

P1788: IEEE STANDARD FOR INTERVAL ARITHMETIC VERSION 02.1

JOHN PRYCE, TECHNICAL EDITOR

CONTENTS

0. Overview	3
0.1. Scope	3
0.2. Purpose	3
0.3. Inclusions	3
0.4. Exclusions	3
0.5. The meaning of conformance	3
0.6. Programming environment considerations	3
0.7. Word usage	4
1. Introduction	5
1.1. Mathematical context	5
1.2. Specification Levels	5
1.3. The Fundamental Theorem	6
1.4. Operations	6
1.5. More to be added?	6
2. Notation, abbreviations, definitions	7
2.1. Notation and abbreviations	7
2.2. Definitions	7
3. Structure of the standard in levels	10
3.1. Specification levels overview	10
3.2. Conformance requirements	11
4. Level 1 description	12
4.1. Numbers	12
4.2. Intervals	12
4.3. Hull	12
4.4. Functions	12
5. Level 2 description	14
5.1. Number formats	14
5.2. Interval formats	14
5.3. Exception handling	15
5.4. Required operations	19
5.5. Recommended operations (informative)	22
5.6. Sharpness measures	22
6. Level 3 description	24
6.1. Representation of intervals by lower/upper bounds	24
6.2. Format conversion	24
6.3. Interchange formats	24
6.4. Support levels for interval elementary functions	25

Date: March 14, 2010.

Draft 02.1	IEEE Std P1788 IEEE Standard For Interval Arithmetic	§-1.0
6.5.	Operation tables for basic interval operations	25
7.	Level 4 description	28
Appendix A.	Language issues	28
Appendix B.	Conventions for examples in the standard	28
References		28

⚠ To the P1788 reader.

Passages in this color are my (JDP's) comments: mostly noting a blank that needs filling; or asking for answers to, and debate on, a question; or giving my opinion; or noting changes made.

I have tried to **respond** to all comments I have received that are directly about the Version 01 text. Also, George Corliss has collated comments made during voting on each motion. I have responded to some of these but much remains to be looked at.

Change tracking is somewhat informal. Previous versions can henceforth be extracted from the P1788 Subversion repository. I have marked most changes between V01 and V02.1 with a marginal note like the one shown, which indicates the author, and date of the email containing the comment that led to the change, in this case "Dan Zuras, 2010 Jan 20". The abbreviations are RBK = Kearfott, JWG = Wolff von Gudenberg, AN = Neumaier, DZ = Dan Zuras.

Traceability is not supported at all yet. That is, a 2-way mapping between decisions in motions and text in the standard. I believe we need such a system.

DRAFT 02.1

0. OVERVIEW

⚠ Scope and Purpose taken word for word from the Project Authorization Request (PAR), but changed from future to present tense. Other IEEE front-matter, such as list of participants, to be included in due course.

DZ
20100120

0.1. Scope. This standard specifies basic interval arithmetic (IA) operations selecting and following one of the commonly used mathematical interval models and at least one floating-point type defined by the IEEE-754/2008 standard. Exception conditions are defined and standard handling of these conditions are specified. Consistency with the model is tempered with practical considerations based on input from representatives of vendors and owners of existing systems.

The standard provides a layer between the hardware and the programming language levels. It does not mandate that any operations be implemented in hardware. It does not define any realization of the basic operations as functions in a programming language.

0.2. Purpose. The standard's aim is to improve the availability of reliable computing in modern hardware and software environments by defining the basic building blocks needed for performing interval arithmetic. There are presently many systems for interval arithmetic in use, and lack of a standard inhibits development, portability, and ability to verify correctness of codes.

0.3. Inclusions. This standard specifies

- Formats for interval data based on underlying floating-point formats.
- Constructors for intervals from floating-point and character sequence data.
- Addition, subtraction, multiplication, division, fused multiply add, square root, compare, and other interval-valued operations for intervals.
- Midpoint, radius and other numeric functions of intervals.
- Mandatory elementary functions.
- Data type and operations for the calculation of exact sums or dotproducts.
- Conversions between different interval formats.
- Conversions between interval formats and external representations as character sequences.
- Interval-related exceptions and their handling.

JWvG
20091209

0.4. Exclusions. This standard does not specify

- Which floating-point formats supported by the underlying system shall have an associated interval format.
- Details of how an implementation represents intervals internally by floating-point numbers. However interchange formats for intervals are specified.

⚠ To be revised later.

0.5. The meaning of conformance. This standard specifies interval arithmetic in terms of an underlying floating-point system. An implementation, to conform to this standard, shall support at least one interval format associated with a floating-point format of the underlying system. It probably will support several such formats. The standard does not require the underlying floating-point system to be 754-conforming. However it introduces the notion of a 754-conforming implementation, having more stringent requirements, mainly concerning support for mixed-format interval operations.

0.6. Programming environment considerations.

⚠ This is taken from 754§1.5 and will surely need modification. I believe we MUST in due course include language requirements in P1788.


This standard does not define all aspects of a conforming programming environment. Such behavior should be defined by a programming language definition supporting this standard, if

JWvG
20091209

available, and otherwise by a particular implementation. Some programming language specifications might permit some behaviors to be defined by the implementation.

Language-defined behavior should be defined by a programming language standard supporting this standard. Then all implementations conforming both to this interval standard and to that language standard behave identically with respect to such language-defined behaviors. Standards for languages intended to reproduce results exactly on all platforms are expected to specify behavior more tightly than do standards for languages intended to maximize performance on every platform. Because this standard requires facilities that are not currently available in common programming languages, the standards for such languages might not be able to fully conform to this standard if they are no longer being revised. If the language can be extended by a function library or class or package to provide a conforming environment, then that extension should define all the language-defined behaviors that would normally be defined by a language standard. Implementation-defined behavior is defined by a specific implementation of a specific programming environment conforming to this standard. Implementations define behaviors not specified by this standard nor by any relevant programming language standard or programming language extension. Conformance to this standard is a property of a specific implementation of a specific programming environment, rather than of a language specification. However a language standard could also be said to conform to this standard if it were constructed so that every conforming implementation of that language also conformed automatically to this standard.

