQuot                    (08.01.1993)
⍝ Initialisation of a quotient field
⍝=
⍝  called routins:
RΔinit
⍝=
⍝  global variables
Δt←rΔt+1                ⍝ depth of a field element
Δnull←rΔnull rΔeins     ⍝ zero of the field
Δeins←rΔeins rΔeins     ⍝ unity of the field
⍝═══════════════════════════════════════════════════════════════════════════


r←Δq a
⍝ V1.1  08.01.1993  D.Kilsch     KhAlQuot                      (08.01.1993)
⍝ Transforming ring elements to field elements with denominator rΔeins.
⍝ a    S,V  integral element
⍝═══════════════════════════════════════════════════════════════════════════
⍎((rΔt=1+≡a)∧0=ρρa)/'a←,a'   ⍝ ring element as vector if depth is too low
→(rΔt=≡a)/1+⎕LC⋄r←Δq¨a⋄→0
r←a rΔeins


r←Kurzen rel
⍝ V1.3  27.10.1997  D.Kilsch  KHALGEBRA: EUKLRING              (20.12.1992)
⍝ Cancellation of a fraction, denominator with positive norm
⍝ rel  AV[2]  nominator and denominator
⍝ r    AV[2]  shortened fraction, denominator with posiitv norm
⍝=
⍝  called routines
⍝ Euklring: ggT (gcd)
⍝
⍝  called routines from underlying ring model:
⍝ RΔe  Euclidean algorithm
⍝ RΔc  Norm (of denominator)
⍝═══════════════════════════════════════════════════════════════════════════
r←⊃[2]rel RΔe¨ggT/rel⋄→(∨/∊⎕CT<|2 ¯1↑r)/F1
r←(÷RΔc 2⊃r)×r←1⌷[2]r⋄→0              ⍝ denominator with positive norm
⍝═══════════  Fehlerbehandlung
F1:'Error on input'⋄rel⋄'Kurzen: Shortening fails' ⎕ES 11 21


r←f Δdar a
⍝ V1.2  20.11.2020  D.Kilsch     KhAlQuot                    (08.01.1993)
⍝ Formatting of fractions
⍝ a    S,V  fractions
⍝ f    TV   format
⍝═══════════════════════════════════════════════════════════════════════════
→(2=⎕NC 'f')/1+⎕LC⋄f←0
→(Δt=≡a)/ELSE
r←(-∊⌈/,ρ¨r~¨' ')↑¨r←f Δdar¨a⋄→(2≠ρρr)/0
r←(~(2⌷ρr)↑(∧≠0 1 2∘.⌽3∧/∧≠r=' '))/r←⍉r,¨' '
→0
ELSE:
r←' ',1↓∊'/',¨f RΔdar¨(-rΔeins≡2⊃a)↓a
```

```
r←a Δa b
⍝ V1.1  08.01.1993  D.Kilsch     KhAlQuot                    (08.01.1993)
⍝ Addition of fractions
⍝ a,b  S,V  fractions
⍝═══════════════════════════════════════════════════════════════════════
→(∧/Δt=≡¨a b)/1+⎕LC⋄r←a Δa¨b⋄→0
→((1≠(⍴⍴a),⍴⍴b)∨2≠(⍴a),⍴b)/F1
r←Kurzen(RΔa/a RΔm¨⌽b),⊂(2⊃a)RΔm 2⊃b
→0
⍝═══════════════════════════════════════                error handling
F1:('Δa: Structure of left or right argument wrong:: ',⍕a,b)⎕ES 4 1

r←a Δs b
⍝ V1.1  08.01.1993  D.Kilsch     KhAlQuot                    (08.01.1993)
⍝ Subtraction of fractions
⍝ a,b  S,V  fractions
⍝═══════════════════════════════════════════════════════════════════════
→(∧/Δt=≡¨a b)/1+⎕LC⋄r←a Δs¨b⋄→0              ⍝ Subtraction in each cell
b[1]←-b[1]⋄r←a Δa b

r←a Δm b
⍝ V1.1  08.01.1993  D.Kilsch     KhAlQuot                    (08.01.1993)
⍝ Multiplication of fractions
⍝ a,b  S,V  fractions
⍝═══════════════════════════════════════════════════════════════════════
→(∧/Δt=≡¨a b)/1+⎕LC⋄r←a Δm¨b⋄→0
→((1≠(⍴⍴a),⍴⍴b),2≠(⍴,a),⍴,b)/F1⋄r←Kurzen a RΔm b⋄→0
⍝═══════════════════════════════════                error handling
F1:('Structure of left or right argument wrong:: ',⍕a,b)⎕ES 4 13

r←a Δd b
⍝ V1.1  08.01.1993  D.Kilsch     KhAlQuot                    (08.01.1993)
⍝ Division of fractions
⍝ a,b  S,V  fractions
⍝═══════════════════════════════════════════════════════════════════════
→(∧/Δt=≡¨a b)/1+⎕LC⋄r←a Δd¨b⋄→0
r←a Δm⌽b                          ⍝ Interchange nominator and denominator
```

A.6 KhAlMatr: Functions and Operations on Matrices

```
r←Alchpo m
⍝  V1.1  01.08.1992  D.Kilsch     KhAlMatr                    (01.08.1992)
⍝ Calculating the characteristic polynomial of a square matrix
⍝ m      AM   square matrix
⍝=
⍝  Global variables
⍝ Δvx         Variable x
⍝=
⍝  Called functions:  Aldet1
⍝ KhAlQuot:           Δinit   Δs
⍝=========================================================================
(1 1⍉m)←(1 1⍉m)Δs⊂Δvx
r←Aldet1 m


r←Aldet mat
⍝  V1.2  30.11.1997  D.Kilsch     KhAlMatr                    (01.08.1992)
⍝ Calculating the determinant of a square matrix.
⍝ mat    AM   square matrix
⍝ r      AS   determinant
⍝
⍝ Method: Laplace method with respect to first row, taking zeros in this
⍝          row into account
⍝=
⍝  Global variables
⍝ Δnull ??  Nnull
```

```
⍝=
⍝  Called functions:  Aldet1
⍝ KhAlQuot:           ∆init   ∆q
⍝=========================================================================
→(2=⎕NC '∆t')/1+⎕LC⋄∆init
→((≡mat)=1+∆t)/1+⎕LC⋄mat←∆q mat
→((2≠⍴⍴mat)∨≠/⍴mat)/FE1                        ⍝ Matrix not square
r←Aldet1 mat
→0
⍝======     error handling
FE1:'Aldet: Matrix not square: ',⍕⍴mat


r←Aldet1 mat;i;n;rr
⍝  V1.1  01.08.1992  D.Kilsch     KhAlMatr                    (01.08.1992)
⍝ Calculating the determinant of a square matrix.
⍝ mat    AM   square matrix
⍝ r      AS   determinant
⍝
⍝ Method: Laplace method with respect to first row, taking zeros in this
⍝          row into account
⍝=
⍝  Global variables
⍝ ∆null ??  Nnull
⍝=
⍝  Local variables
⍝ n      S   degree of matrix
⍝ i      S   loop variable
⍝ rr     AS  subdeterminant
⍝
⍝  Called functions: Aldet1
⍝ KhAlQuot:          ∆m      ∆a      ∆s
⍝=========================================================================
→(1≠↑⍴mat)/ELSE
r←,↑mat⋄→ENDIF
ELSE:r←∆null⋄i←0⋄n←↑⍴mat
DO:→(n<i←i+1)/UNDO⋄→(∧/0=∊mat[1;i])/DO
rr←(⊃mat[1;i])∆m Aldet1(i≠⍳n)/1 0↓mat
⍎'r←r ',(∊(0 1=2|1+i)/'∆a' '∆s'),' rr'⋄→DO
UNDO:
ENDIF:→0


r←ip Algaus mat;j;m
⍝  V3.1 15.11.2020  D.Kilsch     KhAlMatr                    (11.11.1995)
⍝ One step of Gauss' algorithm with the operations ∆a, ∆s, ∆m, ∆s
⍝ ip  V [1] number of pivot column / row
⍝       [2] =0: Gauss elimination below diagonal only            [DEF.: 0]
⍝           =1: Gauss elimination below and above diagonal
⍝           =2: in addition to 1: divide pivot row by pivot element
⍝       [3] ≠0: pivot row: choose row with max. absolut element  [DEF.: 0]
⍝       [4] ≠0: in addition to 0: (determinat does not change)   [DEF.: 0]
⍝               - divide pivot row by pivot element
⍝               - multiply next row by pivot element
⍝ mat M       input matrix
⍝ r   AV [1]  S error, 0: no error
⍝     AV [2]  M resulting matrix
⍝=
⍝  Locale variables:
⍝ j   S     ←ip[1]
⍝ m   V     1. ⍳1↑⍴mat ab j     {j, j+1, ..., 1↑⍴mat}
⍝     S     2. found pivot row
⍝     V     3. ⍳1↑⍴mat \ {j} =  {1, 2, ..., j-1, j+1, ..., 1↑⍴mat}
⍝=========================================================================
j←↑ip←4↑ip                          ⍝ ip[2 3 4] not inserted: default
→((j≤0)∨j>⌊/⍴mat)/F1
```

