WHAT ARE THE LEVEL 2 DATUMS?
VERSION 1

JOHN PRYCE

1. Introduction

In revising the draft standard text I have struggled with a boring but crucial point: *When are two level 2 datums the same?* As any computer scientist or mathematician knows, being too picky about the meaning of equality turns readers or listeners off a topic pretty fast; but not being picky enough leaves them with only partial understanding, and liable to draw wrong conclusions by confusing one meaning of equality with another.

My conclusions are below, with a summary at the end. Intervals being "sometimes tagged by their i-datatype, sometimes not" is more about presentation than substance. The rest is definitely about substance. I submit this paper for discussion, and expect to submit a revised version as a motion.

1.1. **Equality of intervals.** If the same mathematical interval, say $[0, 2]$, is represented using different level 3 interval formats, are these different level 2 datums, or the same datum? I propose to resolve this in the common sense way used in IEEE754-2008 (754 for short) where it also arises. Namely "sometimes they are the same, sometimes different, depending on context". In 754 Table 3.1, the level 2 datums are depicted (apart from the two kinds of zero) as being a subset of the extended reals. However the Table heading has the phrase "for a particular format", indicating these extended-real numbers are *tagged* by their format when context requires.

Similarly, I propose P1788 regards a level 2 (bare) interval datum as being a mathematical interval tagged by its i-datatype (e.g. when specifying that arithmetic operations are defined between intervals of the same type, but not when one is represented in radix 2 and the other in radix 10, say), whose tag is sometimes ignored (e.g. when defining the mathematical result of an operation before rounding).[1]

A decorated interval, which is a pair (bare interval, decoration), see Motion 8 §1.3, is considered tagged by the tag of its interval part when context requires.

[*Note. The tag consists of the i-datatype, not of the level 3 values used to represent the datum. E.g., using an inf-sup form, let datums* $\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}$ *be represented at level 3 by "binary32* `(-0,2)`*", "binary32* `(+0,2)`*", "decimal64* `(+0,2)`*" respectively. Tagged,* $\boldsymbol{x}$ *and* $\boldsymbol{y}$ *are the same datum "binary32* $[0, 2]$*" since whether* `-0` *or* `+0` *is used is immaterial;* $\boldsymbol{z}$ *is the different datum "decimal64* $[0, 2]$*". Untagged, they are all the interval* $[0, 2]$*.*]

[*Note. Whether two datums are equal is separate from whether P1788 provides a "compareEqual" predicate to compare them. E.g., in the previous Note, untagged* $\boldsymbol{x}$ *and* $\boldsymbol{z}$ *are equal, but cannot be tested for equality.*]

1.2. **Ill-formed intervals.** The "well/ill-formed" decoration property—call it `formed`, with values `well`, `ill`, say—concentrated my mind on the equality issue. I am convinced we need it, since to meet its aim of enabling provably correct computation, P1788 must be able to flag as "ill-formed" the result of an invalid interval construction from real or text data, and propagate this information through subsequent calculations.

The recommended level 3 *mechanism* to implement `formed` is as a property in the decoration field of a decorated interval. However `formed` is *not a decoration property* in the way that, say, "domain" and "continuous" are properties, for the following reasons.

For a well-formed decorated interval one can extract its (bare) interval part, its bounds, its radius, the values of its "domain" and "continuous" properties, etc. For an ill-formed datum—whatever may be stored in the level 3 representation—none of these things have any meaning. Its bounds should be returned as NaN, for instance. Hence at level 2, ill-formed intervals cannot be told apart. To put it another way, there is not a multiplicity of ill-formed decorated interval data, *but only one*, to which we may as well give our old name "Not an Interval", NaI.

In view of the above, the getter function for the value of `formed` becomes the enquiry function `isnai()`, which returns `true` for an NaI and `false` otherwise, analogously to `isnan()` for FP values.

Date: October 5, 2010.

[1] Formally, the datum is a pair (interval, idatatypeID), where the idatatypeID is ignored in some contexts.