0.7. Word usage.

⚠ (Lifted from 754.)  this standard three words are used to differentiate between different levels of requirements and optionality, as follows:

- **may** indicates a course of action permissible within the limits of the standard with no implied preference (“may” means “is permitted to”);
- **shall** indicates mandatory requirements strictly to be followed in order to conform to the standard and from which no deviation is permitted (“shall” means “is required to”);
- **should** indicates that among several possibilities, one is recommended as particularly suitable, without mentioning or excluding others; or that a certain course of action is preferred but not necessarily required; or that (in the negative form) a certain course of action is deprecated but not prohibited (“should” means “is recommended to”).

Further:

- **might** indicates the possibility of a situation that could occur, with no implication of the likelihood of that situation (“might” means “could possibly”);
- **see** followed by a number is a cross-reference to the clause or subclause of this standard identified by that number;
- **Note** introduces text that is informative (that is, is not a requirement of this standard).

1. INTRODUCTION

This introduction explains some of the alternative interpretations, and sometimes competing objectives, that influenced the design of this standard, but is not part of the standard.

1.1. Mathematical context. Interval computation is a collaboration between human programmer and machine infrastructure which, correctly done, produces mathematically proven numerical results about continuous problems—for instance, rigorous bounds on the global minimum of a function or the solution of a differential equation. It is part of the discipline of “constructive real analysis”. In the long term, the results of such computations may become sufficiently trusted to be accepted as contributing to legal decisions. The machine infrastructure acts as a body of theorems on which the correctness of an interval algorithm relies, so it must be made as reliable as is practical. In its logical chain are many links—hardware, underlying floating-point system, etc.—over which this standard has no control. The standard aims to strengthen one specific link, by defining interval objects and operations that are theoretically well-founded and practical to implement. There are several mathematical bases of interval computation, which are not wholly compatible with each other. Thus it was necessary to make a choice for the purposes of this standard. The working group generally agreed that a high priority should be given to making the mathematical basis easy to grasp, easy to teach, and easy to interpret in the context of real-world applications. We believe the one we have chosen is the best match for those aims. In this theory

- Intervals are sets.
- They are subsets of the real numbers \mathbb{R} .
- An interval operation is defined algebraically, in contrast to topologically. The interval version of an elementary function such as $\sin x$ is essentially the natural extension to sets of the corresponding pointwise function on real numbers.

This contrasts on the one hand with *Kaucher* or *modal* interval theory, where an interval is a formal object, an ordered pair of real numbers \underline{x}, \bar{x} , of which those having $\underline{x} \leq \bar{x}$ are interpretable as intervals in the set sense; and on the other hand with *containment set* (cset) theory, where intervals are subsets of the extended reals \mathbb{R}^* , and operations are defined topologically, in terms of limits. In this standard, the set \mathbb{IR} of intervals comprises precisely the closed and connected (in the topological sense) subsets of \mathbb{R} . This includes the empty set, as well as intervals that are unbounded on one or both sides.

1.2. Specification Levels. The 754-2008 standard describes itself as layered into four Specification Levels. To manage complexity, P1788 uses a corresponding structure: level 1, of mathematical *interval theory*; level 2, the finite set of *interval datums* in terms of which finite-precision interval computation is defined; level 3, of *representations* of intervals by floating-point numbers; level 4, of *bit strings* and memory.

There is another important player: the programming language. We acknowledge the experience of the 754-2008 working group, who recognized a serious defect of the 754-1985 standard, namely that it specified individual operations but not how they should be used in expressions. Over the years, compilers made clever transformations so that it became impossible to know the precisions used and the roundings performed while evaluating an expression, or whether the compiler had even “optimized away” $(1.0 + x) - 1.0$ to become simply x .

This is also a problem for intervals. Thus the standard makes requirements and recommendations on language implementations, thereby defining the notion of a standard-conforming *implementation of intervals within a language*.

The language does not constitute a fifth level in some linear sequence; from the user’s viewpoint it sits above datum level 2, alongside theory level 1, as a practical means to implement interval

algorithms by manipulating level 2 entities (though most languages have influence on levels 3 and 4 also).

1.3. The Fundamental Theorem. Ramon Moore’s Fundamental Theorem of Interval Arithmetic (FTIA) is central to interval computation. Roughly, it says that if f is an *explicit expression* defining a real function $f(x_1, \dots, x_n)$, then evaluating f “in interval mode” over any interval inputs $(\mathbf{x}_1, \dots, \mathbf{x}_n)$ is guaranteed to give an enclosure of the range of f over those inputs. A version of the FTIA holds in all variants of interval theory, but with varying hypotheses and conclusions. A standard must adhere to one theory, for if ambiguities or inconsistencies exist, a user may believe the FTIA holds in a case where it does not, with possibly serious results in applications. As stated, the FTIA is about the mathematical level. Moore’s achievement was to see that “outward rounding” makes the FTIA hold also in finite precision, and to follow through the consequences. An advantage of the level structure used by the standard is that the mapping between levels 1 and 2 defines a framework where it is easily proved—indeed almost obvious—that

A conforming implementation obeys the finite-precision FTIA.

This holds also for mixed-format computation, which is supported by the standard.

1.4. Operations.

There are several interpretations of *evaluation outside an operation’s domain* and *operations as relations rather than functions*. This includes classical alternative meanings of division by an interval containing zero. Consider $\mathbf{y} = \sqrt{\mathbf{x}}$ where $\mathbf{x} = [-1, 4]$.

1. In *optimization*, when computing lower bounds on the objective function, it is generally appropriate to return the result $\mathbf{y} = [0, 2]$, and ignore the fact that $\sqrt{\cdot}$ has been applied to negative elements of \mathbf{x} .
2. In applications where one must check the hypotheses of a *fixed point theorem* are satisfied (such as solving differential equations):
 - (a) one may need to be sure that the function is defined and continuous on the input and, hence, throw an illegal argument exception when, as in the above case, this fails; or
 - (b) one may need the result $\mathbf{y} = [0, 2]$, but must flag the fact that $\sqrt{\cdot}$ has been evaluated at points where it is undefined or not continuous.
3. In *constraint propagation*, the equation is often to be interpreted as: find all y such that $y^2 = x$ for some $x \in [-1, 4]$. In this case the answer is $[-2, 2]$.