```
→(0=ip[3])/ENDIF                    ⍝ nor search for pivvot element
⍝═══════════  Pivotwahl in [j,1↑⍴mat]  ═══════════════════════════════
m←(j≤⍳1↑⍴mat)/⍳1↑⍴mat               ⍝ indexes  j, j+1, ..., 1↑⍴mat
m←j+¯1+(,Δb mat[m;j])⍳⌈/Δb mat[m;j]    ⍝ row index with max. absolute el
→(m=j)/ENDIF
mat[(m,j);]←mat[(j,m);]             ⍝ interchange rows
ENDIF:→((Δb⊃mat[j;j])≤1E¯7)/F2      ⍝ no pivot element found
⍝═══════════  Gauß - Elimination  ═══════════════════════════════════
⍎(0=2⌷ip)/'m←(j<⍳1↑⍴mat)/⍳1↑⍴mat' ⍝ indexes  j+1, ..., 1↑⍴mat
⍎(0≠2⌷ip)/'m←(j≠⍳1↑⍴mat)/⍳1↑⍴mat' ⍝ Indexes  1, 2,..,j-1, j+1,..,1↑⍴mat
mat[m;]←mat[m;]Δs(mat[m;j]Δd mat[j;j])∘.Δm mat[j;]
⍎(ip[2]=2)/'mat[j;]←mat[j;] Δd mat[j;j]'⍝ diagonal element 1
→(~(ip[1]<↑⍴mat)∧ip[4]>0)/END
mat[j+1;]←mat[j+1;]Δm mat[j;j]      ⍝ next to pivot row: mult. with p. e
mat[j;]←mat[j;]Δd mat[j;j]          ⍝ diagonal element: 1
END:r←0⋄→ENDE
⍝═══════════  Error handling  ═══════════════════════════════════════
F1:j←'wrong column or row index: ',⍕j⋄r←¯101⋄→FE
F2:j←(⍕j),'. pivot element closed to 0: ',⍕mat[j;j]⋄r←¯102⋄→FE
FE:'Algaus: ',j
ENDE:r←r mat
```

```
r←ip Algaut mat;j
⍝  V2.6  11.06.2007  D.Kilsch     KhAlMatr                    (__.01.1988)
⍝ Solving a linear system of ewqautions using Gauss' algorithm
⍝ ip  V [1] =0: Gauss elimination below diagonal only            [DEF.: 0]
⍝           =1: Gauss elimination below and above diagonal
⍝           =2: in addition to 1: divide pivot row by pivot element
⍝       [2] ≠0: pivot row: choose row with max. absolut element  [DEF.: 0]
⍝       [3] ≠0: in addition to 0: (determinant does not change)  [DEF.: 0]
⍝               - divide pivot row by pivot element
⍝               - multiply next row by pivot element
⍝       [4] = 1: output intermediate results                     [DEF.: 0]
⍝ r  AV [1]  S error, 0: no error
⍝    AV [2]  M resulting matrix
⍝       [2..]M ip[3]=1: intermediate results, las comp.: final result
⍝=
⍝  Local variables:
⍝ j    S     loop variable
⍝
⍝  Called routines:  Algaus
⍝ KhAlQuot:            Δq      Δinit
⍝═══════════════════════════════════════════════════════════════════════
→(2=⎕NC 'Δt')/1+⎕LC⋄Δinit
→((≡mat)=1+Δt)/1+⎕LC⋄mat←Δq mat
⍎(2≠⎕NC 'ip')/'ip←0'⋄ip←4↑ip
⍝
j←0⋄r←0 mat
DO:→((⌊0 ¯1+⍴mat)<j←j+1)/UNDO
→(0≠(1⊃r)←1⊃mat←(j,3↑ip)Algaus mat)/FX⋄mat←2⊃mat
→(4⌷ip)/ELSE2⋄(2⊃r)←mat⋄→DO
ELSE2:r←r,⊂mat⋄→DO
⍝
UNDO:→0
⍝═══════════════════════════════════════════════════════════════════════
FX:'Algaut, ',(⍕↑1↓⎕LC),' '
```

```
r←f Alzeim mat;kv;rr
⍝  V2.2  23.11.2020  D.Kilsch     KhAlMatr                    (13.11.1995)
⍝ Formatting the Gauß-system with n[1]vectors on the right hand side.
⍝ f    V   [1]: number of vectors on the right hand side    [DEF.: ⌊/⍴mat]
⍝          [2]: number of decimals to be shown      [DEF.: 0/2: Q / not Q]
⍝ mat  AM  to be solved
⍝      AV  [1]: matrix od coefficients
```

```
⍝          [2]: right hand side vector or matrix
⍝ r     TM  scheme includsing brackets
⍝═
⍝ Local variables:
⍝ kv    S   right hand side
⍝ rr    LV  locally used
⍝═══════════════════════════════════════════════════════════════════════════
⍎(2≠⎕NC 'f')/'f←⌊/⍴mat'⋄f←f,(⍴,f)↓0,(1 0=∧/0=1|∊mat)/0,2
→((1 2=⍴⍴mat),1)/C1,C2,FE0
C1:(mat kv)←mat⋄⍎∊(0 1=⍴⍴kv)/'→FE1' 'kv←,[1.1]kv'⋄→EC
C2:kv←(-1⌷f)↑[2]mat⋄mat←(-1⌷f)↓[2]mat⋄→EC
EC:f←2⌷f⋄→((↑⍴mat)≠↑⍴kv)/FE2
r←((⌽,[2 3]⊃' ',¨⌽¨f ∆dar¨⌽kv),' '),'│'
r←'│',(⌽,[2 3]⊃' ',¨⌽¨f ∆dar¨⌽mat),((0<¯1↑⍴kv)/((↑⍴mat),2)⍴' ','│'),r
r←((1⌽rr)∨rr←∨⌿r≠' ')/r                  ⍝ delete repeating empty columns
r[1,↑⍴r;1,¯1↑⍴r]←2 2⍴'┌└'⋄→0
⍝═══════  error handling  ═══════════════════════════════════════════════
FE0:'Alzeim: input matrix has wrong shape: ',⍕⍴⍴mat⋄r←¯600⋄→0
FE1:'Alzeim: right jhand side is scalar'⋄r←¯601⋄→0
FE2:'Alzeim: Matrix and right hand side don''t fit: ',⍕(⍴mat),⍴kv⋄r←¯602⋄→0
```

References

[1] Hahn, W. and K. Mohr: *APL/PCXA*. Hanser, München, 1988.

[2] Kaplansky, E.: *Fields and Rings*. Chicaga Lecture Notes in Mathematics Series, 2. ed., 1972.

[3] Lang, S.: *Algebra*. Graduate Texts in Mathematikmatics. Springer, New York, 3. ed., 1971.

[4] Robinson, D. J.: *Abstract Algebra: An Introduction with Applications*. De Gruyter Textbook. Walter de Gruyter, Berlin, 2nd ed., 2015.

[5] Samuel, P.: *On euclidean rings*. Journal Algebra, 19:282–301, 1971. https://doi.org/10.1016/0021-8693(71)90110-4.

[6] Sims, C. C.: *Abstract Algebra – A computational Approach*. Wiley, New York, 1984.

Fenster ribbeln

Michael Baas

Worum geht's?

Zunächst möchte ich ganz klar bekennen, dass die Buchstabenkombination im Titel dieses Aufsatzes recht willkürlich und sinnfrei zwei herausragende APL-Eindeutschungen wiedergibt, mit denen ich meine Schwierigkeiten hatte/habe. Ich selbst spreche eigentlich meistens „englisches APL", nutze also englische Begriffe für die APL-Symbole und Funktionen. Als ich mich aber aufmachte, Videos von engl. Videos von Morten und Adám mit deutschen Untertiteln zu versehen, hatte ich Probleme:

1. In den Untertiteln kann kein APL-Zeichensatz verwendet werden, somit musste ich die Symbole benennen.
2. Im normalen Sprachgebrauch störten mich oft die unnötigen Anglizismen – und nun gefielen mir meine eigenen Texte nicht mehr! Ich suchte also nach deutschen Begriffen für die englischen Worte.

Das klingt erstmal mehr nach einem persönlichen Problem, denn schließlich wurde in Deutschland schon APL programmiert (und gesprochen), bevor an Videos überhaupt zu denken war. Ich selbst habe autodidaktisch durch Lektüre von Gilman/Rose APL gelernt und bin nach ca. 3 Jahren mit APL*PLUS auf Dyalog APL umgestiegen – bin also hauptsächlich an die englische Terminologie gewöhnt. Die Bezeichnung „Fenster" ist mir aus frühen Meetings von APL Germany e.V. in Erinnerung geblieben für das ⎕-Symbol[1].

Ich habe dann einmal begonnen, für meinen eigenen Gebrauch eine Übersetzungstabelle anzulegen – um konsistent zuarbeiten. Schließlich erinnerte ich mich an Hans Lochners „APL Handbuch", in dem ja viele Dinge auch schon vorkamen. Um nun nicht eine „Baas-Sprache" zu schaffen, wäre es sicherlich sinnvoll, auf etablierte Begriffe zurückzugreifen.

Andererseits bin ich bei Dyalog mit einer lebendigen Sprache konfrontiert – mit (fast) jeder neuen Version kommen neue Symbole oder neue Funktionen hinzu, die zunächst auf Englisch benannt sind. Beispielsweise sind mir keine deutschen Begriffe für fork, top, beside, over, trains (im APL-Zusammenhang) bekannt.

Als die Liste dann also wuchs und klar wurde, dass es nicht nur um das Sammeln vorhandener Begriffe gehen würde, sondern auch um Neuschöpfungen, bekam ich Skrupel! Was nützt denn all der Spaß an Sprache und Kreativität, wenn niemand diese Sprache nutzt – helfen diese Eindeutschungen denn überhaupt, oder

bin ich am Ende ein Sonderling, der sich auf gar erquickliche Weise auszudrücken vermag – am Ende aber vielleicht nicht verstanden wird?! Das sei ferne!

So reifte eben langsam aus ganz praktischen Erwägungen heraus die Einsicht, dass Sprache in einer Gruppe Gleichgesinnter entsteht und wächst. Es erscheint daher ganz natürlich, für meine Suche nach deutschem APL in einer „Gruppe Gleichgesinnter“ um Hilfe zu bitten. APL Germany e.V. scheint mir hierfür von allen denkbaren Gruppen am passendsten zu sein! ;)

Doch in einem ersten Schritt möchte ich gerne herausfinden, ob es hierfür Bedarf gibt oder ob die Mehrheit mit englischem Sprachgebrauch zufrieden ist.

Um die Stimmung einzufangen, habe ich eine unter folgender Internet-Adresse eine Umfrage eingerichtet:
https://mbaas.survey.fm/deutsches-apl
Auf dem Treffen von APL Germany am 24.11.2020 rief ich zur Teilnahme auf – und ich wiederhole diesen Aufruf nun gerne nochmals im APL Journal. Voraussichtlich im Herbst 2021 werde ich die Ergebnisse der Umfrage und ggf. nächste Schritte vorstellen. (Bislang ist die Teilnahme eher verhalten – vielleicht ist es doch ein exotisches Unterfangen?)

Ich freue mich über jegliche Kommentare und Hinweise hierzu, gerne auch per Mail oder telefonisch.

Anmerkungen

Diesen Artikel schreibe ich primär als Privatmann Michael Baas, nicht als Mitarbeiter von Dyalog.

1 Zumindest in der Fußnote möchte ich noch den Begriff „Ribbeln“ klären: Als Gitte Christensen vor Jahren mit der Übersetzung englischer Texte von Kenneth Iverson beschäftigt war, fragte sie ihn, wie „ravel“ zu verstehen sei. Er erklärte, sie solle sich vorstellen, einen Wollpullover zu stricken – dann die Stricknadeln herauszuziehen bzw. die Wolle abzuziehen. Bei der Suche nach einem deutschen Begriff hierfür stieß ich auf „ribbeln“. Lochner verwendet hierfür „aufreihen“, was ich wesentlich angenehmer finde.

Kontakt:

Michael Baas, Kinzigstraße 10a, 63584 Gründau
E-Mail: mb@mbaas.de, Tel.: 0160-5887507

Array Programming Using ELI

Hanfeng Chen and Wai-Mee Ching

Abstract ELI is an interactive array-oriented programming language system based on APL. It extends flat array operations of ISO APL with lists for non-homogeneous data, temporal data, symbol type, and control structures. By replacing each APL character with one or two ASCII characters, ELI retains APL's succinct and expressive way of doing array programming in a dataflow style. A scripting file facility helps easily organize programs in a fashion akin to C programming with convenient input/output. With cross-platform support, ELI is freely available on Windows, macOS and Linux. It also comes with a compiler for flat array programs.

1. Introduction

ELI [1] has most of the functionality of the ISO APL standard, but it also has additional facilities such as lists for non-homogeneous data, complex numbers, symbols, temporal data, control structures, scripting files, dictionaries, tables and SQL-like statements. It also has new primitive functions and the ***each*** operator. It comes with a compiler for flat array programs. ELI is succinct, easy to learn, and versatile. Compared with MATLAB and Python, ELI encourages a dataflow style of programming where the output of one operation feeds the input of another, resulting in greater productivity and clarity of code. We explained the history of APL and our rationale for developing ELI in our earlier paper [2] in Vector.

We present examples below to illustrate the flavour of the language. The "//" comments out the string right after it.

