Hypercomplex Quotients in APL

Bob Smith
Sudley Place Software
Originally Written
7 Apr 2016
Updated
11 Apr 2018

Introduction

Division of non-commutative numbers involves a choice. If `a` and `b` are non-commutative numbers (that is, Quaternions or Octonions), then `a÷b` may be calculated either as `a×÷b` (the right quotient) or as `(÷b)×a` (the left quotient), where `÷b` is as always `(+b)÷b×+b`. Which choice is made has wider implications than just that primitive function.

As division is used internally (directly or indirectly) in many primitives (e.g., GCD, LCM, Residue, Encode, and Base Value) on non-commutative numbers, it is important to determine which quotient to use in each case.

Prerequisite Reading

The paper "Hypercomplex Notation in APL"[4] provides a summary of the notation used in this paper for Hypercomplex numbers. The paper "Hypercomplex GCD in APL"[1] is necessary reading in order to understand this paper as it introduces algorithms for Floor, Residue, and GCD, and it resolves the question of Fractionality.

Division Example

These two quotients (left and right) occur with non-commutative numbers only.

They come into play in the following somewhat counterintuitive example: suppose `a` and `b` are Quaternions and `c←a×b`, and we're using the default of the right quotient for the Division primitive. The quotient `c÷b` is reliably `a` because

```
c÷b ↔ (a×b)÷b
    ↔ ((a×b)×+b)÷b×+b
    ↔ (a×b×+b)÷b×+b
    ↔ a
```

and because `b×+b` is a real number, and multiplication of Quaternions is associative.

However, `c÷a` is not always `b` because

```
c÷a ↔  (a×b)÷a
    ↔  ((a×b)×+a)÷a×+a
    ↚/→ ((b×a)×+a)÷a×+a
```

and because Quaternion multiplication is not commutative. Instead, the left quotient of `c÷a` is

```
c÷a ↔ (a×b)÷a
    ↔ ((+a)×a×b)÷(+a)×a
    ↔ (((+a)×a)×b)÷(+a)×a
    ↔ b
```

because `(+a)×a` is a real number, and multiplication of Quaternions is associative. This indicates that the choice of right or left quotients is

one over which the programmer (rather than the system implementor) should have direct control without having to write out the explicit forms such as in the examples above.

Division Quotients

In order to deal with the choice of left or right quotients when dividing non-commutative numbers, a new system variable ⎕LR is introduced. This variable may assume the value of 'r' or 'l' to indicate that the right (resp. left) quotient should be returned from division. The default value is 'r'.

Correspondingly, the Variant operator (⍠) has been extended to allow ⎕LR to be specified in a shorthand form on selected primitive functions and in a longhand form on all primitive functions as well as user defined functions and operators, anonymous functions, and derived functions. For example (in shorthand form),

```
      (a8 b8)←<?2 8⍴20
      c8←a8×b8
      b8=c8÷⍠'l' a8
1
      a8=c8÷⍠'r' b8
1
```

or when used on a user-defined function/operator, anonymous, or derived function (in longhand form)

```
      {⎕LR}⍠('LR' 'l')
l
      {⎕LR}⍠('LR' 'r')
r
```

Floor Primitive

From the conclusion of the earlier paper “Hypercomplex GCD”[1], the

only Floor function that satisfies Fractionality on Quaternions is the one defined by Hurwitz and then only with its domain (and range) extended to Hurwitz Quaternions. Because Floor on Octonions does not have Fractionality, this primitive is not defined on Octonions.

```
Z←UF R;F T
⍝ ⌊R for Hypercomplex numbers
⍝   using Hurwitz's Floor function
⍝   returning the nearest Hurwitzian (half-)integer
⍝ Scalar R
⍝ Sensitive to ⎕CT
F←>R ⋄ Z←<⌊F+(1+⍳F)÷2
F←⌊F ⋄ T←< F+(1+⍳F)÷2
:if (|R-Z)>|R-T ⋄ Z←T ⋄ :end
```

Implementation Note

In the Alpha version of NARS2000, the two definitions of Floor are distinguished by the value of `⎕FEATURE[3]`. If that value is `0`, then McDonnell's version of Floor is used for Complex numbers on Floor, Ceiling, Residue, GCD, LCM, Encode, and Base Value, and Quaternions on Floor, Ceiling, Residue, GCD, LCM, Encode, and Base Value all signal a `DOMAIN ERROR`. If that value is `1`, then Hurwitz's version of Floor, Ceiling, Residue, GCD, LCM, Encode, and Base Value is used for Complex and Quaternion numbers.