The standard provides means to meet these diverse needs, hopefully without compromising clarity and efficiency. It also aims to give limited support for alternative interval theories, such as Kaucher arithmetic, for which none of the above three interpretations is appropriate.

1.5. More to be added?

⚠ I guess we need something on exception handling.

2. NOTATION, ABBREVIATIONS, DEFINITIONS

2.1. Notation and abbreviations.

754	IEEE-Std-754-2008 “IEEE Standard for Floating-Point Arithmetic”.
\mathbb{R}	the set of real numbers.
\mathbb{R}^*	the set of extended real numbers, $\mathbb{R} \cup \{-\infty, +\infty\}$.
\mathbb{IR}	the set of bounded, nonempty closed real intervals.
$\overline{\mathbb{IR}}$	the set of closed real intervals, including unbounded intervals and the empty set.
\mathbb{F}	the subset of \mathbb{R}^* representable in a given floating point format.
\mathbb{IF}	the intervals of \mathbb{IR} whose bounds are in \mathbb{F} .
$\overline{\mathbb{IF}}$	the intervals of $\overline{\mathbb{IR}}$ whose bounds are in \mathbb{F} .
NaI	Not an Interval.
NaN	Not a Number.
qNaN	quiet NaN.
sNaN	signaling NaN.
x, y, \dots [resp. f, g, \dots]	generic notation for a numeric value [resp. numeric function].
$\mathbf{x}, \mathbf{y}, \dots$ [resp. $\mathbf{f}, \mathbf{g}, \dots$]	generic notation for an interval value [resp. interval function].
f, g, \dots	generic notation for an expression, producing a function by evaluation.

⚠ Some of these may not be used.

2.2. Definitions.

⚠ Following comments from DZ I have rewritten this subclause to be self-contained, i.e. one does not need to look at forward references to understand the definitions. One or two definitions have been removed since V01, several added, but more will no doubt need adding.

2.2.1. 754-conforming system (754 system for short). A programming environment that provides floating point arithmetic conforming to IEEE754-2008. [Note. A hardware processor (e.g. the Cell processor) may not be 754-conforming in itself, but a 754-conforming system may be built on it with software assistance.]

2.2.2. basic 754 format. One of the five 754 floating point formats binary32, binary64, binary128, decimal64, decimal128.

2.2.3. basic (arithmetic) operation. One of the five elementary functions $+$, $-$, \times , \div , $\sqrt{\cdot}$.

2.2.4. ci-format (concrete interval format). An interval format associated with a specific number representation. Usually the inf-sup format associated with a specific floating point format. Details in 5.1, 5.2.

2.2.5. fma. Fused multiply-add operation.

2.2.6. hull. The (interval) hull of a subset \mathbf{s} of \mathbb{R} is the tightest interval containing \mathbf{s} . When \mathbb{F} is a number format, the \mathbb{F} -hull of \mathbf{s} is the tightest \mathbb{F} -interval containing \mathbf{s} .

2.2.7. implementation. When used without qualification, means an implementation of this standard.

2.2.8. inf-sup. Describes a representation of an interval based on its lower and upper bounds.

2.2.9. interval datum [respectively, \mathbb{F} -interval datum]. A member of the finite set of intervals representable in a specific (inf-sup unless said otherwise) interval format [resp. the one associated with the number format \mathbb{F}]. Details in 5.2.

W J R T [J G C

JWvG
20091209DZ
20100120

2.2.10. **interval elementary function.** An interval version of a point elementary function, that is provided by an implementation. The set of these is the implementation’s **interval elementary function library** (**interval library** for short). These terms may be qualified by a format, e.g. “binary64 interval library”.

2.2.11. **interval extension.** An interval extension of a point function f is a function \mathbf{f} from intervals to intervals such that $f(x)$ belongs to $\mathbf{f}(\mathbf{x})$ whenever x belongs to \mathbf{x} and $f(x)$ is defined. Details in 4.4.1.

2.2.12. **i-format, interval format, radix of i-format.** The interval format associated with a number format \mathbb{F} means the set of interval datums (\mathbb{F} -intervals for short) associated with \mathbb{F} in a particular representation. When used without qualification, inf-sup representation is assumed — by contrast with, say, mid-rad representation. The radix of an i-format means the radix of its n-format. Details in 5.2.

2.2.13. **interval function, interval mapping.** A function from intervals to intervals is called an interval function if it is an interval extension of a point function, and an interval mapping otherwise. Details in 4.4.1.

2.2.14. **interval library.** See Definition 2.2.10.

2.2.15. **interval version.** Interval version of a point function means the same as interval extension; often used with an indication of its operand and destination formats, as in “binary64 SRMF interval version”.

2.2.16. **mathematical interval of constructor.** The arguments of an interval constructor, if valid, define a mathematical interval \mathbf{x} . The actual interval returned by the constructor is the tightest interval of the destination format that contains \mathbf{x} . Details in 5.4.1.

2.2.17. **mid-rad.** Describes a representation of an interval based on its midpoint and radius.

2.2.18. **mixed-format.** Describes an interval arithmetic operation where the formats of the operand interval(s) and the destination interval may not all be the same.

[Note. *Mixed-format does not mean that the lower and upper bounds of an individual interval object can have different number formats.*]

2.2.19. **MRMF.** Mixed-radix, mixed-format. Describes a level of arithmetic support where operand(s) and destination of an interval operation may be an arbitrary mix of supported formats, not necessarily of the same radix. Details in 6.4.

2.2.20. **multiple-format.** Multiple-format interval arithmetic means support for arithmetic in more than one interval format, and for conversions between these formats.

[Note. *SRSF, SRMF and MRMF interval support are all multiple-format (if more than one interval format is supported). SRMF and MRMF, but not SRSF, are mixed-format.*]

2.2.21. **n-format, number format.** A subset of the extended reals \mathbb{R}^* , containing $\pm\infty$. In practice, it is usually the finite set of numbers representable in a specific floating point format and is then the n-format **associated** with that format. Details in 5.1.