1.3. **Interval part of "ill-formed".** As a level 2 entity, a decorated interval is a pair (interval, decoration). Schemes for packing the decoration into the storage used by the interval part, which have been discussed recently, are just potential level 3 optimisations and must not contradict the (interval, decoration) concept.

There is to be a function, say **bare**, which extracts the bare interval part of any decorated interval. The NaI defined above is a level 2 decorated interval, by definition. Therefore

$\boldsymbol{z} = \mathbf{bare}(\mathrm{NaI})$ must be defined and be a bare interval datum.

What should $\boldsymbol{z}$ be? As said above there is "only one" ill-formed interval, so $\boldsymbol{z}$ must be *well-defined*: whatever values are inside the NaI at level 3, we must always get the same $\boldsymbol{z}$. I see two possibilities. In principle the choice could depend on the i-datatype, but I think that would complicate the standard pointlessly, so I reject it.

(a) $\boldsymbol{z}$ is a special "bare NaI" value different from all ordinary bare intervals.
(b) $\boldsymbol{z}$ is some ordinary interval.

I did (a) in my trial C++ implementation (inf-sup, based on Profil-BIAS), but group discussions have persuaded me against it. "Ask five hardware/compiler experts, and you get six opinions"—nevertheless, I think there was a consensus that P1788's design choices should be angled to making bare interval computation as fast as possible; but to make each interval operation test (in software) for a special "bare NaI" value would compromise speed.

That leaves (b). Either of the two natural choices, Empty or Entire, has its advantages. I propose Empty, since my impression is that most people prefer it because it propagates through all operations except "hull".

1.4. **Other operations on "ill-formed".** Any decorated interval valued operation returns NaI if any of its inputs is NaI. That is part of the definition, and is very efficient if the implementation follows the recommendation of representing well/ill-formedness by a decoration bit. (Later—not now please!—we may consider exceptions to this, analogous to 754's $\max(x, \mathrm{NaN}) = x$ where the NaN is regarded as "missing data" rather than as "nonsense".)

A floating-point valued operation of a decorated interval, $f(\boldsymbol{x})$, must return a well-defined FP value when $\boldsymbol{x}$ is NaI. This applies to

inf(), sup(), mid(), rad(),

and any other getter functions required by exotic i-datatypes if Motion 19 passes. I propose that $f(\mathrm{NaI})$ shall be NaN for all of these. (What about FP systems without a NaN? A good reason to exclude these from the standard.)

For 754's "positive" comparison relations (left halves of 754 Tables 5.1, 5.2), NaN compares false with everything, including itself. By analogy, I propose NaI shall compare false with everything, including itself, for the decorated interval version of each comparison specified by Motion 13.04, if that passes.

In the spirit of Motion 8, I suggest decoration property values for NaI, as returned by the appropriate getter functions, should be as pessimistic as possible: 0 for the Hayes domain tetrit, "discontinuous" for the continuity bit, etc. What is actually stored in the level 3 object is up to the implementation.

1.5. **Bare decorations?** We need enquiry functions to return the value (whether a bit, trit or tetrit) of each *supported property within a decoration*. But the motion 8 model also has *bare decoration objects*. I can't for the life of me see what advantages these bring in the construction of interval algorithms. So I propose that, for the sake of K.I.S.S., bare decorations be abolished as a level 2 concept. (I am open to being persuaded otherwise.)

1.6. **Summary.** I propose:

1. At level 2, P1788 shall support *bare interval* and *decorated interval* datums, as well as the kinds of value (bit, trit, tetrit, ...) taken by supported decoration properties. These values also count as level 2 datums.
2. Bare and decorated intervals are considered tagged by their i-datatype, or not, according to context.
3. A decoration is a list of decoration properties as in Motion 8 §1.2 but not restricted to trits.
4. Every bare interval is a (tagged) mathematical interval. A decorated interval is either the special value NaI, or a pair (bare interval, decoration). The bare interval part of NaI shall be defined as Empty.
5. Any operation with NaI as an input shall return its most pessimistic result: decorated interval output is NaI, floating-point output is NaN, boolean output is `false` (except for "negated" relations in the sense of 754§5.11, of which the enquiry isnai() is one).
6. Bare decorations shall not be supported as a separate type.