5. Level 1 description

In this clause, subclauses 5.1 to 5.5 describe the theory of mathematical intervals and interval functions that underlies this standard. The relation between expressions and the point or interval functions that they define is specified, since it is is central to the Fundamental Theorem of Interval Arithmetic. Subclauses §5.6, 5.7 list the required and recommended *arithmetic operations* (also called elementary functions) with their mathematical specifications. Subclause 5.8 describes, at a mathematical level, the system of *decorations* that is used among other things for exception handling in P1788.

5.1. Numbers. Following the terminology of 754 (e.g., 754§2.1.25), any member of the set $\mathbb{R} \cup \{-\infty, +\infty\}$ of extended reals is called a number: it is a **finite number** if it belongs to $\mathbb{R}$, else an **infinite number**.

5.2. Intervals. The set of mathematical intervals in $\mathbb{R}$, denoted $\overline{\mathbb{IR}}$, comprises[1] all closed intervals of real numbers

$$\boldsymbol{x} = [\underline{x}, \overline{x}] := \{\, x \in \mathbb{R} \mid \underline{x} \le x \le \overline{x} \,\},$$

where the bounds $\underline{x}, \overline{x}$ are extended-real numbers satisfying $-\infty \le \underline{x} \le \overline{x} \le +\infty$.
[*Notes.*

- *In particular, the empty interval* $[-\infty, -\infty] = [+\infty, +\infty] = \emptyset$ *belongs to* $\overline{\mathbb{IR}}$.
- *The above definition implies* $-\infty$ *and* $+\infty$ *can be bounds of an interval, but are never members of it.*
- *The round bracket or outward bracket notations for closed intervals in* $\mathbb{R}$ *with an infinite end point (e.g.,* $[2, +\infty)$ *or* $[2, +\infty[$ *instead of* $[2, +\infty]$*) are not used in this document but are not considered wrong.*

]

A **box** or **interval vector** is an element $\boldsymbol{x} \in \overline{\mathbb{IR}}^n$; we write $x \in \boldsymbol{x}$ if $x_i \in \boldsymbol{x}_i$ for all $i = 1, \ldots, n$. Note that a box $\boldsymbol{x}$ is empty as soon as one of its components $\boldsymbol{x}_i$ is empty.

5.3. Hull. The (interval) **hull** of an arbitrary subset $\boldsymbol{s}$ of $\mathbb{R}^n$, written hull($\boldsymbol{s}$), is the tightest member of $\overline{\mathbb{IR}}^n$ that contains $\boldsymbol{s}$. (The **tightest** set with a given property is the intersection of all sets having that property, provided the intersection itself has this property.)

5.4. Functions.

5.4.1. *Function terminology.* In this standard, operations are written as named functions; in a specific implementation they might be represented by operators (i.e., using some form of infix notation), or by families of format-specific functions, or by operators or functions whose names might differ from those in this standard.

The terms operation, function and mapping are broadly synonymous. The following summarizes the usage in this standard, with references in parentheses to precise definitions of terms.

- A *point function* (§5.4.2) is a mathematical real function of real variables. Otherwise, *function* is usually used with its general mathematical meaning.
- A (point) *arithmetic operation* (§5.4.2) is a mathematical real function for which an implementation provides versions in the implementation's *library* (§5.4.2).
- A *version* of a point function f means a function derived from f, such as an interval extension (§5.4.3) of it; usually applied to library functions.
- An *interval arithmetic operation* is an interval version of a point arithmetic operation (§5.4.3).
- An *interval non-arithmetic operation* is an interval-to-interval library function that is not an interval arithmetic operation (§5.4.3).
- A *constructor* is a function that creates an interval from non-interval data (§6.4.1).

[1] The overline is for compatibility with older notation, which uses $\mathbb{IR}$ for the set of closed and *bounded* real intervals.

5.4.2. *Point functions.* A **point function** is a (possibly partial) multivariate real function: that is, a mapping f from a subset D of $\mathbb{R}^n$ to $\mathbb{R}^m$ for some integers $n \geq 0, m > 0$. The function is called a *scalar* function if $m = 1$, otherwise a *vector* function. When not otherwise specified, scalar is assumed. The set D where f is defined is its **domain**, also written Domain f. To specify n, call f an n-variable point function, or denote values of f as

$$f(x_1, \ldots, x_n). \tag{1}$$

The **range** of f over an arbitrary subset $\boldsymbol{s}$ of $\mathbb{R}^n$ is the set

$$\text{Range}(f \,|\, \boldsymbol{s}) := \{\, f(x) \mid x \in \boldsymbol{s} \text{ and } x \in \text{Domain } f \,\}. \tag{2}$$

Thus mathematically, when evaluating a function over a set, points outside the domain are ignored (e.g., Range(sqrt $|\, [-1, 1]) = [0, 1]$).

Equivalently, for the case where f takes separate arguments $\boldsymbol{s}_1, \ldots, \boldsymbol{s}_n$, each being a subset of $\mathbb{R}$, the range is written as $\text{Range}(f \,|\, \boldsymbol{s}_1, \ldots, \boldsymbol{s}_n)$. This is an alternative notation when $\boldsymbol{s}$ is the cartesian product of the $\boldsymbol{s}_i$.

A (point) **arithmetic operation** is a function for which an implementation provides versions in a collection of user-available operations called its **library**. This includes functions normally written in operator form (e.g., $+$, $\times$) and those normally written in function form (e.g., exp, arctan). It is not specified how an implementation provides library facilities.

5.4.3. *Interval-valued functions.* A box is an interval vector $\boldsymbol{x} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n) \in \overline{\mathbb{IR}}^n$. It is usually identified with the cartesian product $\boldsymbol{x}_1 \times \ldots \times \boldsymbol{x}_n \subseteq \mathbb{R}^n$; however, the correspondence is one-to-one only when all the $\boldsymbol{x}_j$ are nonempty.

Given an n-variable point function f, an **interval extension** of f is a (total) mapping $\boldsymbol{f}$ from $\overline{\mathbb{IR}}^n$ to $\overline{\mathbb{IR}}$, such that

$$\boldsymbol{f}(\boldsymbol{x}) \supseteq \text{Range}(f \,|\, \boldsymbol{x})$$

for any box $\boldsymbol{x} \in \overline{\mathbb{IR}}^n$, regarded as a subset of $\mathbb{R}^n$. The **natural interval extension** of f is defined by

$$\boldsymbol{f}(\boldsymbol{x}) := \text{hull}(\text{Range}(f \,|\, \boldsymbol{x})).$$

Equivalently, using multiple-argument notation for f, an interval extension satisfies

$$\boldsymbol{f}(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n) \supseteq \text{Range}(f \,|\, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_n),$$

and the natural interval extension is defined by

$$\boldsymbol{f}(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n) := \text{hull}(\text{Range}(f \,|\, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_n))$$

for any intervals $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$.

In some contexts it is useful for $\boldsymbol{x}$ to be a general subset of $\mathbb{R}^n$, or the $\boldsymbol{x}_i$ to be general subsets of $\mathbb{R}$; the definition is unchanged.

The natural extension is automatically defined for all interval or set arguments. The decoration system introduced below in 5.8 gives a systematic way of diagnosing when the underlying point function has been evaluated outside its domain.