2.2.22. **point elementary function.** A point function, in the sense of 4.4, that is provided by an implementation. The set of these is the implementation’s point elementary function library (point library for short). These terms may be qualified by a format, e.g. “binary64 point library”. The basic operations count as elementary functions.

2.2.23. **sharpness measure.** A way to describe the quality of an interval version of a function. See 5.6.

2.2.24. **SRMF.** Single-radix, mixed-format. Describes a level of arithmetic support where operand(s) and destination of an interval operation may be an arbitrary mix of supported formats, but must be of the same radix. Details in 6.4.

DZ
20100120

2.2.25. **SRSF.** Single-radix, single-format. Describes a level of arithmetic support where operand(s) and destination of an interval operation must all be of the same (supported) format. Details in 6.4.

JWvG
20091209

2.2.26. **support.**

DZ
20100120

- An implementation **supports** an abstract interval format $\overline{\mathbb{F}}$ if it provides a concrete interval format that represents $\overline{\mathbb{F}}$. Details in 5.1, 5.2.
- A function f is **interval-supported** in an interval format $\overline{\mathbb{F}}$ if there is an interval version of f whose destination format is $\overline{\mathbb{F}}$. For levels of interval support (MRMF, SRMF and SRSF) see Definition 2.2.19, 2.2.24, 2.2.25.

2.2.27. **tightest.** Smallest in the set sense. The tightest set (unique, if it exists) with a given property is contained in every other set with that property.

Relationships between specification levels for interval arithmetic (for a given format)		
Level 1	Number system \mathbb{R} . Set $\overline{\mathbb{R}}$ of allowed intervals over \mathbb{R} . Principles of how $+$ $-$ \times \div and standard functions are extended to intervals.	Mathematical Model level.
	\downarrow <i>interval hull</i> total, many-to-one ^d	\uparrow <i>identity map</i> total, one-to-one ^c
Level 2	The set $\overline{\mathbb{F}}$ of “machine intervals” in $\overline{\mathbb{R}}$	Interval datum level.
		\uparrow “represents” partial, many-to-one, onto ^b
Level 3	Representation of nonempty $[x, \bar{x}]$ as two FP numbers x, \bar{x} , or alternative. Repres’n of \emptyset .	Representations of interval data.
		\uparrow “encodes” partial, many-to-one, onto ^a
Level 4	Encodings 0111000...	Bit strings.

TABLE 1. Specification levels for interval arithmetic

⚠ Made more precise since V01. Someone suggested there should be well-defined mappings Level $2 \rightarrow 3 \rightarrow 4$. Seems to me this represents either an unenforceable requirement, or a severe performance penalty. Could whoever it was, please clarify?

3. STRUCTURE OF THE STANDARD IN LEVELS

3.1. Specification levels overview.

The standard is structured into four levels, summarized in Table 1, that match the levels defined in the 754 standard, see 754 Table 3.1.

Level 1, in Clause 4, defines the mathematical theory underlying the standard. The entities at this level are mathematical intervals and operations on them. Conforming implementations shall implement this theory.

Level 2, in Clause 5, is the central part of the standard. Here the mathematical theory is approximated by an implementation-defined finite set of entities and operations. A level 2 entity is called a *datum* (plural “datums” in this standard, since “data” is often misleading). In addition to an ordinary (bare) interval, this level defines a *decorated* interval, comprising a bare interval and a *decoration*. Decorations implement the P1788 exception handling mechanism.

Level 3, in Clause 6, defines the representation of interval datums in terms of underlying floating-point datums. A level 3 entity is an *interval object*.

Level 4, in Clause 7, is concerned with the encoding of interval objects. A level 4 entity is a *bit string*, though it may be broken into non-contiguous parts in memory.

The arrows denote mappings between levels. The phrases in italics name these mappings. Each phrase “total, many-to-one”, etc., labeled with a letter ^a to ^d, is descriptive of the mapping and is equivalent to the corresponding labeled fact below.

- a. Not every interval encoding necessarily encodes an interval object, but when it does, that object is unique. Each interval object has at least one encoding, and may have more than one.

- b. Not every interval object necessarily represents an interval datum, but when it does, that datum is unique. Each interval datum has at least one representation, and may have more than one.
- c. Each interval datum is a mathematical interval.
- d. For a given format \mathbb{F} , each mathematical interval (indeed each subset of \mathbb{R}) has a unique interval datum as its \mathbb{F} -hull.

3.2. Conformance requirements.

- Implementations shall be based on the mathematical model in Clause 4.
- An implementation shall specify which interval formats it supports (it is permitted to support only one). Support consists of the following two items.
 - (a) For each supported interval format, operations as specified in 5.4 shall be provided, to at least single-radix single-format (SRSF) level—see 6.4.
 - (b) Conversions as defined in 5.2.3 shall be provided between any two supported interval formats.
- An implementation shall be 754-conforming (for a specified set of formats) if: (i) its underlying system is 754-conforming (Definition 2.2.1) for this set of formats, and the implementation supports the corresponding interval format to at least SRSF level; or (ii) it is functionally indistinguishable from case (i). *[Note. The reason for case (ii) is that it seems feasible to code an efficient 754-conforming interval system using only a subset of 754 features. If so, a floating-point system that behaves differently from 754 outside this subset can be used. For example, a system that has only one kind of zero.]*
- A 754-conforming implementation shall provide single-radix multiple-format (SRMF) support to the same extent that the underlying floating-point provides formatOf operations, see 6.4. It may support interval formats outside the five basic 754 formats, such as extended or extendable, see 754§3.7.
- An implementation shall document:
 - (a) Which interval formats are supported.
 - (b) Which interval versions of elementary functions are provided in a given format.
 - (c) For each such version: (i) a sharpness measure as defined in 5.6; (ii) the level of mixed-format support, SRSF or SRMF; and which operand interval formats are supported in the SRMF case.

4. LEVEL 1 DESCRIPTION

This clause describes the theory of mathematical intervals and operations, on which this standard is based.