```
    1 2 3         // an integer vector having three elements: 1,2,3
1 2 3
    %1 2 3        // 1 divide by 1, 2, and 3
1 0.5 0.3333333333
    1024*.%1 2 3  // 1024 takes 1 root, square root, and cube root
1024 32 10.0793684
    1 - 2         // 1 minus 2
_1
    _1*.0.5       // square root of minus 1
0j1
    @1            // pi
3.141592654
    *.0j1*@1      // e^(iπ) = -1
_1
    2012.12.25+!7 // 7 days following Christmas of 2012
2012.12.26 2012.12.27 2012.12.28 2012.12.29 2012.12.30 2012.12.31 2013.01.01
```

ELI language, following the ***minimality*** principle of classical APL, has simple but consistent rules. By adding ***lists*** on top homogeneous arrays as the only other data organizing structure and the addition of symbols and date-time on top of numbers and characters as atomic data types, it

comes with a database management subsystem. ELI is suitable for writing complex data analytic programs quickly, and it is ideal for teaching youngsters to learn applied mathematics and useful coding easily.

To overcome the efficiency problem of an interpretive language, ELI provides a compiler for those ELI programs which only use flat arrays. In addition, we provide a translator from flat array APL programs to ELI.

We would like ELI to be more widely used and the ELI compiler to compile more than flat arrays. On one hand, we intend to search application areas where ELI's productivity can shine and on the other hand, we plan to provide more educational materials in order to introduce ELI to young students.

2. Array Language ELI

ELI is based on APL1 as described in the ISO APL standard [3], and it replaces each APL character symbol with one or two ASCII characters.

symbol name (function)	APL	ELI	*symbol name (function)*	APL	ELI
jot	∘	.			
dot	⍤	:			
each	¨	"			
unique/equal	=	=			
minus(negate/subtract)	−	−			
at(pi/circle)	○	@	del	∇	@.
epsilon(member)	∊	?	random	?	?.
star(signum/multiply)	×	*	power	*	*.
percent(reciprocal/div)	÷	%	log	⍟	%.
bar(absolute/residue)	|	|	gamma(factorial/binom)	!	|.
rho(shape/reshape)	⍴	#	domino	⌹	#.
turn(reverse/rotate)	⌽	$	reverse/rotate on 1st axis	⊖	$.
left(grade_up/less)	⍋<	<	pack(enclose/encode)	⊂⊤	<.
right(grade_down/greater)	⍒>	>	unpack(grouping/decode)	⊃⊥	>.
iota(interval/index_of)	⍳	!	execute/drop	⍎↓	!.
count/and	∧	^	first/take	↑	^.
or	∨	&	flip(transpose)	⍉	&.
plus(conjugate/add)	+	+	format	⍕	+.
high_minus	¯	_	lower(floor/minimum)	⌊	_.
not	~	~	upper(ceiling/maximum)	⌈	~.
slash(reduce/compress)	/	/	slashdot(reduct1/compre1)	⌿	/.
back slash(scan/expand)	\	\	bkslashdot(scan1/expand1)	⍀	\.

catenate	,	,	raze/catenate along 1st axis		,.
			assign	←	<–
			branch	→	–>
			nand	⍱	~^
			nor	⍲	~&
			not equal	≠	~=
			less or equal	≤	<=
			greater or equal	≥	>=
			comment	⍝	//
			quad	⎕	[]
			bare_quad	⍞	[)

As we can see the table has a left part and a right part each with 3 columns: the first column is the symbol name with possibly the names of the monadic or dyadic function it represents (a monadic function has only one argument on the right-hand side while a dyadic function has two arguments on both left-hand and right-hand sides); the second column contains APL symbols; and the third column has corresponding ELI symbols. We note that the **or** function represented in ELI by "&" only has a dyadic form in APL and ELI, but the ***and*** (^) function has a monadic incarnation (the ***count*** function) in ELI. In fact, this also applies to the symbol "?" which is a monadic function in ELI meaning ***where*** that gives out the indexing positions of 1s in its input Boolean vector.

On the right side of the table, there are two-character symbols in ELI. In case a symbol has a left side counterpart in our table, the second character is an extra dot (`.`). The way these double-char symbols are chosen is to make it easy to remember the meaning of the primitives involved. As a symbol, of course, there should be no blanks in between. In APL, there is no need to put a space between two symbols. In ELI this is generally still true except for the symbols in the table with an empty left side. For example, in an expression with less than applied to negation, one must write "< –" (with a space in the middle) for "<–" will be interpreted as an assignment. There is also a double bar symbol (`||`) [4] which represents a primitive function in ELI dealing with lists (hence not in APL).

We give examples of ***lists*** as follows: (Note that, unlike arrays which start with just data items separated by a blank, a list starts with a left "`{`" and ends with a closing "`}`" with elements separated by a "`;`")

```
    (2 3#!6;`ny`ma`md;'ABCDE')   // a list of numbers, symbols, and chars
<1 2 3
 4 5 6
<`ny `ma `md
<ABCDE
    #"(2 3#!6;`ny`ma`md;'ABCDE') // shape of each element in the list
<2 3
<3
<5
```

When displaying the value of a list, each item is headed by a "<". As can be seen, a list contains multiple list cells and each cell can be integers, symbols, characters, and any other types. That means a list can have cells of which may be another list.

3. Database subsystem and SQL-like statements in ELI

Through dictionaries with the domain being symbols and target being arrays of equal length, we get tables. Given a dictionary as follows, we can see three symbols as keys (starting with a tick `) and their corresponding vectors as values. It is unnecessary to require all vectors to have the same type because they often have items with different types, such as numbers and names.