When f is a binary operator $\bullet$ written in infix notation, this gives the usual definition of its natural interval extension as

$$\boldsymbol{x} \bullet \boldsymbol{y} = \text{hull}(\{\, x \bullet y \mid x \in \boldsymbol{x},\ y \in \boldsymbol{y}, \text{ and } x \bullet y \text{ is defined} \,\}).$$

[*Example. With these definitions, the relevant natural interval extensions satisfy* $\sqrt{[-1, 4]} = [0, 2]$ *and* $\sqrt{[-2, -1]} = \emptyset$*; also* $\boldsymbol{x} \times [0, 0] = [0, 0]$ *for any nonempty* $\boldsymbol{x}$*, and* $\boldsymbol{x}/[0, 0] = \emptyset$*, for any* $\boldsymbol{x}$*.*]

When f is a vector point function, a vector interval function with the same number of inputs and outputs as f is called an interval extension of f if each of its components is an interval extension of the corresponding component of f.

An interval-valued function in the library is called an **interval arithmetic operation** if it is an interval extension of a point arithmetic operation, and an **interval non-arithmetic operation** otherwise. Examples of the latter are interval intersection and union, $(\boldsymbol{x}, \boldsymbol{y}) \mapsto \boldsymbol{x} \cap \boldsymbol{y}$ and $(\boldsymbol{x}, \boldsymbol{y}) \mapsto \text{hull}(\boldsymbol{x} \cup \boldsymbol{y})$.

5.4.4. *Constants.* A real scalar function with no arguments—a mapping $\mathbb{R}^n \to \mathbb{R}^m$ with $n = 0$ and $m = 1$—is a **real constant**. Languages may distinguish between a literal constant (e.g., the decimal value defined by the string `1.23e4`) and a named constant (e.g., π) but the difference is not relevant on Level 1 (and easily handled by outward rounding on Level 2).

From the definition, an interval extension of a constant is any zero-argument interval function that returns an interval containing c. The *natural extension* returns the interval $[c, c]$.

[*Note. This specification of constants gives a Level 1 definition of* NaN*, "Not a Number"—not as a value, but as a constant function.* $\mathbb{R}^0$ *is the zero-dimensional vector space* $\{0\}$*—it has one element, conventionally named 0. The real numbers* c *are in one-to-one correspondence with the mappings* $c() : 0 \mapsto c$*, so that* $\mathbb{R}$ *can be identified with the* total *functions* $\mathbb{R}^0 \to \mathbb{R}$*. There is one* non-total $c()$*, the function* NaN() *with empty domain and, therefore, no value. Its natural interval extension is the constant interval function whose value is the empty interval.*]

5.5. Expressions and the functions they define.

5.5.1. *Expressions.* A variable is a symbolic name. A **expression** in zero or more (independent) variables is a symbolic object defined to be either one of those variables or, recursively, of the form $\phi(g_1, \ldots, g_k)$ where ϕ is a symbolic *operation* taking k arguments ($k \geq 0$), and the g_i are expressions. Other syntax may be used, such as traditional algebraic notation and/or program pseudo-code. An **arithmetic expression** is one in which all the operations are arithmetic operations.

[*Examples.*

– *The object* f *where*

$$\mathsf{f} = (e^x + e^{-x})/(2y)$$

is an arithmetic expression in two variables x, y *that may be written in the above form as*

$$\mathrm{div}(\mathrm{plus}(\exp(x), \exp(\mathrm{uminus}(x))), \mathrm{times}(2(), y)),$$

where uminus, plus, times, div *and* exp *name the arithmetic operations "unary minus",* $+$, $\times$, $\div$ *and exponential function. The literal constant 2 is regarded as a zero-argument arithmetic operation* $2()$*, see 5.4.4.*

The expression may be broken (e.g., by a compiler) into elementary operations that may be sequenced in several ways, one of which is

$$\begin{aligned} v_1 &= \exp(x) \\ v_2 &= \mathrm{uminus}(x) \\ v_3 &= \exp(v_2) \\ v_4 &= \mathrm{plus}(v_1, v_3) \\ v_5 &= 2() \\ v_6 &= \mathrm{times}(v_5, y) \\ f &= \mathrm{div}(v_4, v_6). \end{aligned}$$

All these forms are regarded as defining the same expression f*.*

– *An expression that uses the interval intersection or union operation is not an arithmetic expression.*

]

To define functions of variables, an expression must be made into a **bound expression** by giving it a *formal argument list* with notation such as

$$f(z_1, \ldots, z_n) = \text{expression}.$$

This defines f to be the indicated expression with the formal argument list $z_1, \ldots, z_n$, which must include at least the variables that actually occur in the right-hand side. An expression without such an argument list is a **free expression**.

5.5.2. *Generic functions.* An arithmetic operation name such as $+$, or a bound arithmetic expression such as $f(x, y) = x + \sin y$, may be used to refer to different, related, functions: such operations and expressions, or the resulting functions, are called generic (or polymorphic). Whether generic function syntax can be used in program code is language-defined.

An implementation provides a number of arithmetic operation names ϕ, each representing various functions as follows.

(a) The point function of ϕ. This is unique, theoretical and generally non-computable in finite precision.
(b) Various **interval versions** of ϕ, among them in particular
  (b1) The natural interval extension of the point function. This also is unique, theoretical and generally non-computable.
  (b2) Computable interval extensions of the point function.
(c) Decorated interval versions of ϕ, see §5.8.

The implementation's library contains all computable versions of all provided arithmetic operations: see §5.6 for those that are required, and §5.7 for those recommended.

An arithmetic operation or expression may also denote a floating point function; this is not relevant to this standard.

5.5.3. *Point functions and interval versions of expressions.* A bound arithmetic expression $\mathsf{f} = \mathsf{f}(z_1, \ldots, z_n)$ represents various functions in the categories (a), (b1), (b2), (c) above.

The **point function** of (or defined by) f is a function $f : \text{Domain}\, f \subseteq \mathbb{R}^n \to \mathbb{R}$. The **natural domain** Domain f of f is the set of all $x = (x_1, \ldots, x_n) \in \mathbb{R}^n$ where its value is defined according to the following rules:

– If f is the variable z_i, $f(x)$ is the number x_i. It is defined for all $x \in \mathbb{R}^n$.
– Recursively, if $\mathsf{f} = \phi(\mathsf{g}_1, \ldots, \mathsf{g}_k)$, $f(x)$ is the value of the point function of ϕ at $u = (u_1, \ldots, u_k)$ where u_i is the value of the point function of g_i at x. It is defined for those $x \in \mathbb{R}^n$ such that each of $u_1, \ldots, u_k$ is defined and the resulting u is in ϕ's domain Domain ϕ, which is part of its mathematical definition.

  In the recursive clause of this definition, ϕ can be a constant (a function with $k = 0$ arguments), so that $f(x) = \phi()$. Then, see §5.4.4, there are two cases: (a) ϕ has a real value c; then $f(x) = c$ for all $x \in \mathbb{R}^n$; (b) ϕ is the undefined function NaN, in which case $f(x)$ is not defined for any $x \in \mathbb{R}^n$.

[*Example. The specifications*

$$f(x, y) = (e^x + e^{-x})/(2y),$$
$$f(y, x) = (e^x + e^{-x})/(2y),$$
$$f(w, x, y, z) = (e^x + e^{-x})/(2y)$$

are all valid and different (they define different functions), while

$$f(y) = (e^x + e^{-x})/(2y)$$

is invalid since the argument list does not include the variable x, which occurs in f.]

