There shall be a $\mathbb{T}$-version of each of the comparison relations in Table 10.3 of 10.5.10, for each supported bare interval type $\mathbb{T}$. Its inputs are $\mathbb{T}$-intervals.

For a 754-conforming part of an implementation, mixed-type versions of these relations shall be provided, where the inputs have arbitrary types of the same radix.

These comparisons shall return, in all cases, the correct value of the comparison applied to the intervals denoted by the inputs as if in infinite precision. In particular `equal`$(\boldsymbol{x}, \boldsymbol{y})$, for those bare interval inputs $\boldsymbol{x}$ and $\boldsymbol{y}$ for which it is defined, shall return `true` if and only if $\boldsymbol{x}$ and $\boldsymbol{y}$ (ignoring their type) are the same mathematical interval, see 12.3.

Each bare interval operation in this subclause shall have a decorated version, where each input of bare interval type is replaced by an input having the corresponding decorated interval type. Following 11.7, if any input is NaI, the result is `false` (in particular `equal`(NaI, NaI) is `false`). Otherwise the result is obtained by discarding the decoration and applying the corresponding bare interval operation.

12.12.10. Interval type conversion

An implementation shall provide[1], for each supported bare interval type $\mathbb{T}$[2], the operation $\mathbb{T}$-`convertType` to convert an interval of any supported bare interval type to type $\mathbb{T}$[3] ~~and from~~ [4], and the operation $\mathbb{DT}$-`convertType` to convert an interval of any supported decorated interval type[5]type to the corresponding decorated type $\mathbb{DT}$. Conversion is done by applying the $\mathbb{T}$-hull operation, see 12.8.1. For a bare interval $\boldsymbol{x}$:

$$\mathbb{T}\text{-}\texttt{convertType}(\boldsymbol{x}) = \text{hull}_{\mathbb{T}}(\boldsymbol{x}).$$

Thus if $\mathbb{T}$ is an explicit type, see 12.8.1, the result is the unique tightest $\mathbb{T}$-interval containing $\boldsymbol{x}$.

Conversion of a decorated interval is done by converting the interval part, except that if the decoration is `com` and the conversion overflows (produces an unbounded interval) the decoration becomes `dac`. That is, [6]~~$\mathbb{T}\text{-}\texttt{convertType}(\boldsymbol{x}_{dx}) = \boldsymbol{y}_{dy}$~~ [7]$\mathbb{DT}\text{-}\texttt{convertType}(\boldsymbol{x}_{dx}) = \boldsymbol{y}_{dy}$ where

$$\boldsymbol{y} = \mathbb{T}\text{-}\texttt{convertType}(\boldsymbol{x});$$

$$dy = \begin{cases} \texttt{dac} & \text{if } dx = \texttt{com} \text{ and } \boldsymbol{y} \text{ is unbounded,} \\ dx & \text{otherwise.} \end{cases}$$

12.12.11. Operations on/with decorations

An implementation shall provide the operations of 11.5. These comprise the comparison operations [8]~~$=, \neq, >, <, \geq, \leq$~~ [9]$=, \neq, \succ, \prec, \succeq, \preceq$ for decorations; and, for each supported bare interval type and corresponding decorated type, the operations `newDec`, `intervalPart`, `decorationPart` and `setDec`.

A call `intervalPart`(NaI), whose value is undefined at Level 1, shall return Empty at Level 2, and shall signal the `IntvlPartOfNaI` exception to indicate that a valid interval has been created from the ill-formed interval.

12.12.12. Reduction operations

For each supported 754-conforming interval type, an implementation shall provide, for the parent format of that type, the four reduction operations `sum`, `dot`, `sumSquare` and `sumAbs` in 9.4 of IEEE Std 754-2008, correctly rounded.

Correctly rounded means that the returned result is defined as follows.

- If the exact result is defined as an extended-real number, return this after rounding to the relevant format according to the current rounding direction. An exact zero shall be returned as +0 in all rounding directions, except for roundTowardNegative, where −0 shall be returned.
- For `dot` and `sum`, if a NaN is encountered, or if infinities of both signs were encountered in the sum, NaN shall be returned. ("NaN encountered" includes the case $\infty \times 0$ for `dot`.)
- For `sumAbs` and `sumSquare`, if an Infinity is encountered, $+\infty$ shall be returned. Otherwise, if a NaN is encountered, NaN shall be returned.