Residue

The Residue function (`L|R`) is defined on Quaternions by splitting it into two special cases (`L=0 1`) and one general case (none of the above), a naïve version of which is as follows:

```
Z←L UR1 R
⍝ L|R for Hypercomplex numbers
⍝   using Hurwitz's Floor function
⍝ Scalar L and R
⍝ Sensitive to ⎕CT
→(L(=⌊0) 0 1)/L0 L1 ⍝ a.k.a. L=0 1 with ⎕CT←0
Z←L×1∇R÷L ⋄ →0
L0:Z←R ⋄ →0
L1:Z←R-UF R
```

This version works perfectly fine on commutative numbers, however it has problems on non-commutative numbers depending upon the setting of ⎕LR – for example,

```
      a←1i2x ⋄ b←1i5j2x
      ⎕LR←'l' ⋄ ⎕←c1L←a UR1 b
¯1r2i¯1r2j1r2k¯1r2
      ⎕LR←'r' ⋄ ⎕←c1R←a UR1 b
¯1r2i¯1r2j¯7r10k1r10
      ⎕LR←'l' ⋄ a UR1 b-c1L
0
      ⎕LR←'r' ⋄ a UR1 b-c1R
0j24r25k¯7r25
```

where `c1R` is clearly a bogus result as it is not even a Quaternion Integer, not to mention the fact that it doesn't satisfy the most basic identity of Residue (`0=a|b-a|b`). Note that the arguments to this function (and the ones below) are all expressed as Rational numbers[3] (e.g., `3x` or `1r3`) so as to avoid inexact floating point results due to round off error in the division.

The problem with `UR1` and `⎕LR←'r'` lies with the statement `Z←L×1∇R÷L` where we divide by `L` on one side returning a right quotient and multiply by `L` on the left side. To be consistent, we need to multiply on the same side as the quotient we get from division, that

is, we need to use all right divisions and multiplications or all left divisions and multiplications. In other words, we need to make this statement sensitive to ⎕LR. One way to accomplish this is as follows:

```
Z←L UR R
⍝ L|R for Hypercomplex numbers
⍝   using Hurwitz's Floor function
⍝ Scalar L and R
⍝ Sensitive to ⎕CT and ⎕LR
→(L(=⌊0) 0 1)/L0 L1 ⍝ a.k.a. L=0 1 with ⎕CT←0
:select ⎕LR
  :case 'l' ⋄ Z←L× 1∇R÷L ⋄ →0 ⍝ ∇ is recursive call
  :case 'r' ⋄ Z←L×⍨1∇R÷L ⋄ →0 ⍝ ⍨ is Commute oper
:end
L0:Z←R ⋄ →0
L1:Z←R-UF R
```

This definition supersedes the one from “Hypercomplex GCD”.

Essentially, this provides two results from Residue depending upon the value of ⎕LR, one for left quotients and one for right quotients:

```
      a←1i2x ⋄ b←1i5j2x
      ⎕LR←'l' ⋄ ⎕←c2L←a UR b
¯1r2i¯1r2j1r2k¯1r2
      ⎕LR←'r' ⋄ ⎕←c2R←a UR b
¯1r2i¯1r2j1r2k1r2
```

Also, these results check out as valid residues:

```
      ⎕LR←'l' ⋄ a UR b-c2L
0
      ⎕LR←'r' ⋄ a UR b-c2R
0
```

The function UR is implemented in NARS2000 as the Residue primitive function on Quaternions. This primitive is not defined on Octonions because it is defined in terms of the Floor primitive which is not defined on Octonions.

Encode

This primitive function (L⍷R) also has a naïve definition on Hypercomplex numbers expressed as an APL function as follows:

```
Z←L UE1 R;⎕CT I
⍝ L⍷R for Hypercomplex numbers
⍝   using Hurwitz's Residue function
⍝ Scalar/Vector L, Scalar R
⍝ Sensitive to ⎕LR
⎕CT←0
L←1/L ⋄ Z←(⍴L)⍴0
:for I :in ⌽⍳⍴L
  Z[I]←L[I] UR R
  :leaveif (L[I]=0)∨R=Z[I]
  R←(R-Z[I])÷L[I]
:end
```

For example, using randomly chosen Quaternions:

```
      ⎕←L←0,<?5 4⍴10x
0 3i4j2k5 9i10j4k7 4i1j9k9 1i4j6k5 8i7j7k4
      ⎕←R←<?4⍴50x
8i10j30k40
      ⎕LR←'r' ⋄ ⎕←ZR←L UE1 R
0 0 0 0 3i1k2 ¯1i¯5j¯1k5
```

Note that we prepend a zero to L so as to be able to invert the function.

