When To Use ⎕CT With Rationals?

Bob Smith
Sudley Place Software

Originally Written
31 Sep 2017
Updated
14 Nov 2017

The Question

Should ⎕CT be used when comparing a Rational number with an Integer or Rational number?

The following (symmetric) table lays out the existing choices (ignoring Hypercomplex numbers), almost all of which use ⎕CT for comparisons:

	INT	FLT	MPIR	MPFR
INT	N	Y	**?**	Y
FLT	Y	Y	Y	Y
MPIR	**?**	Y	**?**	Y
MPFR	Y	Y	Y	Y

INT	= Fixed-Precision 64-bit Integer
FLT	= Fixed-Precision 64-bit Floating Point
MPIR	= Multiple-Precision Integer/Rational
MPFR	= Multiple-Precision Floating Point

The above table applies to the usual dyadic ⎕CT-sensitive comparison functions (≡≢=<≤=≥>≠⍳∊⍸⍷) all of which normally use ⎕CT on FLTs and MPFRs. All comparisons are assumed to be with both arguments non-zero so absolute tolerance is not an issue.

In case you are not familiar with Rational numbers, they are stored as as two Multiple-Precision Integers – numerator and denominator – and written as, for example, 2r3 which is an exact version of 2÷3. The range of Multiple-Precision Integers is limited only by the amount of available workspace.

Examples

```
      ⎕CT←1E¯10
      ⎕←af←1+1E¯15
1.000000000000001
      ⎕←bf←1-1E¯15
0.999999999999999
      af=bf
1
      ⎕←ar←1+1E¯15x
1000000000000001r1000000000000000
      ⎕←br←1-1E¯15x
999999999999999r1000000000000000
      ar=br
????
```

Near Integer Functions

This question also applies to other primitives such as **Floor** and **Ceiling** where ⎕CT is normally used to decide the result. In particular, in the above example, what is the value of ⌈ar (1 or 2) or ⌊br (0 or 1) – should they be sensitive to ⎕CT?

Integer-Only Functions

Other functions such as **GCD**, **LCM**, **Residue**, and **Encode** on FLTs/MPFRs employ some form of Comparison Tolerance (perhaps not exactly `⎕CT`) to decide when to terminate. These functions reference Comparison Tolerance in two ways: directly through some form of (Fixed System-wide?) Comparison Tolerance and indirectly as `⎕CT` through their reliance on Floor and/or Ceiling. How should they treat MPIRs?

I view these primitives as fundamentally **Integer-only** functions not only because that's their fundamental domain, but that should be their only domain. In fact, I go so far as to suggest banning FLTs/MPFRs (`DOMAIN ERROR`) on these functions, but that's a separate topic for discussion.

Integer Tolerance

On the other hand, we should continue to use **Integer Tolerance** on all kinds of numeric datatypes to detect whether or not a non-integer is close enough to use as an integer such as with the left argument to the various structural primitives.

The Arguments

Pro: As MPIRs are dense in the real number line, they can be arbitrarily close to Integers as well as to each other (unlike INT v. INT comparisons): so `⎕CT` should be used because MPIRs are dense. If you don't want to use `⎕CT` for such comparisons, set it to zero, either explicitly or through the Variant operator on a primitive-by-primitive basis.

Con: As opposed to FLTs, MPIRs are meant to be exact and not an approximation to a range of FLTs which are limited by the IEEE-754

Standard to 53 bits of precision: so `⎕CT` should not be used because MPIRs are exact, not a stand-in for an infinite range of more precise numbers. That is, no matter how close together are two MPIRs, they are still different exact numbers. If you want to use `⎕CT` for such comparisons, convert the INTs/MPIRs to MPFRs (using `{'v' ⎕DCω}`) and then compare them.

What Do You Think?

As you can see, my arguments revolve around dense v. exact and I can't decide which is more important here. At one time or another, I have implemented it both ways. Recently, I noticed that the current implementation is of both minds! Help!

Bob Smith
bsmith@sudleyplace.com
304-707-7963