[*Example. The natural domain of $f(x, y) = 1/(\sqrt{x-1} - y)$ is the set of (x, y) in the plane that do not cause either square root of a negative number or division by zero, i.e., where $x \geq 1$ and $y \neq \sqrt{x-1}$.*]

An **interval version** of $\mathsf{f}(z_1, \ldots, z_n)$ is a (total) function $f : \overline{\mathbb{IR}}^n \to \overline{\mathbb{IR}}$. Its value at an actual argument $\boldsymbol{x} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n) \in \overline{\mathbb{IR}}^n$, a box, is defined as follows:

– If f is the variable z_i, the value is some interval containing $\boldsymbol{x}_i$.
– Recursively, if $\mathsf{f} = \phi(\mathsf{g}_1, \ldots, \mathsf{g}_k)$, the value is some interval extension of ϕ evaluated at $(\boldsymbol{u}_1, \ldots, \boldsymbol{u}_k)$ where $\boldsymbol{u}_i$ is the value of some interval version of g_i at $\boldsymbol{x}$.

The **natural** interval version of f is the unique interval version that uses the natural interval extension of each ϕ.

Moore's theorem states:

Theorem 5.1 (Fundamental Theorem of Interval Arithmetic, FTIA).
(i) Every interval version of a arithmetic expression $\mathsf{f}(z_1, \ldots, z_n)$ is an interval extension of the point function defined by the expression.
(ii) If every variable occurs at most once in f *and, for some $\boldsymbol{x} \in \overline{\mathbb{IR}}^n$, all elementary operations occurring in* f *evaluate to their range then $f(\boldsymbol{x}) = \text{Range}(f \,|\, \boldsymbol{x})$.*

[*Note. To part (ii). Using exact arithmetic, that is, the natural interval version of* f*, it is known that an elementary operation evaluates to its range if it is continuous on its input box. This can be checked by the decoration system (§5.8), so the conditions of (ii) are computable. The result is that in finite precision it is often verifiable at run time that $f(\boldsymbol{x})$ equals* Range$(f \,|\, \boldsymbol{x})$ *up to roundoff error.*]

5.8. The decoration system.

5.8.1. *The purpose of decorated intervals.* Decorations are properties of a function for which an interval enclosure of its range over a box is being computed. There are two main objectives of interval calculations:

(1) obtaining correct range enclosures for a real-valued function f of real variables;
(2) verifying the assumptions of existence, uniqueness, or nonexistence theorems.

While traditional interval analysis targets the first aim, decorations target the second aim.

A decoration primarily describes a property, not of the interval it is attached to, but of the function defined by a section of code that produced that interval.

The decoration system is designed in a way that naive users of interval arithmetic do not notice anything about decorations, unless they inquire explicitly about their values. They are only required to

– call the domain operation, see §5.8.5, for the input vector of any function evaluation used to invoke an existence theorem,
– explicitly convert all floating-point constants (but not integer constants) to intervals,

and have the full rigor of interval calculations available. A smart compiler may even relieve users from these tasks. Expert users can inspect, set and modify decorations to improve code efficiency, but are responsible for checking that computations done in this way remain rigorously valid.

Decorations are based on the desire that, from an interval evaluation of a real function f at a box $\boldsymbol{x}$, one should get not only a range enclosure $f(\boldsymbol{x})$ but also a guarantee that the pair $(f, \boldsymbol{x})$ has certain important properties, such as $f(x)$ being defined for all $x \in \boldsymbol{x}$, f restricted to $\boldsymbol{x}$ being continuous, etc. This goal is achieved, in parts of a program that require it, by adding to each interval extra information in the form of a **decoration**, whose semantics is summarized as follows:

> Each intermediate step of the original computation depends on some or all of the inputs, so it can be viewed as an intermediate function of these inputs. The result interval obtained on each intermediate step is an enclosure for the range of the corresponding intermediate function. The decoration attached to this intermediate interval reflects the available knowledge about whether this intermediate function is guaranteed to be everywhere defined, continuous, bounded, etc., on the given inputs.

This applies to *arithmetic* interval operations. *Nonarithmetic* operations, such as hull and intersection of two intervals, occur in too varied contexts to allow a uniform decoration-handling scheme. Hence they ignore decorations, and give undecorated output. Programmers are responsible for the appropriate propagation of decorations by these operations.

The P1788 decoration model, in contrast with 754's, has no global flags. A general aim, as in 754's use of NaN and flags, is not to interrupt the flow of computation: rather, to collate information during evaluation for inspection afterwards. This enables a fully local handling of exceptional conditions in interval calculations—important in a concurrent computing environment.

The system is outlined here at a mathematical level, with the finite-precision aspects in §6.5. §5.8.2 to §5.8.5 give the basic concepts. §5.8.6 gives examples. §5.8.7, 5.8.8 discuss the decoration of library arithmetic operations, and of user-defined arithmetic operations. §5.8.9 is about type conversion in mixed operations. §5.8.10 defines a restricted arithmetic that is sufficient for some important uses of decorations and allows especially efficient implementation on current computer architectures. Though its rationale is an issue for Levels 2 onward, its description is appropriate to put in Level 1. Finally, correctness proofs are in Annex C.

5.8.2. *Definitions.* The set $\mathbb{D}$ of **decorations** has seven elements:

⚠ Note from AN+JDP. Please don't criticise on the grounds of the number of decorations. We don't necessarily favour this but want to ensure the theory works for the 'most complicated' case. It works equally well for simpler schemes where some of these categories are merged.

Value	Short description
`ein`	empty input
`bnd`	bounded
`dac`	defined & continuous
`def`	defined
`con`	containing
`emp`	empty
`ill`	ill-formed

Formally, each $d \in \mathbb{D}$ is the set of pairs $(f, \boldsymbol{x})$ consisting of a real-valued function f with domain Domain $f \subseteq \mathbb{R}^n$ for some n and a box $\boldsymbol{x} \in \overline{\mathbb{IR}}^n$ for which the property $p_d(f, \boldsymbol{x})$ is true, where

$$\begin{aligned}
&p_{\texttt{ein}}(f, \boldsymbol{x}) : \boldsymbol{x} \text{ is empty (i.e., has some empty component);}\\
&p_{\texttt{bnd}}(f, \boldsymbol{x}) : \boldsymbol{x} \text{ is a subset of Domain } f\text{, and the restriction of } f \text{ to } \boldsymbol{x} \text{ is continuous and bounded;}\\
&p_{\texttt{dac}}(f, \boldsymbol{x}) : \boldsymbol{x} \text{ is a subset of Domain } f\text{, and the restriction of } f \text{ to } \boldsymbol{x} \text{ is continuous;}\\
&p_{\texttt{def}}(f, \boldsymbol{x}) : \boldsymbol{x} \text{ is a subset of Domain } f\text{;}\\
&p_{\texttt{con}}(f, \boldsymbol{x}) : \text{always true;}\\
&p_{\texttt{emp}}(f, \boldsymbol{x}) : \boldsymbol{x} \text{ is nonempty and disjoint from Domain } f\text{;}\\
&p_{\texttt{ill}}(f, \boldsymbol{x}) : \boldsymbol{x} \text{ is nonempty and Domain } f \text{ is empty.}
\end{aligned} \tag{8}$$

A **decorated interval** is a pair, written interchangeably as $(\boldsymbol{f}, d)$ or $\boldsymbol{f}_d$, where $\boldsymbol{f} \in \overline{\mathbb{IR}}$ is a real interval and $d \in \mathbb{D}$ is a decoration, such that the following invariance property holds:

$$d \text{ is in } \{\,\texttt{bnd}, \texttt{dac}, \texttt{def}, \texttt{con}\,\} \text{ if } \boldsymbol{f} \text{ is nonempty, and in } \{\,\texttt{emp}, \texttt{ill}, \texttt{ein}\,\} \text{ if } \boldsymbol{f} \text{ is empty.} \tag{9}$$

(The other combinations would not make sense.) An interval or decoration may be referred to as a **bare** interval or decoration, to emphasize that it is not a decorated interval.