This algorithm uses two primitives (Residue and Division) sensitive to ⎕LR. However, trying the other combinations of one 'l' and the other 'r' in either order as Variant operator operands to those two primitives yields values that are not Hurwitzian Quaternions. The combination of both primitives calculated with left quotients does yield an integral result:

```
      ⎕LR←'l' ⋄ ⎕←ZL←L ⌸E1 R
0 0 0 0 3i¯1j2k1 ¯5i¯2j4k¯1
```

Using the Base Value primitive to convert these two results back into scalars depends on how one defines Base Value. For example, if Base Value is defined as follows:

```
      WR←×/⍤(⍳-0..¯1+⍴1/L)↑⍤⊂L
      R=ZR+.×WR
1
```

or if Base Value is defined as follows:

```
      WL←⍳1,×\⍳1↓L
      R=WL+.×ZL
1
```

Note the switched arguments between the two examples.

While the second definition looks cleaner, it doesn't represent how one normally thinks of constructing the weighting vector for Base Value. Essentially, in the definition of the weighting vector WR, for (say) a three-element L, its values are defined as follows:

```
(×/L[1 2 3]),(×/L[2 3]),L[3],1
```

whereas the corresponding weighting vector WL is defined as follows:

```
(×/L[3 2 1]),(×/L[3 2]),L[3],1
```

If multiplication in this context were commutative, these two weighting vectors would be the same, but it is not. As can be seen, the weighting vector `WR` multiplies the values in `L` in the correct order as they are encountered while scanning `L` from right to left, whereas `WL` reverses their order before multiplying them which rules it out as a definition of Base Value.

Thus, Base Value is a right quotient primitive function only.

If there's only one way to define the Base Value primitive (with `⎕LR←'r'`), there is only one way to define the Encode primitive (with `⎕LR←'r`). This corresponds to the following algorithm:

```
Z←L UE R;⎕CT ⎕LR I
⍝ L⊤R for Hypercomplex numbers
⍝   using Hurwitz's Residue function
⍝ Scalar/Vector L, Scalar R
⎕CT←0 ⋄ ⎕LR←'r'
L←1/L ⋄ Z←(⍴L)⍴0
:for I :in ⌽⍳⍴L
  Z[I]←L[I] UR R
  :leaveif (L[I]=0)∨R=Z[I]
  R←(R-Z[I])÷L[I]
:end
```

The function `UE` is implemented in NARS2000 as the Encode primitive function on Quaternions. This primitive is not defined on Octonions because it is defined in terms of the Residue primitive which is not defined on Octonions.

Base Value

In order to enable the identity `R≡(0,L)⊥(0,L)⊤R`, care must be taken as to how this primitive calculates its result. In particular, its weighting vector must be calculated as described above and the multiplication with the value in the right argument with the weighting vector from the left argument must be done with the right argument on the left and the weighting vector on the right.

```
Z←L UB R;W
⍝ L⊥R for Hypercomplex numbers
⍝ Scalar/Vector L, Scalar R
W←×/¨(⌽-0..¯1+⍴1/L)↑¨⊂L
Z←R+.×W
```

The function `UB` is implemented in NARS2000 as the Base Value primitive function on Quaternions. This primitive is not defined on Octonions because the inverse function Encode is defined in terms of the Residue primitive which is not defined on Octonions.