4.1. Numbers. Following the terminology of 754 (§2.1.25 and elsewhere), any member of the extended reals $\mathbb{R} \cup \{-\infty, +\infty\}$ is called a number: it is a **finite number** if it is in the reals \mathbb{R} , else an **infinite number**.

4.2. Intervals. The set of mathematical intervals in \mathbb{R} , denoted \mathbb{IR} , comprises the empty set \emptyset together with all closed nonempty intervals of real numbers

$$\mathbf{x} = [\underline{x}, \bar{x}] := \{x \in \mathbb{R} \mid \underline{x} \leq x \leq \bar{x}\},$$

where $-\infty \leq \underline{x} \leq \bar{x} \leq +\infty$.

[Note.

- The above definition implies $-\infty$ and $+\infty$ can be bounds of an interval, but are never members of it. The requirement that \mathbf{x} be nonempty implies \underline{x} cannot be $+\infty$, and \bar{x} cannot be $-\infty$; the notations $[-\infty, -\infty]$ and $[+\infty, +\infty]$ are regarded as semantically invalid (rather than denoting the empty set) at the mathematical level.
- Following the traditional notation $(\underline{x}, \bar{x}) = \{x \mid \underline{x} < x < \bar{x}\}$, and so on, the round bracket notation for closed intervals with an infinite end point may be used; e.g. $[2, +\infty)$ is the same as $[2, +\infty]$.

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

4.4. 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 $n \geq 0, m > 0$. When $n = 0$ and $m = 1$, we have a **named real constant**. When not otherwise specified, a scalar function is assumed, i.e. $m = 1$. If $m > 1$, the function is called a vector function. D is the domain of f , also written D_f . To specify n , call f an n -variable point function, or denote f as

$$f(x_1, \dots, x_n).$$

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

$$\text{range}(f; \mathbf{s}) = \{f(x) \mid x \in \mathbf{s} \text{ and } x \in D_f\}.$$

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

[Note.

1. Here, f is a mapping, not an expression.
2. For instance $\text{range}(\sqrt{\cdot}; [-1, 1]) = [0, 1]$. *This follows the usual mathematical convention that when evaluating over sets, points outside the domain of a function are simply ignored.*
3. For the 754 policy on evaluating point functions outside the domain, see 754§9.1.1.

]

4.4.1. *Interval mappings and functions.* Unless otherwise specified, an interval mapping is a mapping \mathbf{f} from \mathbb{IR}^n to \mathbb{IR}^m for some $n \geq 0, m > 0$. To specify n , call \mathbf{f} an “ n -variable interval mapping”, or denote \mathbf{f} as $\mathbf{f}(x_1, \dots, x_n)$. As with point functions, $m = 1$ is assumed unless said otherwise.

An interval mapping is called an **interval function** if it is an interval version of some point function, as defined next. Examples of interval mappings that are not interval functions are the interval intersection and union operations, $(x, y) \mapsto x \cap y$ and $(x, y) \mapsto \text{hull}(x \cup y)$.

Given an n -variable point function f , an **interval extension** of f , also called an **interval version** of f , is an interval mapping \mathbf{f} such that

$$\mathbf{f}(x) \supseteq \text{range}(f; x)$$

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

$$\mathbf{f}(s) = \text{hull}(\text{range}(f; s))$$

for any subset s of \mathbb{R}^n . Equivalently, using multiple-argument notation for f , an interval extension satisfies

$$\mathbf{f}(s_1, \dots, s_n) \supseteq \text{range}(f; s_1, \dots, s_n),$$

and the sharp interval extension satisfies

$$\mathbf{f}(s_1, \dots, s_n) = \text{hull}(\text{range}(f; s_1, \dots, s_n)).$$

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

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

[Note.

- *Example.* With these definitions, the sharp interval extensions of multiplication and division satisfy $x \times \{0\} = \{0\}$ for any nonempty interval x , and $x/0 = \emptyset$, for any interval x .
- All interval functions used here are automatically defined for all arguments—e.g. for the sharp extension of “point square root”, $\sqrt{[-1, 4]} = [0, 2]$, $\sqrt{[-2, -1]} = \emptyset$. At level 2, the exception handling mechanism gives a specific way to diagnose when a function has been evaluated outside its domain, but no specific method is defined at level 1.

]

5. LEVEL 2 DESCRIPTION

The general level 2 interval datum is a decorated interval—an ordered pair (interval, decoration). Decorations implement exception handling. An interval without decoration may be called a bare interval to emphasize the distinction. Bare intervals are described in 5.2 and the exception handling mechanism in 5.3. This clause defines interval formats, which are used to represent a finite subset of the set \mathbb{IR} of mathematical intervals. Each format is derived from an associated number format, usually a floating-point format, which characterizes it uniquely. The choice of which of this standard's formats to support is language-defined or, if the relevant language standard is silent or defers to the implementation, implementation-defined. The names used for formats in this standard are not necessarily those used in programming environments.

5.1. Number formats. A number format (n-format) is a finite subset \mathbb{F} of the extended reals \mathbb{R}^* , containing $-\infty$ and $+\infty$. A floating-point format in the 754 sense, such as binary64, is identified with the n-format comprising those extended-real numbers that are exactly representable in that format, where -0 and $+0$ both represent the mathematical number 0.

When it is necessary to distinguish, the floating-point format is called a concrete n-format, and the subset of \mathbb{R}^* is the abstract n-format associated to it. An n-format need not be associated to a 754 format. It may come from the numbers of some floating-point system that is not 754-conforming, or from a fixed-point number representation, etc.

[Note. In view of this definition, 754's -0 and $+0$ are considered identical. Also, in 754 decimal formats, representations in the same cohort are considered identical.]

5.2. Interval formats.**5.2.1. \mathbb{F} -intervals.**

Let \mathbb{F} be an n-format. An \mathbb{F} -interval is either the empty set, or a mathematical interval whose endpoints are in \mathbb{F} . It may be referred to as an \mathbb{F} -interval datum to emphasize that it is a level 2 object. The interval format (i-format) associated with \mathbb{F} means the set of \mathbb{F} -interval datums. It is denoted by \mathbb{IF} .

[Note.