```
    D<-(`sym `price `hq:(`appl `ibm `hp `goog;449.1 108.2 24.5 890.3;4
2#'CANYCACA'))
    D
sym  | appl ibm hp goog
price| 449.1 108.2 24.5 890.3
hq   | 'CANYCACA'
```

The structure of a dictionary is similar to a *hash* table following the design of key-value mapping. However, it is a general design allowing keys and values may be any type that is different from a table of which a table column consists of a symbol as the key and a vector as the value. Therefore, we provide a new type called ***table*** that has strict rules and it can be generated from a dictionary using the primitive function transpose (&.).

```
    &.D
sym  price hq
-------------
appl 449.1 CA
ibm  108.2 NY
hp   24.5  CA
goog 890.3 CA
```

In addition, a table can be constructed directly using the structure below:

```
T <- ([] column1<-value1; column2<-value2; ...)
```

```
    T<-([]sym<-`appl `ibm `hp `goog;price<-449.1 108.2 24.5 890.3;hq<-4
2#'CANYCACA')
    T
sym  price hq
-------------
appl 449.1 CA
ibm  108.2 NY
hp   24.5  CA
goog 890.3 CA
```

A table can be queried using ELI's specialized query language, called ***esql*** that emulates standard SQL queries. The following example shows a query for finding the stock symbols (sym) and headquarters (hq) whose price is over 100. The query is encapsulated as a string passing to the function ***do*** which is later executed internally with ELI scripts.

```
    do 'SELECT sym,hq FROM T WHERE price>100'
sym  hq
-------
appl CA
ibm  NY
goog CA
```

4. ELI Compiler

ELI compiler ***ecce*** [5] translates ELI source code to C code. The generated C code is further compiled to a binary using the GCC compiler before it is executed with input data. The ELI compiler is designed for handling a subset of ELI language with the support of flat arrays. It is flexible to handle typical computation-intensive programs [6], such as Black-Scholes, K-means, and Hotspot.

The use of the compiler can be broken into several steps: (1) first write a valid ELI program (surrounded by "@."); (2) then define two system variables ***LPARM*** and ***RPARM*** (surrounded by "&"); and (3) finally invoke the compiler as *LPARM COMPILE RPARM* that COMPILE is a function taking two arguments.

```
@.z <- x add y
z <- x + y
@.
```

```
&LPARM C 1 6
'add EE'
&
```

```
&RPARM I 1 4
0 1 _1 0
&
```

As can be seen in the above simple example, LPARM defines function name alongside each type of arguments (E for floating-point numbers) and RPARM defines the starting value of indexing (i.e., 0) and shapes of the input arguments (1 _1 for a one-dimensional array with an unknown length, and 0 for a scalar).

```
/* v18 is result vector z,
 * v19 is input vector a,
 * v20 is input scalar b */
r0=v19.reall; /* length of result */
INCHEAPP3(v18,inchp2); /* alloc result */
p2 = (double *) v18.valp;
lo2 = (double *) v19.valp;
for(v1 =0; v1<r0; v1++) /* code for + */
       p2[v1] = lo2[v1]+v20;
```

The generated C code reflects what the compiler is told to do by understanding the two input arguments for the ELI function ***add*** are one is a vector (lo2) and another one is a scalar (v20).

5. Conclusion and Future Work

We have presented ELI whose language is an extension of standard APL1 having various new features including a brand new design with ASCII keys for array-based primitive functions, a dedicated system for mixing array programming and database query processing, and the support of a compiler for flat arrays. In the future, we plan to enrich the set of primitive functions to cope with those database-related data types, improve *esql* to support more database operations, and extend the compiler to handle new types, such as lists and tables.

References

1. ELI Homepage, https://fastarray.appspot.com
2. Hanfeng Chen and Wai-Mee Ching, ELI: a simple system for array programming, Vector J. Br. APL Assoc. Vol.26, No.1, p94-102, 2013.
3. International Organization for Standardization, ISO Draft Standard APL, ACM SIGAPL Quote Quad, vol.4, no.2, December, 1983.
4. ELI Primer, Section 2.2 Lists, https://fastarray.appspot.com/res/Eli_Primer.pdf
5. Hanfeng Chen, Wai-Mee Ching, and Laurie Hendren, An ELI-to-C Compiler: Design, Implementation, and Performance, Proceedings of the 4th ACM SIGPLAN International Workshop of Libraries, Languages and Compilers for Array Programming (ARRAY'17), pp. 9-16, June 2017.
6. ELI Compiler Examples, https://fastarray.appspot.com/compiler.html#examples

Contact:

Hanfeng Chen
e-mail: hanfeng.chen@mail.mcgill.ca

Wai-Mee Ching
e-mail: waimeeching@gmail.com

A Notation for APL Arrays

Adám Brudzewsky

APL has been the array programming language for over half a century, yet it lacks a way to write most arrays in a literal manner! Indeed, some argue that APL is an array notation, and nothing more is needed. However, while APL expressions are mostly confined to a single line (dfns and control structures being the only constructs that can span multiple lines), today's computer systems easily provide us with two spatial dimensions for our code. And APL's output has always spanned multiple lines for clarity:

```
      2 1⍴10 20
10
20
```

Now, if you have used APL at all, then I am sure you at some point used APL's output as input for subsequent execution. Of course, if we configured APL to output in the form of one–liner APL expressions, then this would work for arrays of higher rank than vectors too. But on the other hand, I am sure that we can agree as to the value of APL's output form, making it possible to see the arrays without mental gymnastics. Having an explicit input notation, that was as clear as the output form, seems a natural extension to the APL environment.

A NEED?

APL code is more than just primitives put together. Often, various constants or configurations appear too. And as soon as you want something more complex than a simple or slightly nested vector, a problem starts to appear. If you have used nested arrays at all, then I am sure you at some point have been frustrated by APL's default output form for such arrays:

```
      ((3 1 4)(1 5))(2 1⍴10 20)
  3 1 4  1 5   10
               20
```

You may have been saved by the ubiquitous box–drawing format, whether through the explicit usage of a utility function, or through a system setting:

```
┌→──────────────────────────────┐
│┌→──────────────────┐  ┌→──┐  │
│└→─────┐ └→─────┐    │↓10 │  │
│ │3 1 4│ │1 5    │    │20  │  │
│ │~    │ │~      │    │~   │  │
│ └─────┘ └───────┘    └────┘  │
└∊──────────────────────────────┘
```

It would be nice if we could input arrays like this, but it would be too tedious to type or edit, and as verbose as the format is, it is still ambiguous:

```
      '1'2'3'
┌→─────┐
│1 2 3 │
└+─────┘
      '1'2 3
┌→─────┐
│1 2 3 │
└+─────┘
```

A SOLUTION?

Behold the proposed array notation:

```
      ((3 1 4)(1 5))[10
                    20]
```

Sure, this might not look like much of a win, but now compare some tables:

```
      4 2⍴'Name' 'Age' 'Sex' 'Anton' 56 'M' 'Berta' 93 'F' 'Cäsar'
11 'M'
```

Did you spot the mistake? That `4 2⍴` should be `4 3⍴` instead! Maybe it is better to write:

```
      t←  'Name'   'Age' 'Sex'
      t←t,'Anton'  56    'M'
      t←t,'Berta'  93    'F'
      r←t,'Cäsar'  11    'M'
```

Did you spot the mistake this time? The first assignment should have read `t←⍉,` to prevent the second row from extending the initial vector. Anyway, you have probably written code like the above few times, and while tedious, it works acceptably for matrices. However, compare with the array notation:

```
      ['Name'   'Age' 'Sex'
       'Anton'  56    'M'
       'Berta'  93    'F'
       'Cäsar'  11    'M']
```

And once you want an array of rank or depth above 2, things get awkward. For example:

```
      2 2 3⍴(13⍴10)⊤⍥1E12
3 1 4
1 5 9

2 6 5
3 5 8
```

A piecemeal construction might involve a temporary variable:

```
t←⍉,3 1 4
t←,t1 5 9
m←⍉,2 6 5
m←,m3 5 8
t←t,m
```

Not only is it cumbersome, but rather error–prone as well. With the corresponding array notation:

```
[[3 1 4
  1 5 9]