$\boldsymbol{x}_d$ may also be a decorated box $\big((\boldsymbol{x}_1, d_1), \ldots, (\boldsymbol{x}_n, d_n)\big)$, where $\boldsymbol{x}$ and d are the vectors of interval parts $\boldsymbol{x}_i$ and decoration parts d_i respectively. The set of decorated intervals is denoted by $\overline{\mathbb{DIR}}$, and the set of decorated boxes of length n is denoted by $\overline{\mathbb{DIR}}^n$. In program format, e.g. pseudocode, the interval and decoration parts of a decorated interval or box named `x` may be written `x.int` and `x.dec` respectively.

The decoration system provides an extension of Moore's Theorem 5.1, as follows. The precise definition of a decorated interval version of an expression is given in §5.8.4.

Theorem 5.2 (Fundamental Theorem of Decorated Interval Arithmetic, FTDIA). *Let* $\boldsymbol{f}_d = f(\boldsymbol{x})$ *be the result of evaluating an arithmetic expression* $\mathsf{f}(z_1, \ldots, z_n)$ *over a bare box* $\boldsymbol{x} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n) \in \overline{\mathbb{IR}}^n$ *using any decorated interval version of* f. *Then*

$$p_d(f, \boldsymbol{x}) \text{ holds.} \tag{10}$$

This is in addition to the enclosure $\boldsymbol{f} \supseteq \mathrm{Range}(f \,|\, \boldsymbol{x})$ *given by Moore's Theorem 5.1.*

The rigor necessary to make interval arithmetic a reliable tool for verified computing warrants a detailed proof, given in Annex C.

If all arithmetic operations are implemented with inclusion isotone interval extensions then the decorations are also inclusion isotone. ⚠ A precise statement will be provided and proved at a later stage.

5.8.3. *Basic properties.* By design, the decorations are partially ordered by set containment giving the **containment order**, which is immediate from the definitions (8):

$$\texttt{ein} \subseteq \texttt{bnd} \subseteq \texttt{dac} \subseteq \texttt{def} \subseteq \texttt{con} \supseteq \texttt{emp} \supseteq \texttt{ill}. \tag{11}$$

E.g., $\texttt{dac} \subseteq \texttt{def}$ is equivalent to $p_{\texttt{dac}}(f, \boldsymbol{x}) \implies p_{\texttt{def}}(f, \boldsymbol{x})$, which is clearly true. Note the reversal of the containment signs after `con`. Also by design, any $d, e \in \mathbb{D}$ satisfy the **exclusivity** rule:

$$\text{If neither } d \subseteq e \text{ nor } e \subseteq d \text{ then } p_d(f, \boldsymbol{x}) \text{ and } p_e(f, \boldsymbol{x}) \text{ cannot both hold, i.e. } d \cap e = \emptyset. \tag{12}$$

[*Note.* `con` *is the loosest decoration, where nothing is claimed;* `ein` *and* `ill` *claim the most and are tightest. The containment order reflects the fact that computed information is often suboptimal owing to the "dependence problem": e.g., one may compute* `con` *when in fact* `dac` *is true.*]

Correct propagation of decoration information requires the (total) **propagation order**:

$$\texttt{ein} > \texttt{bnd} > \texttt{dac} > \texttt{def} > \texttt{con} > \texttt{emp} > \texttt{ill}. \tag{13}$$

This order is implied whenever taking the *minimum* or *maximum* of decorations. Excepting `ein`—which may be considered anomalous—this order reflects increasing "quality" of the decoration: `ill` is the "worst", and the decorations get "better" as one proceeds up the list.

The two orderings are closely related: for any two decorations c, d, we have $c \subseteq d$ iff $c \le d \le \texttt{con}$ or $c \ge d \ge \texttt{con}$. As shown in Annex C, this property is essential for a correct flow of the information contained in the decorations.

For a pair $(f, \boldsymbol{x})$ as in §5.8.2, the **decoration of f over $\boldsymbol{x}$**, written $\mathrm{dec}(f, \boldsymbol{x})$, is defined by

$$\mathrm{dec}(f, \boldsymbol{x}) = d \text{ iff } p_d(f, \boldsymbol{x}) \text{ holds and } p_e(f, \boldsymbol{x}) \text{ fails for all } e \text{ with } d \ne e \subseteq d, \tag{14}$$

that is, $\mathrm{dec}(f, \boldsymbol{x})$ is the unique tightest decoration d for which $p_d(f, \boldsymbol{x})$ is true; its role for decorations is similar to that of the (exact) range for enclosures. It is well-defined because of property (12). More explicitly,

$$\mathrm{dec}(f, \boldsymbol{x}) = \begin{cases} \texttt{ein} & \text{if } p_{\texttt{ein}}(f, \boldsymbol{x}) \text{ holds;} \\ \texttt{bnd} & \text{if } p_{\texttt{ein}}(f, \boldsymbol{x}) \text{ fails and } p_{\texttt{bnd}}(f, \boldsymbol{x}) \text{ holds;} \\ \texttt{dac} & \text{if } p_{\texttt{bnd}}(f, \boldsymbol{x}) \text{ fails and } p_{\texttt{dac}}(f, \boldsymbol{x}) \text{ holds;} \\ \texttt{def} & \text{if } p_{\texttt{dac}}(f, \boldsymbol{x}) \text{ fails and } p_{\texttt{def}}(f, \boldsymbol{x}) \text{ holds;} \\ \texttt{ill} & \text{if } p_{\texttt{ill}}(f, \boldsymbol{x}) \text{ holds;} \\ \texttt{emp} & \text{if } p_{\texttt{ill}}(f, \boldsymbol{x}) \text{ fails and } p_{\texttt{emp}}(f, \boldsymbol{x}) \text{ holds;} \\ \texttt{con} & \text{otherwise.} \end{cases} \tag{15}$$

[*Note. Though* `con` *is trivial in itself, to have* $\mathrm{dec}(f, \boldsymbol{x}) = \texttt{con}$ *is not trivial: it asserts* $p_{\texttt{emp}}$ *and* $p_{\texttt{def}}$ *are both false, which implies in particular that* $\boldsymbol{x}$ *contains infinitely many points.*]

5.8.4. *Decorated interval extensions and versions.*

Similarly to the bare interval case in §5.5.3, a *decorated interval version* f of a user-defined arithmetic expression is constructed from library functions ϕ that are *decorated interval extensions* of arithmetic operations. Both f and ϕ output a decorated interval; but f in effect acts on bare intervals, while ϕ acts on decorated intervals, combining their decorations with its "local" decoration, to propagate a decoration to the output.