- This definition implies an inf-sup (infimum-supremum) representation of intervals, or an equivalent, at level 3.
- A level 2 interval, being a set of reals, does not “know” what format it belongs to. E.g., $x = [1, 2]$, which is exactly representable in binary32, binary64 and decimal64, is the same datum whichever format it derives from. At level 3 however, it would be represented by three different objects, of the corresponding concrete interval formats (§6.1).
- An n-format uniquely determines an i-format (also vice versa), and a 754 concrete format uniquely determines an n-format.
- An example of a possibly useful n-format that is not associated to a concrete format (but is derived from one), is an “underflow flushed to zero” system. Say the concrete format is binary64. Then \mathbb{F} is defined to consist of all binary64 numbers that are not subnormal. Interval arithmetic based on such an \mathbb{F} may give speed advantages on some architectures.
- Since \mathbb{F} -intervals are sets of reals, it is meaningless to speak of an interval datum that has NaN as an endpoint, or has its lower bound greater than its upper bound. Such notions make sense at level 3 but not at level 2.

]

5.2.2. Finite precision hull. The (interval) \mathbb{F} -hull of an arbitrary subset s of \mathbb{R} , written $\text{hull}_{\mathbb{F}}(s)$, is the tightest \mathbb{F} -interval that contains s .

[Note.

- The set $\text{hull}_{\mathbb{F}}(s)$ always exists, because \mathbb{F} is finite and contains ± 1 .
 - Always, $\text{hull}_{\mathbb{F}}(s)$ contains $\text{hull}(s)$. If n -format \mathbb{G} (as a subset of \mathbb{R}^*) contains n -format \mathbb{F} — equivalently, if \mathbb{G} is wider than \mathbb{F} in the sense of 754§2.1.36 — then $\text{hull}_{\mathbb{F}}(s)$ contains $\text{hull}_{\mathbb{G}}(s)$.
-]

5.2.3. *Interval format conversion.* If \mathbb{F} is an n -format, the **conversion** to format \mathbb{F} is the operation that maps an interval x of any supported format to its \mathbb{F} -hull,

$$y = \text{hull}_{\mathbb{F}}(x).$$

5.3. Exception handling.

⚠ This subclause is JDP's interpretation of Motion 8. I have made many changes in response to Neumaier's comments, to clarify various points, but it is time for motions to decide others. References to nonstandard intervals have been removed pending a decision on these. I have made assumptions about the supported trits and how trits are set by operations, based on discussions on Motion 8.

AN
20100129

Exceptional events in a P1788 implementation are recorded by means of *decorations* that are added to interval datums in sections of a program where such recording is needed, and removed in sections where it is safe to do so and speed is essential. The general aim, as in 754, is that exceptions do not interrupt the flow of computation. In contrast with 754, there are no global flags in the standard P1788 exception model, though an implementation may add such flags.

5.3.1. *Definitions.* A **trit** is a quantity taking three values interpreted as “certainly false”, “uncertain = possibly false = possibly true” and “certainly true”, and denoted by the symbols $-$, 0 and $+$ or the numbers $-1, 0, 1$.

AN
20100129

A decoration is a list of trits (more precisely, of named trit-valued attributes). The values $+$ and $-$ of a trit make opposite certainty claims about the associated attribute; the value 0 indicates the lack of certainty about the attribute. A “new” interval created from a constructor has a no-0 decoration of the appropriate form. The all-0 decoration is least informative.

A **decorated interval** consists of one interval and one decoration. Decorations are a level 2 concept, hence the interval part of a decorated interval is always a machine interval. However, in examples, it is convenient to assume exact arithmetic on arbitrary mathematical intervals.

AN
20100129

An interval or decoration is called a bare interval or bare decoration when the bareness is to be emphasized.

5.3.2. *Interval types.* Implementations shall provide:

- a bare interval type, for each supported number format;
- a bare decoration type as defined in 5.3.5, which is independent of the number format;
- a decorated interval type, for each supported number format: this is made up of the corresponding bare interval type, and the bare decoration type.

Every constructor for intervals creates a decorated interval with the most informative decoration consistent with the interval part.

AN
20100129

An implementation may support other decoration types which carry extra information such as other trits, or traceback data for debugging, etc.

5.3.3. *Forgetful operations.* Implementations shall provide operations

AN
20100129

intervalPart(x)
decorationPart(x)

that return the interval or decoration part of the input, respectively.

There shall be operations

Name	Values.		
	+	0	–
valid	<i>isValid</i>	x	<i>notValid</i>
defined	<i>isDefined</i>	<i>possiblyDefined</i>	<i>notDefined</i>
continuous	<i>isContinuous</i>	<i>possiblyContinuous</i>	x
bounded	<i>isBounded</i>	<i>possiblyBounded</i>	<i>notBounded</i>

FIGURE 1. Mnemonic for each trit value. An x marks a value that is not used, as explained in the text.

valid(x)
defined(x)
continuous(x)
bounded(x)

that act on either a decorated interval or a bare decoration, and return the value of the indicated trit. These four operations shall also act on bare intervals, in which case the result is the least informative among the possible results for all decorated intervals consistent with the input.

[*Example.* For the bare interval $x = [0, +\infty]$, valid(), defined(), continuous() and bounded() return 0, 0, 0 and – respectively. For $x = \emptyset$ they return 0, +, + and +.]

5.3.4. *Decorated-interval functions.* Implementations shall provide the same set of elementary functions (this includes the basic arithmetic operations) for decorated intervals as for bare intervals. This applies to both required and recommended functions, see 5.4.

An elementary function applied to decorated intervals returns, by definition, a decorated interval whose interval is the result of the operation on the argument intervals, and whose decorations are computed from the arguments such that they retain the most informative and valid information about the result, as specified in 5.3.5. Hence

- The interval part of the result depends only on the interval part of the inputs.
- The decoration part of the result depends, in general, on both interval and decoration parts of the inputs. ⚠ Oh dear. Also the result of comparisons such as *containedIn*(x, y) (Vienna 5.4) depends on both interval part and bounded attribute. I'm not sure I like this because...

Motion 8§2.2 says “An arithmetic operation on standard decorated intervals returns a standard decorated interval whose interval is the result of the operation on the argument intervals, and whose decorations are computed from the arguments such that ...”