 [2 6 5
  3 5 8]]
```

While the middle, empty, line is optional, it is allowed, and thus for most simple arrays, the array notation can actually contain the default display form of the array, only with some added lightweight annotation. Of course, you should not store very large arrays in this format. For those you will want a proper database. But for all the medium–sized arrays (many of which are really constants) that litter our code — this is very convenient.

Medium–sized arrays

An example is Dyalog APL's dyadic Stencil operator (⌹) which conceptually moves a stencil over an array, applying the left operand (here: `f`) to one neighbourhood at a time. It takes a right operand which is a matrix where the first row specifies the neighbourhood length for each dimension and the second row specifies the step size for each dimension when moving the stencil over the argument array. If we want a neighbourhood size of 3 rows and 2 columns, with a subsequent neighbourhood overlapping its neighbours by 1 element (that is, we want a step size of 2 rows and 1 column) then the full derived function would be `f⌹(2 2⍴3 2 2 1)` which isn't exactly easy to read. Compare:

```
f⌹[3 2
   2 1]
```

As almost everywhere else in APL, the array notation allows replacing line breaks with diamonds, and we can write this as `f⌹[3 2⋄2 1]` which, in my opinion, is quite readable.

Speaking of line breaks, it is quite common to embed medium–length texts in APL code, for example for interactive documentation purposes. The format is usually a vector of character vectors, and one must resort to multiple assignments just like with the matrix above. With the array notation, we can easily write a vector of vectors:

```
r←(
  '<terms>    space separated search terms'
  ''
  '-list      list all results'
  '-list=<n>  list no more than <n>'
  ''
)
```

Since diamonds and line break are equivalent, we can let multiple shorter texts share a line:

```
r←(
  '<terms>    space separated search terms' ⋄ ''
  '-list      list all results' ⋄ '-list=<n>  list no more than <n>'
  ''
)
```

The above values are "constants" in the sense that `r` gets the same value in any environment where the code would appear. However, it is worth noting that the notation is an integral part of the language, not just a notation for literal constant values. If we use formulas (that is, APL expressions), instead of literal numbers or characters, then, wherever the notation is executed (that is, the code is run), the array will encompass the computed values. You could see this as a form of "interpolation", using regular APL:

```
stats←[
  'Average'  ((+⌿x)÷≢x)
  'Extremes' (⌊⌿x ⋄ ⌈⌿x)
]
```

The last inner parenthesis is itself using this new notation to build a 2–element vector, and as such, it is equivalent to `((⌊⌿x)(⌈⌿x))`.

Dictionaries

The notation also includes the possibility of literal dictionaries, which are collections of name–value pairs, like APL2's name scopes, GNU APL's structured variables, and Dyalog APL's namespaces. These have a very simple and convenient syntax.

For example, consider this code:

GNU APL

```
info←38 ⎕CR 0
info.name←'Anton' 'Berta' 'Cäsar'
info.age←56 93 11
info.sex←'M' 'F' 'M'
info.meta←38 ⎕CR 0
info.meta.time←⎕TS
info.meta.author←'Adám'
```

Dyalog APL

```
info←⎕NS⍬
info.name←'Anton' 'Berta' 'Cäsar'
info.age←56 93 11
info.sex←'M' 'F' 'M'
info.meta←⎕NS⍬
info.meta.time←⎕TS
info.meta.author←'Adám'
```

With the array notation, it would be:

```
info←(
    name←'Anton' 'Berta' 'Cäsar'
    age←56 93 11
    sex←'M' 'F' 'M'
    meta←(
        time←⎕TS
        author←'Adám'
      )
)
```

These dictionaries are not limited to creation of constants. They can also conveniently be used to pass named arguments to a function. Many utility functions taking a vector of arguments for all kinds of parameters. A common problem is remembering the order of parameters, and how to supply a placeholder value so that the default setting is used. For example, Dyalog's HttpCommand utility has a function called GET that takes a right argument consisting of `URL Params Headers Cert SSLFlags Priority` with the default values `'' '' (0 2⍴⊂'') ⍬ 32 'NORMAL:!CTYPE-OPENPGP'` (although the empty elements are lenient about their shape and type). This means that if I just want to specify the URL and set the Priority, I would have to write:

```
resp←HttpCommand.Get myUrl ⍬ ⍬ ⍬ 32 myPri
```

Alternatively, I could create a GET request and set the priority later:

```
req←⎕NEW HttpCommand 'GET' ⋄ req.Priority←myPri
```

With the array notation, the syntax could conceivably be amended to allow:

```
resp←HttpCommand.Get(URL:myUrl ⋄ Priority:myPri)
```

Or we could create a request while immediately specifying my parameters:

```
req←⎕NEW HttpCommand(Command:'GET' ⋄ URL:myUrl ⋄ Priority:myPri)
```

Since Dyalog introduced the variant operator (⍠) which is based on Ken Iverson's Custom operator as seen in *A Dictionary of APL* (as `:`) and in J (as `!.`), there have been occasional calls for allowing this operator to act on primitives. For example, one would temporarily disable the comparison tolerance with `=⍠0` (`=⍠0` in dictionary APL and `=!.0` in J). However, this amounts to an ad–hoc addition to the language rather than a general extension. Using a temporary dictionary, we can hook into the existing `⎕CT` mechanism with `(⎕CT:0).=`. This also solves the problem of having multiple system variables affect a single function. For example, dyadic `⍳` is affected both by `⎕CT` and by `⎕IO`, but `(⎕CT:0⋄⎕IO:0).⍳` would set both.

APL CODE IN TEXT FILES

You have heard the news, right? The APL world is moving towards source code stored in text files rather than in binary workspace files, development is moving towards teams rather than being done by individuals, and companies increasingly have audits of their software. For all these reasons, many development tools are gaining importance, but only those that are made specifically for *your* specific APL system can deal with arrays (or, for that sake, code) stored in a workspace.

What kind of development tools?

- Versioning systems like Git, Mercurial, Subversion, etc.
- Hosting services like GitHub, GitLab, BitBucket, etc.
- Comparison utilities like diff, fc, Beyond Compare, etc.
- Editors like Notepad++, VS Code, Sublime Text, Atom, UltraEdit, Vim, emacs, etc.
- The list goes on and on…

Let us just take a single, simple example. I have two tables, `A` and `B`, that look pretty much the same:

```
      A B
  Name        Age    Name       Age
  Anton        56    Anton       56
  Berta        93    Bruno       93
  Cäsar        11    Caeser      11
  Dora          6    Dora         6
  Emil         61    Emil        67
  Friedrich    37    Fritz       37
  Gustav      103    Gustav     103
  Heinrich     95    Heinrich    95
  Ida          94    Ida         94
  Julius       58    Josef       53
  Kaufmann    110    Konrad     110
  Ludwig        2    Ludwig       2
  Martha       40    Martha      40
```

However, they are *not* identical:

```
      A ≡ B
0
      (⍴A) ≡ (⍴B)
1
      (≡A) = (≡B)
1
      +/, A ≢¨ B
21
```

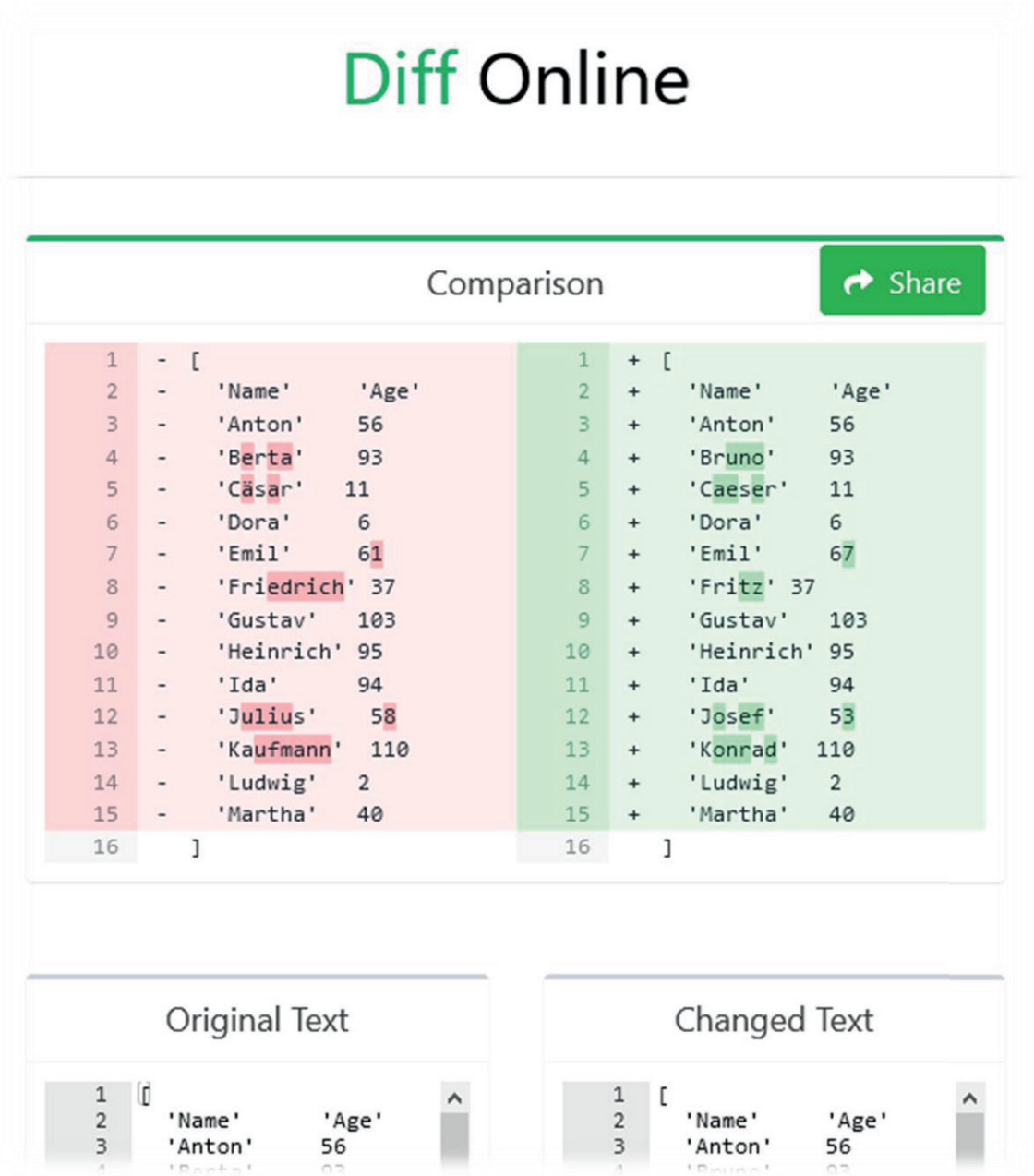
So, they have the same, shape and depth, but still differ in 7 places. Where and what exactly are the differences?

Dyalog APL comes with a model which can construct array notation for us. The next version will also include a user command to make this easier:

```
     ]repr A -f=apl
[
  'Name'       'Age'
  'Anton'      56
  'Berta'      93
  'Cäsar'      11
  'Dora'       6
  'Emil'       61
  'Friedrich'  37
  'Gustav'     103
  'Heinrich'   95
  'Ida'        94
  'Julius'     58
  'Kaufmann'   110
  'Ludwig'     2
  'Martha'     40
]
```

```
     ]repr B -f=apl
[
  'Name'      'Age'
  'Anton'     56
  'Bruno'     93
  'Caeser'    11
  'Dora'      6
  'Emil'      67
  'Fritz'     37
  'Gustav'    103
  'Heinrich'  95
  'Ida'       94
  'Josef'     53
  'Konrad'    110
  'Ludwig'    2
  'Martha'    40
]
```

Let us paste these into the free DiffOnline.net service:



The Link and Acre tools can create such files for you, or, again in the next version of Dyalog, you could write:

```
]repr A -f=apl -o=/tmp/A.txt
```

After that, you could use any 3rd party tool to compare them.

Alternatives?

Another indication of APL needing an array notation is the enormous success of the JSON format for all manner of communication, with both originator and recipient being able to be a human or a computer. JSON is JavaScript Object Notation, and considering that in Java–

Script, “everything is an object”, so too in APL where “everything is an array”, APLAN — APL Array Notation — is natural.

Why don’t we just adopt JSON as APL notation? Obviously, JSON does not have a way to represent arrays of rank 2 or higher, though one could envision an extension to JSON that can handle that. While superficially, the notations look similar, JSON uses comma (`,`) to separate elements of a list and to separate name–value pairs of an object. Comma already has a meaning in APL, and things like `[1,2,3]` would be ambiguous. Furthermore, we cannot have `[42]` be a one–element list (equivalent to `,42`) as this would clash with the existing bracket indexing and axis syntax.

Indeed, even with this proposed syntax, we have problems. If `[3 2⋄2 1]` is a 2–row matrix, then clearly `[3 2]` should be a 1–row matrix, but this cannot be, because that already has meaning, namely as bracket indexing (or a very unusual bracket axis). To avoid introducing ambiguity into the language, we must enforce a new rule: To use the new array notation, the square brackets or round parentheses must contain at least one line break or diamond. In most cases, this does not matter, as you would have multiple cells or elements anyway, but in the edge case of length–1 axes, it does.

Another option would be to add dedicated glyphs for the array notation, for example `⟦…⟧` could be used for collections of cells, and `⟨…⟩` for vectors. However, I think length–1 axes too rare to justify adding another four glyphs to the language — and thus to the keyboard — and lowering the already limited number of fonts that can be used for APL. While maybe unpleasant to the eye, a 1–row matrix can be written as `[3 2⋄]` or `[⋄3 2]` or even `[⋄3 2⋄]`. When writing arrays in a fully expanded form, over multiple lines, the problem does not present itself at all:

```
[
 3 2
    ]
```

The details

So, after all these examples, let us have a look at the formal rules for the notation. APL has a tradition of trying to maintain backwards compatibility. The array notation follows this tradition by ensuring that the only changes to the language are through giving meaning to previously invalid statements. We define *broken* to mean *interrupted by one or more diamonds or line breaks* (outside any dfns). With this definition, the added syntax consists of three constructs that are currently SYNTAX ERRORs:

- *broken* round parentheses: `( … ⋄ … )`
- *broken* square brackets: `[ … ⋄ … ]`
- *empty* round parentheses: `()`

And the added meanings are as follows:

- A *name–value pair* consist of a valid APL identifier, followed by a `:` and a value expression (`name:value`).
- A *broken* round parenthesis creates a namespace if every diamond/line break–separated statement is a *name–value pair*: `(Q:'everything' ⋄ ans:42)` is a namespace with `Q←'everything'` and `ans←42`.

- A *broken* round parenthesis creates a vector if every diamond/line break–separated statement is a value expression. In that case, every such statement forms an element in the resulting vector: `(1 2 ⋄ 3 4)` is equivalent to `(1 2)(3 4)` and `('A' ⋄ ⎕UCS 10 ⋄ 'B')` is equivalent to `'A'(⎕UCS 10)'B'`.
- A *broken* square bracket creates a an array, where every diamond/line break–separated statement forms a major cell (of rank 1 or higher) in the resulting array: `[1 2 ⋄ 3 4]` is equivalent to `2 2⍴1 2 3 4`.
- `()` represents a new empty namespace: `(⎕NS 0⍴⊂'')` in Dyalog APL or `(38 ⎕CR 0)` in GNU APL.

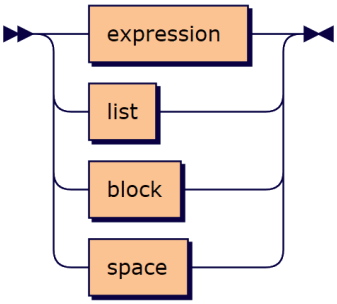
Value expressions are evaluated in the surrounding scope, similar to inline expressions in JavaScript's object notation, but as opposed expressions in `:Namespace` scripts. This is so that the notation can pick up existing values into a namespace. As opposed to what JavaScript does, any intermediate assignments are local to the value expression, similar to assignments in dfns. Global assignment is still available using `⎕THIS.name←value`, just as in dfns.

And that is really all there is to it. The array notation can be described using Extended Backus-Naur form, where an `expression` is any traditional APL expression:

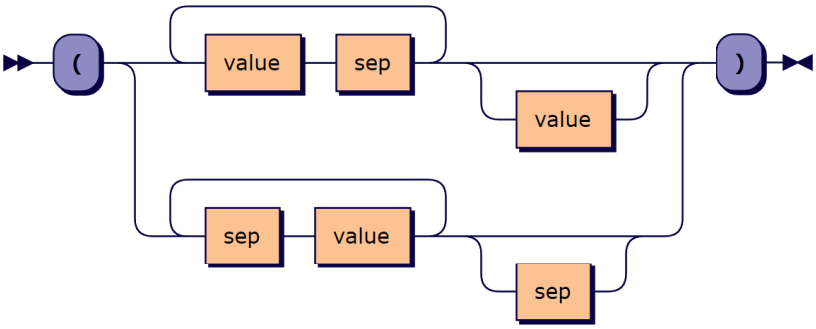
```
value ::= expression | list | block | space
list  ::= '(' ( ( value sep )+ value? | ( sep value )+ sep? ) ')'
block ::= '[' ( ( value sep )+ value? | ( sep value )+ sep? ) ']'
space ::= '(' sep? ( name ':' value ( sep name ':' value )* )? sep? ')'
sep   ::= [⋄#x000A#x000D#x0085]+
```

We can use this to draw a syntax diagram for the added notation:

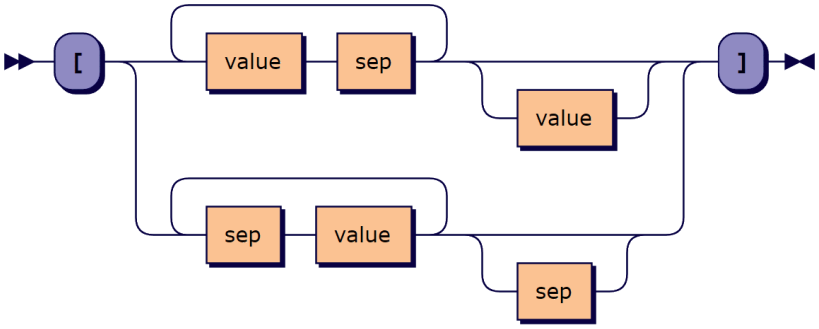
value:



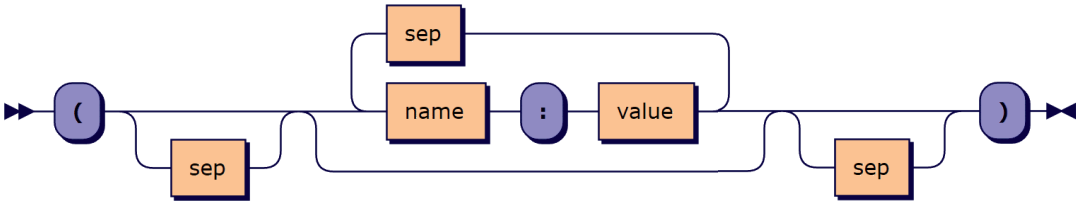
list:



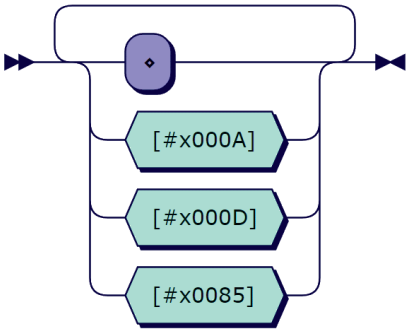
block:



space:



sep:



The past, the present, and the future

While all the details are available on APL Wiki, here is an abridged timeline of major events leading to where we are holding today:

1981 The NARS reference manual included round parentheses to informally delimit nested arrays.

2014 Dyalog CTO Morten Kromberg says that it probably is time to come up with a notation for constants.

2015 Phil Last presents a notation using square brackets for both *constants* of rank 2 or higher and for namespaces. Name–value pairs are delimited by ←. Adám Brudzewsky engages Phil Last in a dialogue that continues over the years.

2016 Phil Last adds escape sequences to strings found inside array notation.

2017 Adám Brudzewsky presents an alternative proposal using only round parentheses, and using : to delimit name–value pairs. This allows the notation to be a fully integrated part of the language.

2018 Adám Brudzewsky reintroduced square brackets for higher–rank arrays, repurposing round parentheses for a vector notation. Namespaces get a formal scoping rules.

2020 dzaima/APL adopts the array notation with a slight variation. Adám Brudzewsky demonstrates Dyalog's model for serialising to, and deserialising from the array notation.

Throughout the last half–decade or so, the Carlisle Group and Dyalog Ltd. Have been working on Acre and Link, their respective systems for integrating interactive development in an APL with storing APL code in text files. Both systems then need a notation for arrays. Over these years, the companies have been discussing the requirements, and converging on the notation described in this article. So, unless something major comes up, this notation is final. As mentioned above, the next version of Dyalog APL, will also feature a user command that makes it easy to try out the notation.

It is likely that native experimental support, not using an APL model, will be added to the Dyalog interpreter within the next year or two. Once work begins on this, there will be some further considerations to assignments inside namespace value expression. Currently, tradfns and dfns have mutually incompatible rules for when multiple names appear immediately to the left of an assignment arrow, without a parenthesis:

```
a b←c
```

Unlike some other implementations, like APL2, Dyalog allows this syntax for multiple assignment, that is, distributing a single array value to multiple variable names. However, this leads to an ambiguity when one of the names on the left of the assignment function has already been used for a defined function.

If the above code appears in a dfn or dop, any such predefined function is ignored. Its name is shadowed, and a new variable of that name takes its place. This means that `a b←c` is the same as `(a b)←c`, even if `a` or `b` is a function.

Outside of dfns and dops, any existing function `a` or `b` is used. This means that if `b` is a function, the expression `a b←c` denotes modified assignment, like how `a-←b` means `b⊣a←a-b`. Conversely, if `a` is a function, the expression `a b←c` includes usage of the pass–through value of the assignment, like how `-a←b` means `-b⊣a←b`.

To keep the array notation consistent between tradfns and dfns, a new unambiguous syntax will be required for modified assignment, with the modifying function being parenthesised together with the assignment arrow. Simultaneously, multiple assignment will require a parenthesis. Together, these two restrictions ensure unambiguous and consistent code. `a(b←)c` is a modified assignment equivalent to `c⊣a←a b c` where `b` is a function, and `a b←c` lets `a`, which must be a function, use the passthrough value from the assignment equivalent to `a c⊣b←c`.

Contact:

Adám Brudzewsky
e-mail: adam@dyalog.com

Was kann APL von R lernen?

Martin Barghoorn

1 Motivation

Wir erinnern uns, in APL konnte man numerische Daten und Text nicht einfach so verbinden. Nur wenn man vorher den numerischen Teil in Text oder den Text in Zahlen umgewandelt hatte, konnte man eine ein reine Text- bzw. Zahlenmatrix bekommen. Und die Überschriften der Spalten und die Bezeichnung der Zeilen waren Nummern bei der Zahlenmatrix. Und mit den Zahlen in der reinen Textmatrix konnte man nicht rechnen, man musste sie aktivieren. Erst APL2 bot diese langersehnte Möglichkeit, durch das Konzept der Kapselung (nested arrays) mit gemischten Datenstrukturen zu arbeiten. Dieses war für APL2 ein allgemeines Prinzip für alle Datentypen, ein spezieller Datentyp *Tabelle* wurde bisher nicht eigens vorgesehen.

Im R-System, das später entwickelt wurde, hat man speziell, um statistische Auswertungen zu erleichtern, einen eigenen Datentyp *data.frame* definiert und implementiert. Der Data.frame ist eine intelligente datenbankartige Tabelle, in der in den Spalten die Variable und in den Zeilen die Fälle angeordnet sind. Der data.frame wird standardmäßig beschriftet mit Spaltenüberschriften und Zeilennummern, er ist potentiell vom Typ *mixed,* d.h. in jeder Spalte können unterschiedliche Datentypen angeordnet sein. Innerhalb der Spalten ist der Datentyp jedoch einheitlich, wie auch in einer Datenbank- oder Exeltabelle. Die Beschriftungen des Data.frame sind ein eigenes Objekt vom Datentyp *Liste* aus 2 Elementen, das erste Element enthält die Zeilen- und das 2. Element die Spaltenüberschriften. Durch die R-Funktion *unlist* dieser beiden Listenelemente erhält man die Beschriftungen in reiner Form, also meist als Text (character). Außerdem gibt es in R die Möglichkeit, gemäß den Prinzipien der objektorientierten Programmierung den Objekten bestimmte Eigenschaften zu verleihen und diese auch per Definition zu verändern. Zum Beispiel gehört der Data.frame in die *Klasse* Data.frame und eine Matrix in die Klasse Matrix und Array. Vektoren, Matrizen und Arrays dürfen nicht mixed sein, dh. alle Elemente müssen denselben Datentyp haben.

2 Vergleich APL – R mit einem Beispiel

```
Tabellen und Datenobjekte beschriften
    Version APL2, Definition einer Tabelle
    name←'eins' 'zwei' 'drei' 'vier' 'fuenf'
    dat←7+⍳5 ⋄ dat            ⍝ Datendefinition
8 9 10 11 12

Verbinden von Überschriften (Labels) und Daten
vek←name,[.5]dat  ⋄  vek
 eins zwei drei vier fuenf
    8    9   10   11    12
    ⍴vek      ⍝ der Vektor ist eine Matrix, 2 Zeilen
2 5

Gezielte Suche im Vektor, welche Zahl steht beim Label vier?
 (vek[1;]∊⊂'vier')/vek      ⍝ nur nur die 1. Zeile
 vier
   11
```

Vorläufiges Fazit: ein beschrifteter Vektor lässt sich in APL2 derzeit nicht definieren.