Namely, a **decorated interval extension** of a real k ary point function ϕ is a (total) function, also called ϕ, from decorated boxes $\boldsymbol{g}_c = ((\boldsymbol{g}_1, c_1), \ldots, (\boldsymbol{g}_k, c_k)) \in \overline{\mathbb{DIR}}^k$ to decorated intervals, such that $\boldsymbol{\phi}_d := \phi(\boldsymbol{g}_c)$ satisfies

$$\boldsymbol{\phi} \supseteq \mathrm{Range}(\phi \,|\, \boldsymbol{g}) \qquad \text{(enclosure)} \tag{16}$$

together with

$$d = \min\{e, c_1, \ldots, c_k\} \qquad \text{(propagation rule)} \tag{17}$$

for some e such that

$$p_e(\phi, \boldsymbol{g}) \text{ holds} \qquad \text{(local decoration).} \tag{18}$$

The **natural** decorated interval extension is the unique one for which $\boldsymbol{\phi} = \mathrm{hull}(\mathrm{Range}(\phi \,|\, \boldsymbol{g}))$—i.e., the interval part is the natural bare interval extension of ϕ—and e equals $\mathrm{dec}(\phi, \boldsymbol{g})$, the decoration of ϕ over $\boldsymbol{g}$.

A **decorated interval version** of an expression $\mathsf{f}(z_1, \ldots, z_n)$ is a (total) function f from bare boxes $\boldsymbol{x} \in \overline{\mathbb{IR}}^n$ to decorated intervals whose value $\boldsymbol{f}_d$ at an actual argument $\boldsymbol{x} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)$ is defined as follows.

(Eval1) If $\boldsymbol{x}$ is empty (i.e., has an empty component) then $\boldsymbol{f} = \emptyset$ and $d = \texttt{ein}$.

(Eval2) Otherwise, if f is the variable z_i, the value is

$$\boldsymbol{f}_d = (\boldsymbol{x}_i, \mathrm{dec}(\mathrm{id}, \boldsymbol{x}_i))$$

where id is the real identity map $\mathrm{id}(x) = x$.
That is, d equals `dac` if $\boldsymbol{x}_i$ is unbounded, and `bnd` if $\boldsymbol{x}_i$ is bounded.

(Eval3) Otherwise, recursively, if $\mathsf{f} = \phi(\mathsf{g}_1, \ldots, \mathsf{g}_k)$, the value is some decorated interval extension of ϕ evaluated at $\boldsymbol{g}_c = ((\boldsymbol{g}_1, c_1), \ldots, (\boldsymbol{g}_k, c_k))$ where $(\boldsymbol{g}_i, c_i)$ is the value of some decorated interval version of g_i at $\boldsymbol{x}$.

Decorated interval evaluation means evaluating a decorated interval version of an expression in this way. The **natural** decorated interval version is the unique one that uses the natural decorated interval extension of each ϕ in (Eval3).

[*Notes.*

– *The specification in the "empty" case (Eval1) is needed in order to cover correctly the case when some input variable is not present in the expression—e.g., where $f(x) = f(x_1, x_2, x_3) = x_1 x_3$, and the actual argument $\boldsymbol{x}_2$ is empty.*
– *In the recursion base case (Eval2) it suffices to have $\boldsymbol{f}_d$ such that $\boldsymbol{f} \supseteq \boldsymbol{x}_i$ and $p_d(\mathsf{id}, \boldsymbol{x}_i)$ holds. In some finite-precision situations such looser values may be necessary.*
– *Properties (17, 18) ensure a correct flow of decoration information and justify the recursive step (Eval3). Namely—as shown in Annex C—if $\boldsymbol{x}$ is nonempty, and $\mathsf{f} = \phi(\mathsf{g}_1, \ldots, \mathsf{g}_k)$, and $p_{c_i}(g_i, \boldsymbol{x})$ holds for $i = 1, \ldots, k$, then $p_d(f, \boldsymbol{x})$ holds.*

]

5.8.5. *Using decorated interval functions.*

The **domain** () function converts a bare box $\boldsymbol{x} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)$ into a decorated box $\mathbf{domain}(\boldsymbol{x}) := \boldsymbol{x}'_c = ((\boldsymbol{x}'_1, c_1), \ldots, (\boldsymbol{x}'_n, c_n))$ that is suitable initial data for decorated interval evaluation, as follows.

If *some* component of $\boldsymbol{x}$ is empty,	$(\boldsymbol{x}'_i, c_i) = (\emptyset, \mathtt{emp})$ for all $i = 1, \ldots, n$,
else	$(\boldsymbol{x}'_i, c_i) = (\boldsymbol{x}_i, \mathrm{dec}(\mathrm{id}, \boldsymbol{x}_i))$ for all $i = 1, \ldots, n$.

The second part of this definition performs step (Eval2) of decorated interval evaluation; the first part is equivalent to (Eval1) in the sense that it ensures any empty component of $\boldsymbol{x}$ produces an empty result. Thus decorated interval evaluation of a function f defined by an expression may be done by these steps:

1. Replace $\boldsymbol{x}$ by $\mathbf{domain}(\boldsymbol{x})$;
2. Perform step (Eval3) by applying (decorated interval versions of) arithmetic operations in a suitable sequence, giving a decorated interval result $\boldsymbol{f}_d$.

Then by Theorem 5.2, $\boldsymbol{f}$ contains $\mathrm{Range}(f \mid \boldsymbol{x})$ and $p_d(f, \boldsymbol{x})$ holds. From the definition (14), the latter is equivalent to: d contains the true decoration $D := \mathrm{dec}(f, \boldsymbol{x})$. These are the possible cases:

(i) $d = \mathtt{ein}$ iff $D = \mathtt{ein}$ iff the input box is empty.
(ii) Otherwise the input box is guaranteed to be nonempty, and one of the following holds:
  – If $d = \mathtt{bnd}$ then D must also be `bnd`, so f is guaranteed to be everywhere defined, continuous and bounded on the nonempty box $\boldsymbol{x}$.
  – If $d = \mathtt{dac}$ then D must be `dac` or `bnd`, so f is guaranteed to be everywhere defined and continuous on the nonempty box $\boldsymbol{x}$. It might be bounded there as well, but this is not known.
  – If $d = \mathtt{def}$, then D must be `def` or `dac` or `bnd`, so f is guaranteed to be everywhere defined on the nonempty box $\boldsymbol{x}$. It might be continuous and/or bounded there as well, but this is not known.
  – If $d = \mathtt{ill}$, then D must be `ill`: Domain f is empty, that is, f is nowhere defined, although the box $\boldsymbol{x}$ is nonempty. This occurs if and only if some intermediate constant operation was ill-formed. This has a special significance of *Not an Interval* at the computational level: see §C.2. In particular, ill-formedness is sticky in expressions evaluated on a box generated by the domain function.
  – If $d = \mathtt{emp}$, then D must be `emp` or `ill`: Domain f is guaranteed to be disjoint from the nonempty box $\boldsymbol{x}$.
  – If $d = \mathtt{con}$, then D might be any decoration $\neq \mathtt{ein}$; no further conclusion can be drawn.