The intent of this is surely that the interval part of an output depends only on the interval parts of the inputs. Should we relax this generally? Should we make a special exception for bounded since this aims to fill a hole in the 754 standard? Should comparisons obey different rules from arithmetic operations? Should we dump bounded?

An operation on bare decorations returns, by definition, a bare decoration representing the least informative case of all possible results of the operation on all decorated intervals consistently decorated according to the bare input decorations.

5.3.5. *Supported attributes.* The decoration type shall support the four attributes

valid, defined, continuous, bounded.

It may support others.

Figure 1 gives mnemonics for attribute values. Their usage is that “interval x is *notValid*” means the valid attribute of x has the value –, and so on.

X F S X X J W
Y M J Y J Y
G W F H P J Y X

5.3.6. *Attribute semantics.* The attributes are used to guarantee certain properties of an individual interval or of the computational history that produced it. By the nature of interval computation one cannot, except for `valid`, state simple *necessary and sufficient* conditions for an attribute to have a given value, but the positive values—`isValid`, `isDefined`, `isContinuous`, `isBounded`—give *sufficient* conditions for properties needed in application algorithms, as described below.

Care has been taken in this standard to make the semantics *implementable*. Interval operations are specified in such a way that: (a) they have required semantic properties; (b) they can be realized in software with good efficiency on modern architectures.

⚠ **Implementability is an important feature! But is this appropriate to state in the standard?**



valid. An interval is `isValid` if, and only if, it is the result of a valid constructor call, or of an operation whose inputs were `isValid`. Otherwise it is `notValid`. The `possiblyValid` value is not used.

[*Note. An `isValid` interval is one that “makes sense” at the code level, irrespective of whether the algorithm that produced it is correct.*



All `notValid` intervals originate from invalid constructions, e.g. $x = \text{doublesToInterval}(3, \text{NaN})$ (meaningless bound) and $y = \text{stringToInterval}("[0.2.0.4]")$ (syntax error: the second period should be a comma) are `notValid`, as are $x + z$ for any z , $\sin(y)$, etc. Real-valued functions of a `notValid` interval, such as its bounds, radius, etc., return `NaN`. Relational operations with a `notValid` input always return `False`. All other decoration trits of a `notValid` interval have the value 0.]

defined, continuous. These have meaning in the context of the Fundamental Theorem of Interval Arithmetic (FTIA). Suppose f is a function of real variables x_1, \dots, x_n defined by an explicit expression $f(x_1, \dots, x_n)$ built of elementary functions, and that the decorated interval $y = f(x_1, \dots, x_n)$ is computed by applying f to decorated intervals x_1, \dots, x_n , using decorated-interval versions of the elementary functions. Let \mathbf{X} be the input box $x_1 \times \dots \times x_n$ (meaning the cartesian product of the interval parts of the x_i , dropping the decorations). The FTIA asserts that y contains $\text{range}(f, \mathbf{X})$, irrespective of whether f is everywhere defined, or continuous, on \mathbf{X} . Further:

- (i) y being `isDefined` guarantees that the mathematical function f is defined everywhere on \mathbf{X} : that is, \mathbf{X} is contained in the natural domain D_f of f .
- (ii) y being `isContinuous` guarantees that the restriction to \mathbf{X} of the mathematical function f is defined and continuous everywhere (on \mathbf{X}).

[*Notes.*

1. A common use of these attributes is to let an algorithm verify the mathematical hypotheses of a fixed-point theorem, e.g., during the validated solution of a differential equation problem.
2. For brevity, the wording “ f is continuous on \mathbf{X} ” is used to mean “the restriction of f to \mathbf{X} is continuous”. This is sufficient for fixed-point theorems, but note it is not the same as “the whole function f is continuous at each point of \mathbf{X} ”. For instance, if $f(x) = \text{floor}(x)$, the restriction of f to $x = [0, \frac{1}{2}]$ is continuous, but the whole function f is discontinuous at $0 \in x$.
3. To make these features useful it must be ensured that on entry to f , each x_j is an `isDefined` [respectively `isContinuous`] decorated interval.
4. The above are sufficient conditions for f to be defined, or continuous, on \mathbf{X} . They are in general not necessary because of overestimation due to three factors, of which only the last can be controlled by the implementation:
 - (a) Dependencies within the expression f .

- (b) Effects of outward rounding.
- (c) The tightness with which the interval elementary functions are implemented.
5. The operations making up f above are, by definition, point elementary functions (for f) or their interval versions (for f). The theory allows f to contain logical branches in some circumstances,  reference? but f may not use non-point operations such as intersection or hull of two intervals.  Sometimes to prove f is defined or continuous on X one covers X with smaller boxes and proves the property on each such box. The standard does not directly support such methods but I guess the way intersection, hull and other set operations handle defined and continuous, should aim to be convenient for such methods. Comments please.

]

bounded. This attribute aims to record whether endpoints of an interval are finite, including cases where they are finite but too large to be representable. An interval is set **isBounded** if can be determined from the operation that produced it, and the latter's inputs, that it is bounded. Similarly, it is set **notBounded** if can be determined that it is definitely unbounded. Otherwise it is set **possiblyBounded**.


[Notes. The **valid**, **defined** and **continuous** attributes have sticky behaviour. This means that their values are regarded as ordered +, 0, – from “best” to “worst”, and for most (but not all) operations, the result value is the worst of (a) the input values and (b) the value set by the operation itself. In this way they record past computation history. The **bounded** attribute is not sticky.]

5.3.7. *Allowed combinations.* Only certain combinations of property values are logically possible, as shown below, where v means **valid**, and so on. An implementation shall ensure this.

v	d	c	b	no. of cases
–	0	0	0	1
+	–	0	0	1
+	0	0	0–	2
+	+	+0	+0–	6

\ W T S L K T

The completeness of this table depends on the following conventions adopted in this standard.