The implementation of `textToInterval` *would need to accept this string as meaning the same as* `[Entire]`*. Such a string is not a portable literal, see 12.11.5.]*

Among the user-controllable features should be the following, where l, u are the interval bounds for inf-sup form, and m, r are the base point and radius for uncertain form, as defined in 12.11.

a) It should be possible to specify the preferred overall field width (the length of $\boldsymbol{s}$), and whether output is in inf-sup or uncertain form.

b) It should be possible to specify how Empty, Entire and NaI are output, e.g., whether lower or upper case, and whether Entire becomes `[Entire]` or `[-Inf, Inf]`.

c) For l, u and m, it should be possible to specify the field width, and the number of digits after the point or the number of significant digits. For r, which is a non-negative integer ulp-count, it should be possible to specify the field width. There should be a choice of radix, at least between decimal and hexadecimal.

d) For uncertain form, it should be possible to select the default symmetric form, or the one sided (`u` or `d`) forms. It should be possible to choose whether an exponent field is absent (and m is output to a given number of digits after the point) or present (and m is output to a given number of significant digits). [10]Despite the normalization rules in 13.4.1[11], trailing zeros may be added to m as needed. E.g., if $\boldsymbol{X} \approx [2.1995, 2.2007]$, $\boldsymbol{s}$ might be `2.200?7`, `2.20?1` or `2.2?1` depending on the user-requested tightness.

[12]It is implementation-defined how large r can be in the m `?` r form before switching to one of the m`??` forms denoting an unbounded interval. In m `?` r form, m and r should be chosen to give the tightest enclosure of $\boldsymbol{X}$ subject to m's specified number of digits after the point, or significant digits. For example, to convert $[0.9999, 1.0001]$ to this form with 2 significant digits, `9.9?2e-1`, with exact value $[0.97, 1.01]$, might be considered preferable to `1.0?1e0`, with exact value $[0.9, 1.1]$.

e) It should be possible to output the bounds of an interval without punctuation, e.g., `1.234 2.345` instead of `[1.234, 2.345]`. For instance, this might be a convenient way to write intervals to a file for use by another application.

If *cs* is absent, output should be in a general-purpose layout (analogous, e.g., to the `%g` specifier of `fprintf` in C). There should be a value of *cs* that selects this layout explicitly.

NOTE—This provides the basis for free-format output of intervals to a text stream, as might be provided by overloading the `<<` operator in C++.

If an implementation supports more general syntax of interval literals than the portable syntax defined in 12.11.5, there shall be a value of *cs* that restricts output strings to the portable syntax.

If $\mathbb{T}$ is a 754-conforming bare type, there shall be a value of *cs* that produces behavior identical with that of `intervalToExact`, below. That is, the output is an interval literal that, when read back by $\mathbb{T}$-`textToInterval`, recovers the original datum exactly.

13.4. Exact text representation

For any supported bare interval type $\mathbb{T}$, an implementation shall provide operations `intervalToExact` and `exactToInterval`. Their purpose is to provide a portable exact representation of every bare interval datum as a string.

These operations shall obey the **recovery requirement**:

> For any $\mathbb{T}$-datum $\boldsymbol{x}$, the value $\boldsymbol{s} = \mathbb{T}$-`intervalToExact`$(\boldsymbol{x})$ is a string,
> such that $\boldsymbol{y} = \mathbb{T}$-`exactToInterval`$(\boldsymbol{s})$ is defined and equals $\boldsymbol{x}$.

NOTE—From 12.3, this is datum identicality: $\boldsymbol{x}$ and $\boldsymbol{y}$ have the same Level 1 value and the same type. They might differ at Level 3, e.g., a zero bound might be stored as -0 in one and $+0$ in the other.

If $\mathbb{T}$ is a 754-conforming type, the string $\boldsymbol{s}$ shall be an interval literal which, for nonempty $\boldsymbol{x}$, is of inf-sup type, with the lower and upper bounds of $\boldsymbol{x}$ converted as described in 13.4.1. For such $\boldsymbol{s}$, the operation `exactToInterval` is functionally equivalent to `textToInterval`.