```
Version mit R, Definition eines Vektors
>name<-unlist(strsplit("eins zwei drei vier fuenf", " "))
# Leerzeichen ist das Trennzeichen
name
[1] "eins"  "zwei"  "drei"  "vier"  "fuenf"
# R-Funktionen strsplit und unlist, werden angewandt, da strsplit
eine Liste erzeugt, muss danach mit unlist „entlistet" werden
> vek<-7+1:5 ; vek     # Definition mit Indexfunktion :
[1]  8  9 10 11 12
 > names(vek)<-name   ; vek
 eins  zwei  drei  vier fuenf
    8     9    10    11   12
Welche Zahl steht in Spalte ,'vier'?
> vek['vier']
vier
  11
> vek['vier']+99   # Finden und sofort rechnen
vier
 110
> vek['vier']<-999  ; vek  # Selektive Zuweisung
eins zwei drei vier fuenf
   8    9   10  999   12
```

```
Funktion match (APL - dyadisches Iota)
> element<-vek[3] ; element
drei
  10
> match(vek,element)     # an dritter Position
[1] NA NA  1 NA NA
```

Suchen und ersetzen mit match

```
> vek[!is.na(match(vek,10))]<-123 ; vek
 eins  zwei  drei  vier fuenf
    8     9   123   999    12
```

Anwendung statistischer Kenngrößen

```
> sum(vek) ; mean(vek)
[1] 125
[1] 12.5
```

Fazit: die Beschriftung stört beim Rechnen nicht.

Nun die nächste Dimension, 2-dimensionale Matrix:

```
> mat<-matrix(1:12) ; dim(mat)<-3:4
> mat
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
> dimnames(mat)<-list(NULL, c("IMI", "ATA", "PERSIL", "FEWA"))
> mat
     IMI ATA PERSIL FEWA
[1,]   1   4      7   10
[2,]   2   5      8   11
[3,]   3   6      9   12
> dim(mat)
[1] 3 4
> mat[,'PERSIL']           # Spalte PERSIL
[1] 7 8 9
> mat[,'PERSIL'] *17   # suchen und losrechnen
[1] 119 136 153
> mat[,'PERSIL']<-mat[,'PERSIL']*17  # Zuweisung
# selective Assigment
> mat   # nach Zuweisung
     IMI ATA PERSIL FEWA
[1,]   1   4    119   10
[2,]   2   5    136   11
[3,]   3   6    153   12
```

In gleicher Weise funktioniert es bei R auch in der nächsten Dimension: beim Array

3 Stapeln und retour

```
Funktion stack ist nur beim data.frame definiert
> data<-as.data.frame(mat) ; data
# eine Matrix wird data.frame, äußerlich ähnlich
  IMI ATA PERSIL FEWA
1   1   4    119   10
2   2   5    136   11
3   3   6    153   12
> data$PERSIL
# Zugriff mit $ nur beim data.frame möglich
[1] 119 136 153
> s1<- stack(data) ; dim(s1)
[1] 12  2
> s1
   values    ind
1       1    IMI
2       2    IMI
3       3    IMI
4       4    ATA
5       5    ATA
6       6    ATA
7     119 PERSIL
8     136 PERSIL
9     153 PERSIL
10     10   FEWA
11     11   FEWA
12     12   FEWA
> s2<-unstack(s1)  ; s2 # data retour
  IMI ATA PERSIL FEWA
1   1   4    119   10
2   2   5    136   11
3   3   6    153   12
> mat==s2
      IMI  ATA PERSIL FEWA
[1,] TRUE TRUE   TRUE TRUE
[2,] TRUE TRUE   TRUE TRUE
[3,] TRUE TRUE   TRUE TRUE
```

Mein Vorschlag lautet: Implementierung eines Objektes APL-*Tabelle* in einem Workspace oder einer externen Funktion.

Kontakt:

Martin Barghoorn
E-mail: Martin@Barghoorn.com

Theoretische Physik

Modellierung zeigt, welche Quantensysteme sich für Quantensimulationen eignen

Berlin. Eine gemeinsame Forschungsgruppe um Prof. Jens Eisert von der Freien Universität Berlin und des Helmholtz-Zentrum Berlin (HZB) hat einen Weg aufgezeigt, um die quantenphysikalischen Eigenschaften komplexer Festkörpersysteme zu simulieren. Und zwar mithilfe von komplexen Festkörpersystemen, die experimentell untersucht werden können. Die Studie wurde in der renommierten Fachzeitschrift Proceedings of the National Academy of Sciences of the United States of America (PNAS) veröffentlicht.

„Das eigentliche Ziel ist ein robuster Quantencomputer, der auch bei Fehlern stabile Ergebnisse erzeugt und diese Fehler korrigiert", erklärt Jens Eisert, Professor an der Freien Universität Berlin und Leiter einer gemeinsamen Forschungsgruppe am HZB. Bislang ist die Entwicklung robuster Quantencomputer noch in weiter Ferne, denn Quantenbits reagieren extrem empfindlich auf kleinste Schwankungen der Umgebungsparameter. Doch nun könnte ein neuer Ansatz Erfolg versprechen:

Eine Postdoktorandin und ein Postdoktorand aus der Gruppe um Jens Eisert, Maria Laura Baez und Marek Gluza, haben eine Idee von Richard Feynman aufgegriffen, einem genialen US-amerikanischen Physiker der Nachkriegszeit. Feynman hatte vorgeschlagen, reale Systeme aus Atomen mit ihren quantenphysikalischen Eigenschaften für die Simulation anderer Quantensysteme heranzuziehen. Diese Quantensysteme können aus perlenkettenartig aufgereihten Atomen mit besonderen Spin-Eigenschaften bestehen, geeignet aber wären aber auch Ionenfallen, Rydbergatome, supraleitende Qbits oder Atome in optischen Gittern. Gemeinsam ist ihnen, dass man sie im Labor erzeugen und auch kontrollieren kann. Ihre quantenphysikalischen Eigenschaften könnten dazu herangezogen werden, um das Verhalten anderer Quantensysteme vorherzusagen. Doch welche Quantensysteme wären gute Kandidaten? Gibt es eine Möglichkeit, dies vorab herauszufinden?

Das Team um Eisert hat diese Frage nun mit einer Kombination aus mathematischen und numerischen Methoden untersucht. Tatsächlich zeigte die Gruppe, dass der sogenannte dynamische Strukturfaktor solcher Systeme ein mögliches Werkzeug ist, um Aussagen über andere Quantensysteme zu treffen. Dieser Faktor bildet indirekt ab, wie sich Spins oder andere Quantengrößen mit der Zeit verhalten, er wird durch eine Fouriertransformation errechnet.

„Diese Arbeit schlägt eine Brücke zwischen zwei Welten", erklärt Jens Eisert. „Da ist zum einen die Gemeinschaft der Kondensierten Materie, die Quantensysteme untersucht und die daraus neue Erkenntnisse gewinnt – und zum anderen die Quanteninformatik – die sich mit Quanteninformation befasst. Wir glauben, dass große Fortschritte möglich werden, wenn wir die beiden Welten zusammenbringen", sagt der Wissenschaftler.

(DOI: 10.1073/pnas.2006103117)

(Quelle: Helmholtz-Zentrum Berlin für Materialien und Energie GmbH, Dr. Antonia Rötger)

APL-Journal
39. Jg. 2020, ISSN 1438-4531
Herausgeber: Prof. Dr. Dieter Kilsch, APL-Germany e.V., Mannheim, Homepage: https://apl-germany.de/, E-Mail: d.kilsch@th-bingen.de
Redaktion: Dipl.-Volksw. Martin Barghoorn (verantw.), Lückhoffstr. 8, 14129 Berlin, E-Mail: Martin@Barghoorn.com
Verlag: RHOMBOS-VERLAG, Berlin, Postfach 67 02 17, D-10207 Berlin, Tel. (030) 261 9461, eMail: verlag@rhombos.de, Internet: https://rhombos.de/
Erscheinungsweise: halbjährlich
Erscheinungsort: Berlin
Druck: dbusiness.de GmbH, Berlin

Allgemeine Informationen

(Stand 2019)

APL-Germany e.V. ist ein gemeinnütziger Verein mit Sitz in Düsseldorf. Sein Zweck ist es, die Programmiersprache APL, sowie die Verbreitung des Verständnisses der Mensch-Maschine Kommunikation zu fördern. Für Interessenten, die zum Gedankenaustausch den Kontakt zu anderen APL-Benutzern suchen, sowie für solche, die sich aktiv an der Weiterverbreitung der Sprache APL beteiligen wollen, bietet APL-Germany den adäquaten organisatorischen Rahmen.

Auf Antrag, über den der Vorstand entscheidet, kann jede natürliche oder juristische Person Mitglied werden. Organe des Vereins sind die mindestens einmal jährlich stattfindende Mitgliederversammlung sowie der jeweils auf zwei Jahre gewählte Vorstand.

1. Vorstandsvorsitzender

Prof. Dr. Dieter Kilsch,
Dumontstraße 12, 55313 Mainz,
Tel. 06131 6982200, E-Mail:
d.kilsch@th-bingen.de.

2. Vorstandsvorsitzender:

Martin Barghoorn
Lückhoffstr. 8, 14129 Berlin,
E-Mail: Martin@Barghoorn.com

Schatzmeister

Jürgen Beckmann
Im Freudenheimer Grün 10
68259 Mannheim
Tel. 0621 7 98 08 40,
eMail: JBecki@onlinehome.de

Beitragssätze

Persönliche Mitglieder:

Natürliche Personen	32,- EUR*
Studenten / Schüler	11,- EUR*

Institutionelle Mitglieder:

Jurist./natürl. Pers.	500,- EUR*

* Jahresbeitrag

Bankverbindung
BVB Volksbank eG Bad Vilbel
BLZ 518 613 25
Konto-Nr. 523 2694

Hinweis:
Wir bitten alle Mitglieder, uns Adressänderungen und neue Bankverbindungen immer sofort mitzuteilen. Geben Sie bei Überweisungen den Namen und/oder die Mitgliedsnummer an.

https://apl-germany.de/

www.apl-germany.de
APL Germany e.V.