GCD

Greatest Common Divisor on Quaternions is defined in "Hypercomplex GCD" as follows:

```
Z←L UG R;T ⎕CT
⍝ L∨R for Hypercomplex numbers
⍝   using Hurwitz's Residue function
⍝ Scalar L and R
⍝ Sensitive to ⎕LR
⎕CT←1E¯10   ⍝ FP numbers only
:repeat     ⍝ Euclidean Algorithm
  T←L
  L←L UR R
  :Assert (|L)<|T
  R←T
:until (|L)≤4E¯15   ⍝ Again, FP numbers only
⍝ Rotate R into the first quadrant
⍝  or first two bi-quadrants
Z←rotateGCD R
```

The GCD function satisfies the following identities on commutative and non-commutative numbers (with `IsHalfInt←{∨≠∧/0 1∘.=2|2×>⍵}`):

```
0=L∨R|L
0=L∨R|R
IsHalfInt L÷L∨R
IsHalfInt R÷L∨R
```

The function `UG` is implemented in NARS2000 as the GCD primitive function on Quaternions. This primitive is not defined on Octonions because it is defined in terms of the Residue primitive which is not defined on Octonions.

LCM

The Least Common Multiple primitive (`L∧R`) is defined as `(L×R)÷L∨R`.

The LCM function satisfies the following identities on commutative numbers:

```
0=L|L∧R
0=R|L∧R
IsHalfInt (L∧R)÷L
IsHalfInt (L∧R)÷R
```

but because Residue and Division are sensitive to the value of ⎕LR, so are these identities, and in a complicated way. In particular, if the LCM is calculated with ⎕LR←'r', the first and third identities are valid with ⎕LR←'l', and if the LCM is calculated with ⎕LR←'l', the second and fourth are valid with ⎕LR←'r'.

That is,

```
⎕LR←'r' ⋄ Z←L∧R
⎕LR←'l' ⋄ 0=L|Z
⎕LR←'l' ⋄ IsHalfInt Z÷L

⎕LR←'l' ⋄ Z←L∧R
⎕LR←'r' ⋄ 0=R|Z
⎕LR←'r' ⋄ IsHalfInt Z÷R
```

The corresponding function is

```
Z←L UL R
⍝ L∧R for Hypercomplex numbers
⍝   using Hurwitz's GCD function
⍝ Scalar L and R
⍝ Sensitive to ⎕LR
Z←(L×R)÷L UG R
```

The function UL is implemented in NARS2000 as the LCM primitive function on Quaternions. This primitive is not defined on Octonions

because it is defined in terms of the GCD primitive which is not defined on Octonions.

Conclusion

Division Quotients come into play not just with Division but also with other primitives whose definition uses Division either directly or indirectly. The system var ⎕LR is introduced so as to enable the programmer to control the choice of such quotients and results. The programmer needs to take care when using the results of primitive functions sensitive to ⎕LR.

Encode and Base Value are right quotient primitives only. GCD, LCM, and Residue are sensitive to the value of ⎕LR and in general they each produce different results depending upon that system variable.

Online Version

This paper is an ongoing effort and can be out-of-date the next day. To find the most recent version, goto http://sudleyplace.com/APL/ and look for the title of this paper on that page. Related papers such as "Hypercomplex Notation in APL"[4], "Hypercomplex GCD in APL"[1], "Hypercomplex Numbers in APL"[2] as well as "Rational & Variable-precision Floating Point Numbers"[3] may be found in the same place.

Executable Version

All of the above APL functions may be executed in NARS2000, an experimental APL interpreter available for free as Open Source software.

The latest released version of the NARS2000 software may be found in http://www.nars2000.org/download/ in either 32- or 64-bit versions.

This software runs natively under Microsoft Windows XP or later as well as any Linux or Mac OS version which supports Wine (32-bit only) which acts as a translation layer.

The choice of which Floor function to invoke on Hypercomplex numbers is under user-control. This choice applies not only to the Floor primitive, but also all other primitive functions directly of indirectly sensitive to Floor. NARS2000 uses McDonnell's Floor function when `⎕FEATURE[3]←0` and Hurwitz's Floor function when `⎕FEATURE[3]←1`.

When using McDonnell's Floor function, all primitives that depend on Floor on Quaternions signal a `DOMAIN ERROR`. Only when using Hurwitz's Floor function does the system produce valid results.

References

1. “Hypercomplex GCD in APL”, http://www.sudleyplace.com/APL/HyperComplex GCD in APL.pdf
2. Hypercomplex Numbers in APL”, http://www.sudleyplace.com/APL/HyperComplex Numbers in APL.pdf
3. “Rational & Variable-precision Floating Point Numbers”, http://www.sudleyplace.com/APL/Rational & Variable-Precision FP.pdf
4. Hypercomplex Notation in APL”, http://www.sudleyplace.com/APL/HyperComplex Notation in APL.pdf