The **domain** () function also acts on decorated boxes $\boldsymbol{x}_b$. In effect this discards b and acts on $\boldsymbol{x}$ only, *except* that if some component of $\boldsymbol{x}_b$ is ill-formed it makes every component of the result ill-formed. Namely $\mathbf{domain}(\boldsymbol{x}_b) := \boldsymbol{x}'_c$ where

If *some* component of b equals `ill`,	$(\boldsymbol{x}'_i, c_i) = (\emptyset, \mathtt{ill})$ for all $i = 1, \ldots, n$,
else	$\boldsymbol{x}'_c = \mathbf{domain}(\boldsymbol{x})$.

This feature reduces the risk of data about ill-formedness being lost when initializing input to decorated interval evaluation.

5.8.6. *Examples.*

Consider decorated interval evaluation of $f(x,y) = \sqrt{x(y-x)-1}$ with various input intervals $\boldsymbol{x}$, $\boldsymbol{y}$. The natural domain $\mathrm{Domain}(f)$ is easily seen to be the union of the regions $x > 0$, $y \geq x+1/x$ and $x < 0$, $y \leq x + 1/x$. We use exact arithmetic, i.e., the *natural* decorated interval extension of each arithmetic operation. Finite precision would produce valid but usually slightly different results.

For manageable notation, let an interval named $\boldsymbol{x}$ be given a decoration dx, and so on.

[*Examples.*

(i) Let $\boldsymbol{x} = [1,2]$, $\boldsymbol{y} = [3,4]$, defining a box $(\boldsymbol{x},\boldsymbol{y})$ contained in Domain f. *Applying the* **domain** *function gives initial decorated intervals $\boldsymbol{x}_{dx} = [1,2]_{\mathtt{bnd}}$, $\boldsymbol{y}_{dy} = [3,4]_{\mathtt{bnd}}$. The first operation is*

$$\boldsymbol{u}_{du} = \boldsymbol{y}_{dy} - \boldsymbol{x}_{dx} \qquad = [1,3]_{\mathtt{bnd}}.$$

Namely, subtraction is defined and continuous on all of $\mathbb{R}^2$, and bounded on bounded rectangles (call this property "nice" for short), so the bare result decoration is $du' = \mathrm{dec}(-,(\boldsymbol{y},\boldsymbol{x})) = \mathtt{bnd}$, whence by (D1) the (best possible) decoration on $\boldsymbol{u}$ is $du = \min\{du', dy, dx\} = \min\{\mathtt{bnd},\mathtt{bnd},\mathtt{bnd}\} = \mathtt{bnd}$. Multiplication is also "nice", so the second operation similarly gives

$$\boldsymbol{v}_{dv} = \boldsymbol{x}_{dx} \times \boldsymbol{u}_{du} \qquad = [1,6]_{\mathtt{bnd}}.$$

The constant 1, following §5.4.4, becomes a decorated interval function returning the constant value $[1,1]_{\mathtt{bnd}}$. The next operation is again "nice", and gives

$$\boldsymbol{w}_{dw} = \boldsymbol{v}_{dv} - 1 \qquad = [0,5]_{\mathtt{bnd}}$$

Finally $\sqrt{\cdot}$ is defined, continuous and bounded on $\boldsymbol{w} = [0,5]$, so, arguing similarly, one has the final result

$$\boldsymbol{f}_{df} = \sqrt{\boldsymbol{w}_{dw}} \qquad = [0,\sqrt{5}]_{\mathtt{bnd}}.$$

By Theorem 5.2 this provides a rigorous proof that for the box $\boldsymbol{z} = (\boldsymbol{x},\boldsymbol{y}) = ([1,2],[3,4])$,

$$[0,\sqrt{5}] \supseteq \mathrm{Range}(f \mid \boldsymbol{z}),$$

$$p_{\mathtt{bnd}}(f,\boldsymbol{z}) \textit{ holds.}$$

That is, f is defined, continuous and bounded on $1 \leq x \leq 2,\ 3 \leq y \leq 4$, and its range over this box is a subset of $[0,\sqrt{5}]$.

(ii) Let $\boldsymbol{x} = [1,2]$ as before, but $\boldsymbol{y} = [\frac{5}{2},4]$. The box $\boldsymbol{z}$ is still contained in Domain f *so the true value of $\mathrm{dec}(f,\boldsymbol{z})$ is still* bnd. *However the evaluation fails to detect this because of interval widening due to the dependence problem of interval arithmetic. Namely after $\boldsymbol{u}_{du} = [\frac{5}{2},3]_{\mathtt{bnd}}$, $\boldsymbol{v}_{dv} = [\frac{5}{2},6]_{\mathtt{bnd}}$, $\boldsymbol{w}_{dw} = [-\frac{1}{2},5]_{\mathtt{bnd}}$, the final result has interval part $\boldsymbol{f} = \sqrt{[-\frac{1}{2},5]} = [0,\sqrt{5}]$ as before, but $\sqrt{\cdot}$ is not everywhere defined on $\boldsymbol{w}$, so that $dw' = \mathrm{dec}(\sqrt{\cdot},\boldsymbol{w}) = \mathrm{dec}(\sqrt{\cdot},[-\frac{1}{2},5]) = \mathtt{con}$ giving $\mathrm{dec}(\sqrt{\cdot},\boldsymbol{w}_{dw}) = \min\{dw',dv\} = \mathtt{con}$, so finally $\boldsymbol{f}_{df} = [0,\sqrt{5}]_{\mathtt{con}}$. This is a valid enclosure of the decorated range $[0,\sqrt{5}]_{\mathtt{bnd}}$, but we have been unable to verify the* bnd *property.*

(iii) If $\boldsymbol{x} = [1,2]$, $\boldsymbol{y} = [1,1]$, the box $\boldsymbol{z}$ is now wholly outside Domain f, *and evaluation detects this, giving the exact result $\boldsymbol{f}_{df} = \emptyset_{\mathtt{emp}}$. However, if $\boldsymbol{x} = [1,2]$, $\boldsymbol{y} = [1,\frac{3}{2}]$, the box is still wholly outside* Domain f, *but owing to widening, evaluation fails to detect this, giving $\boldsymbol{f}_{df} = [0,0]_{\mathtt{con}}$—a valid enclosure but of little use.*

]

5.8.7. *Decoration of library functions.*

Subclause C.1 gives tables of $\mathrm{dec}(\phi,\boldsymbol{x})$ for all required and recommended arithmetic operations ϕ and an arbitrary input box $\boldsymbol{x}$. They are generally straightforward to compute exactly in finite precision, except for a few functions like $\tan x$, where it may be expensive to determine for certain whether a singularity lies inside or outside an interval with a very large endpoint. In such cases, an implementation may return a less precise decoration.

The function $\mathbf{case}(b,g,h)$ deserves mention.

[*** This seems to need updating for the 7-decoration scheme. And I feel there are some errors. ***]

Subclause 5.6.2 defines it and its behavior on bare intervals. The decorated interval version

case$(\boldsymbol{b}_{db}, \boldsymbol{g}_{dg}, \boldsymbol{h}_{dh})$ is defined according to the standard worst case semantics, and returns $\boldsymbol{f}_{df}$ where $df = \min\{df', db, dg, dh\}$ and

$$df' = \begin{cases} \texttt{emp} & \text{if } \boldsymbol{b} \text{ contains neither 0 nor 1} \\ & \text{(in normal use this only occurs when } \boldsymbol{b} \text{ is empty)} \\ \texttt{bnd} & \text{elseif } \boldsymbol{b} \text{ contains 0 and } \boldsymbol{h} \text{ is bounded} \\ & \text{or } \boldsymbol{b} \text{ contains 1 and } \boldsymbol{g} \text{ is bounded} \\ \texttt{dac} & \text{elseif } \boldsymbol{b} \text{ contains 0 and } \boldsymbol{h} \text{ is unbounded} \\ & \text{or } \boldsymbol{b} \text{ contains 1 and } \boldsymbol{g} \text{ is unbounded} \\ \texttt{con} & \text{otherwise.} \end{cases}$$

Note that this correctly handles empty input: In this case, one of db, dg, dh is `ein`, resulting in $df =$ `ein`. However, in most cases, functions defined using **case** give very suboptimal enclosures, and it is preferable to use methods illustrated in §5.8.8.

5.8.8. *User-supplied functions.* A user program may define a decorated interval version of a point function, to be used within expressions as if it were a library function. This does not invalidate Theorem 5.2, subject to the one requirement that the decorated interval version be a decorated interval extension of the point function.

[*Examples.*

(1) In some applications, an interval extension of the function defined by

$$\psi(x) = x + 1/x$$

is required. The expression as it stands can give poor enclosures: e.g., with $\boldsymbol{x} = [\frac{1}{2}, 2]$*, one obtains*

$$\psi(\boldsymbol{x}) = [\tfrac{1}{2}, 2] + 1/[\tfrac{1}{2}, 2] = [\tfrac{1}{2}, 2] + [\tfrac{1}{2}, 2] = [1, 4],$$

which is much wider than $\mathsf{Range}(\psi \,|\, \boldsymbol{x}) = [2, 2\frac{1}{2}]$*.*

Thus it is useful to code a tight enclosure by special methods, e.g. monotonicity arguments, and provide this as a new library function. Suppose this has been done. To implement a decorated interval extension for the above function just entails adding code to compute an enclosure of the decoration $d = \mathsf{dec}(\psi, \boldsymbol{x}_c)$ *over an input decorated interval* $\boldsymbol{x}_c$*, along the lines of the following:*

// compute decoration c_0 *over bare interval* $\boldsymbol{x}$*:*

1. *if* $\boldsymbol{x}$ *is empty then* $c_0 =$ `ein`*;*
2. *if* $\boldsymbol{x}$ *is the singleton* $[0, 0]$ *then* $c_0 =$ `emp`*;*
3. *elseif* $0 \in \boldsymbol{x}$ *then* $c_0 =$ `con`*;*
4. *elseif* $\boldsymbol{x}$ *is unbounded then* $c_0 =$ `dac`*;*
5. *else* $c_0 =$ `bnd`*;*

// combine with input decoration by equation (D1):

6. $d = \min\{c_0, c\}$.

(2) The next example shows how an expert may manipulate decorations explicitly to give a function, defined piecewise by different formulas in different regions of its domain, the best possible decoration. Suppose that

$$f(x) = \begin{cases} f_1(x) := \sqrt{x^2 - 4} & \text{if } |x| > 2, \\ f_2(x) := -\sqrt{4 - x^2} & \text{otherwise.} \end{cases}$$

The standard gives no way to determine if the pieces join continuously at region boundaries: e.g., if f *is implemented by the* **case** *function, the continuity information is lost when evaluating it on, say,* $\boldsymbol{x} = [1, 3]$*, where both branches contribute for different values of* $x \in \boldsymbol{x}$*. However, a user-defined decorated interval operation along the following lines provides this function with best possible decorations.*

function $\boldsymbol{y}_d = f(\boldsymbol{x}_c)$
$\boldsymbol{u} = f_1(\boldsymbol{x} \cap [-\infty, -2])$
$\boldsymbol{v} = f_2(\boldsymbol{x} \cap [-2, 2])$
$\boldsymbol{w} = f_1(\boldsymbol{x} \cap [2, +\infty])$
$\boldsymbol{y} = \mathit{union}(\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w})$
$d = c$

Here "union" forms the hull of the set-union of its arguments. The user's knowledge that f is continuous is expressed by the statement $d = c$, propagating the input decoration unchanged.

]

5.8.9. *Type conversion in mixed operations.*

⚠ This needs checking by language and compiler experts and should be the subject of a separate motion. Also, does it all belong in the decorations section? Indeed should it be in Level 1 at all?

Decorated interval arithmetic is designed for maximal safety, while being simple to handle by inexperienced users. Safety requirements can be enforced only by restrictions on the kinds of type conversions permitted.

Operations between integers and decorated intervals are well-defined and hence permitted, with integers treated as constant functions.

Operations between floats and decorated intervals are error-prone and hence forbidden, since, e.g., $(2/3) * \boldsymbol{x}$ in program text would generate uncovered roundoff, and $0.2 * \boldsymbol{x}$ would generate uncovered conversion errors. This ensures that the user must call explicitly a conversion function **iconst** that performs the outward rounding, see §6.6, to convey the precise semantics of such mixed expressions. This avoids a loss of containment because of rounding errors or conversion errors.

In particular, there is no implicit type casting for real times decorated interval. Therefore, $2/3 * \boldsymbol{x}$ with reals or integers 2 and 3 and a decorated interval $\boldsymbol{x}$ results in a type error when trying to evaluate the multiplication.

However, implicit type casting for text constants times interval is harmless, as text constants have no arithmetic operations defined on them, hence they can be unambiguously type cast to decorated intervals when occurring in an interval expression if the implementation language allows that. Therefore, $2/3 * \boldsymbol{x}$ is allowed if the compiler translates 2 and 3 into constant functions.

Mixed operations between bare intervals and decorated intervals are also forbidden, to avoid loss of rigor through non-arithmetic operations; again, explicit conversion using the function domain must be used. However, explicit, constant bare intervals in program code may be treated by the compiler as constant functions with uncertain value when the bare interval is nonempty, and as the ill-formed constant when the bare interval is empty or ill-formed.

5.8.10. *Bare object arithmetic with a threshold.* Bare intervals and bare decorations are called **bare objects**. Bare object arithmetic, described here, lets experienced users obtain more efficient execution (see §7.6) in applications where the use of decorations is limited to the following situation.

Namely, frequently the use that is made of a function evaluation $\boldsymbol{y}_d = f(\boldsymbol{x}_c)$ at a decorated interval $\boldsymbol{x}_c$ depends on a check of the resulting decoration d against an application-dependent **exception threshold** T, which is one of $\{\texttt{con}, \texttt{def}, \texttt{dac}, \texttt{bnd}\}$.

- $d \geq T$ represents normal computation. The decoration is not used, but one exploits the range enclosure given by the interval part.
- $d < T$ declares an exception to have occurred. The interval part is not used, but one exploits the information given by the decoration.

In this case, one can implement the decoration scheme without storing explicit decorations. The behavior depends on the exception threshold, which the programmer or a compiler can choose appropriately to the final use made of the interval evaluation.

The storage of bare object data, whether interval or decoration, can be done within two floating-point numbers. Thus in the kind of applications mentioned above, bare objects usually give equivalent results to full decorated intervals at less cost in storage and communication.

The arithmetic operations, and intersection and union, are defined on bare objects, extending the standard arithmetic on bare intervals. The results are determined according to the following **worst case semantics** rules for **promoting** bare objects to decorated intervals. These follow necessarily if the fundamental theorem is to cover computations involving bare and/or decorated arguments. They ensure that an enclosing decoration compatible with the input is returned, so that in an extended calculation, the claim about the result of a function evaluation using bare object arithmetic is never tighter than the result of the same evaluation using full decorated interval arithmetic. Hence, Theorem 5.2 continues to hold.

– Each nonempty bare interval is treated as decorated with decoration T, and each empty bare interval as decorated with decoration `emp`. Operations on bare intervals are performed in this mode as if they were decorated in the way described, resulting in a decorated interval $\boldsymbol{z}_d$ that

TABLE 5. Bare object operations for $+, -, \times, \div$ and $\sqrt{\cdot}$ with threshold $T \in \{\texttt{con}, \texttt{def}, \texttt{dac}, \texttt{bnd}\}$.

Here c, d are bare decorations $< T$, and $\boldsymbol{x}, \boldsymbol{y}$ are bare intervals. Independently of T, if any input is `ill` the result is `ill`, else if any input is `emp` or the empty interval the result is `emp`. The tables below give the remaining cases where

$$\texttt{con} \le c < T,\ \texttt{con} \le d < T, \text{ and } \boldsymbol{x}, \boldsymbol{y} \text{ are nonempty.} \tag{19}$$

Binary operations, where $\boldsymbol{x}$ or c is the left operand and $\boldsymbol{y}$ or d is the right operand.

$+, -, \times$	$\boldsymbol{y}$	d
$\boldsymbol{x}$	Normal bare interval result	d
c	c	$\min(c, d)$

$\div$	$\boldsymbol{y} = [0,0]$	$0 \in \boldsymbol{y} \ne [0,0]$	$0 \notin \boldsymbol{y}$	d
$\boldsymbol{x}$	`emp`	If T>`con` then `con`, else normal bare interval result	Normal bare interval result	`con`
c	`emp`	`con`	c	`con`

Square root, where $\boldsymbol{x} = [\underline{x}, \overline{x}]$.

	case	
$\sqrt{\boldsymbol{x}}$	$\overline{x} < 0$	`emp`
	$\underline{x} < 0 \le \overline{x}$	If T>`con` then `con`, else normal bare interval result
	$\underline{x} \ge 0$	Normal bare interval result
$\sqrt{c}$		`con`

is then converted back into a bare object. When $d < T$, the result is recorded as the bare decoration d; otherwise as the bare interval $\boldsymbol{z}$.

– For arithmetic operations with at least one bare decoration input, the result is always a bare decoration. A bare decoration d in $\{\texttt{emp}, \texttt{ill}, \texttt{ein}\}$ is promoted to $\emptyset_d$, and a bare decoration d in $\{\texttt{con}, \texttt{def}, \texttt{dac}, \texttt{bnd}\}$ is promoted (conceptually, not algorithmically) to an arbitrary $\boldsymbol{x}_d$ with nonempty $\boldsymbol{x}$. In the latter case, after performing the operation leading to the result $\boldsymbol{z}_d$, the tightest decoration (in the containment order (11)) enclosing all compatible $\boldsymbol{z}$ is returned.

This analysis is done conceptually; since there are only a few decorations, one can prepare complete operation tables for the decorations according to these promotion rules, and only these tables need to be implemented. They are simplified by the fact that a bare decoration input to an operation must necessarily be $< T$.

Table 5 gives these tables for the four basic operations. ⚠ These belong in an Annex, but temporarily here for inspection. I believe they are correct for all the schemes P1788 has considered.

[*Examples. In items (b) onwards, conditions (19) are assumed.*

(a) Justification for $\texttt{emp} + \boldsymbol{x} = \texttt{emp}$, *independent of* T.
This promotes to $(\emptyset, \texttt{emp}) + (\boldsymbol{x}, T) = \big(\emptyset, \min(\texttt{emp}, T, \texttt{emp})\big)$. *Since* $\texttt{emp} < T$ *this equals* $(\emptyset, \texttt{emp})$ *which gives an exception (again because* $\texttt{emp} < T$*) so is recorded as the bare decoration* `emp`. *The same holds if* $+$ *is replaced by* $-, \times$ *or* $\div$.

(b) Justification for $\boldsymbol{x} \times d = d$ *independent of* T.
Since $\boldsymbol{x}$ *is nonempty and* $d \ge \texttt{con}$, *this promotes to* $(\boldsymbol{x}, T) \times (\boldsymbol{y}, d)$ *with arbitrary nonempty* $\boldsymbol{y}$, *giving* $\big(\boldsymbol{x} \times \boldsymbol{y}, \min(T, d, e)\big)$ *where* e *is* `dac` *if* $\boldsymbol{x} \times \boldsymbol{y}$ *is bounded, otherwise* `def`. *Now* $d < T$ *so* d *cannot exceed* `def`, *hence* $d \le e$, *so* $\min(T, d, e) = d$.

(c) Justification for $c/d = \texttt{con}$ *independent of* T.
Since $c, d \ge \texttt{con}$, c/d *promotes to* $(\boldsymbol{x}, c)/(\boldsymbol{y}, d)$ *with arbitrary nonempty* $\boldsymbol{x}, \boldsymbol{y}$, *giving* $\big(\boldsymbol{x}/\boldsymbol{y}, \min(c, d, e)\big)$ *where* $e = \texttt{emp}$ *if* $\boldsymbol{y} = [0,0]$, *else* $e = \texttt{con}$ *if* $0 \in \boldsymbol{y}$, *else* $e = \texttt{dac}$. *So* $\min(c, d, e) \ge \texttt{con}$ *and can equal* `con`, *so the tightest enclosing decoration is* `con`.

(d) Justification for $\boldsymbol{x}/\boldsymbol{y}$ *when* $0 \in \boldsymbol{y} \neq [0,0]$.
$\boldsymbol{x}/\boldsymbol{y}$ *promotes to* $(\boldsymbol{x},T)/(\boldsymbol{y},T)$ *giving* $\big(\boldsymbol{x}/\boldsymbol{y}, \min(T,T,\mathtt{con})\big) = (\boldsymbol{x}/\boldsymbol{y},\mathtt{con})$. *If* $T > \mathtt{con}$ *this gives an exception so the decoration* con *is returned; if* $T = \mathtt{con}$ *it is not an exception, so the interval* $\boldsymbol{x}/\boldsymbol{y}$ *is returned.*

(e) Justification for $\sqrt{\boldsymbol{x}}$ *with* $\boldsymbol{x} = [\underline{x},\overline{x}]$ *and* $\underline{x} < 0 \leq \overline{x}$.
$\sqrt{\boldsymbol{x}}$ *promotes to* $\sqrt{(\boldsymbol{x},T)}$, *giving* $(\sqrt{\boldsymbol{x}},\mathtt{con})$ *which in the given case equals* $([0,\sqrt{\overline{x}}],\mathtt{con})$. *As with the previous item, if* $T > \mathtt{con}$ *then* con *is returned; if* $T = \mathtt{con}$ *then* $\sqrt{\boldsymbol{x}}$ *is returned.*

]