- A function can be **continuous** only at a point where it is defined. (E.g., the function $f(x) = x$ ($x \neq 0$), undefined ($x = 0$) is not **continuous** at $x = 0$. This convention is not universal.)
- A function is taken to be **possiblyContinuous** at each point where it is undefined. (The choice between this and **notContinuous** is arbitrary.)
- If a function is not everywhere defined, it cannot be known to be bounded.  Arnold Neumaier, whose table this is, takes this view. I find it shaky but adopt it for simplicity.
- It is mathematically possible for a function to be everywhere defined and nowhere continuous, but such functions do not occur in practical interval computation, so this case is excluded.

Row 1: If an interval is **notValid**, then as said above it is given the value 0=**possibly** for all other decoration properties.

Row 2: If a valid interval is **notDefined**, it results from interval-evaluation of a function f which is undefined everywhere on its input box X . *Example.* $\sqrt{[-2, -1]}$ or $[0, 0]/[0, 0]$. Its interval part is the empty set; however by the above conventions this is classified **possiblyBounded**.

Row 3: If a valid interval is **possiblyDefined**, the underlying f is known to be somewhere but not everywhere defined on the input box. The above conventions imply: it cannot be **isContinuous**; in practice it is not **notContinuous**; and it cannot be known to be **isBounded**.

Row 4: For a valid, *isDefined* interval, by the above conventions, *notContinuous* is excluded but any other combination of *continuous* and *bounded* status is possible.

5.3.8. *Other operations on decorated intervals.* ⚠ It needs to be decided how intersection, hull and similar non-arithmetic operations handle decorations.

Also how decorations affect comparison functions: e.g. $([1, +\infty], \text{notBounded})$ presumably cannot be a subset of $([1, +\infty], \text{isBounded})$.

⚠ We need to specify how the interval part of a *notValid* interval behaves w.r.t. real-valued functions like radius (always NaN?) and “forget decoration” (gives Empty?).

5.4. Required operations.

5.4.1. Constructors and conversions.

⚠ This has not been voted on but I assume, for now, a subset of the operations in Vienna§2.5, §6.

Implementations shall provide, for each supported i-format, the following operations. Their names in this standard do not necessarily correspond to those that any particular language would use. The prefix **fmt** indicates some textual way to show the destination format of the operation.

fmtfloats2interval(\underline{x}, \bar{x}). If $\underline{x} \leq \bar{x}$ this returns the smallest interval of the given i-format containing the two floating point numbers \underline{x}, \bar{x} , flagged as *isValid*. If it is not true that $\underline{x} \leq \bar{x}$ (in particular if either of \underline{x}, \bar{x} is NaN) it returns the empty interval flagged as *notValid*.

The n-formats of \underline{x} and \bar{x} need not be the one associated with the destination i-format, and need not be the same.

fmtfloat2interval(x) has the same effect as **fmtfloats2interval**(x, x).

fmttext2interval(s) returns the smallest interval of the given i-format containing the mathematical interval defined by the text string s , and flagged as *isValid*. If s does not define an interval, it returns the empty interval flagged as *notValid*.

⚠ A motion is needed to decide the details. Vienna§6 has a specification we may like to follow. Meantime I assume, in examples, that things like **text2interval**("[1.2, 3.4]") have the natural effect.

When a newly constructed interval is flagged as *isValid*, it is also flagged as *isDefined* and *isContinuous*. The *bounded* attribute is set as follows.

5.4.2. Forward-mode elementary functions.

⚠ The following issues are outstanding:

- Accuracy model—still to be defined, e.g. Vienna “tightest, accurate, valid”.
- Details on evaluation of functions outside their domain—handled by the common exception handling scheme, which currently means decorated intervals as in motion 8 and 5.3.

⚠ Jürgen: I have taken the liberty of changing the “rough” interval-oriented domains and ranges, in the tables in your motion, to precise domain and range of each point function. Reason: this, plus the requirement to be an interval extension, sufficiently specifies the interval function.

The term *operation* includes functions normally written in function notation $f(x, y, \dots)$, as well as those normally written in binary operator notation. The names of operations in this standard do not necessarily correspond to those that any particular language would use. Table 2 lists operations that are *required* in the sense that:

- A programming-language-specific implementation shall provide an interval version of each function in the table that is provided by the underlying floating-point implementation in that language.
- An implementation that is not language-specific shall provide interval versions of all functions in the table.

name	definition	point function domain	point function range	accuracy mode	note
$\begin{cases} \text{add} \\ \text{sub} \\ \text{mul} \end{cases}$	$+, -, \times$	\mathbb{R}^2	\mathbb{R}	t	
div	\div	$\{(x, y) \in \mathbb{R}^2 y \neq 0\}$	\mathbb{R}	t	
sqr	x^2	\mathbb{R}	$[0, \infty)$	t	
pown	$x^p, p \in \mathbb{Z}$	$\begin{cases} \mathbb{R} & \text{if } p \geq 0 \\ \mathbb{R} \setminus \{0\} & \text{if } p < 0 \end{cases}$	$\begin{cases} \mathbb{R} & \text{if } p > 0 \text{ odd} \\ [0, \infty) & \text{if } p > 0 \text{ even} \\ \{1\} & \text{if } p = 0 \\ \mathbb{R} \setminus \{0\} & \text{if } p < 0 \text{ odd} \\ (0, \infty) & \text{if } p < 0 \text{ even} \end{cases}$?	1
pow	x^y	see 5.4.3	see 5.4.3	?	2
sqrt	\sqrt{x}	$[0, \infty)$	$[0, \infty)$	t	
$\begin{cases} \text{exp} \\ \text{exp2} \\ \text{exp10} \end{cases}$	b^x	\mathbb{R}	$(0, \infty)$?	3
$\begin{cases} \text{log} \\ \text{log2} \\ \text{log10} \end{cases}$	$\log_b x$	$(0, \infty)$	\mathbb{R}	?	3
$\begin{cases} \text{expm1} \\ \text{exp2m1} \\ \text{exp10m1} \end{cases}$	$b^x - 1$	\mathbb{R}	$(-1, \infty)$?	3
$\begin{cases} \text{logp1} \\ \text{log2p1} \\ \text{log10p1} \end{cases}$	$\log_b(x+1)$	$(-1, \infty)$	\mathbb{R}	?	3
sin		\mathbb{R}	$[-1, 1]$?	
cos		\mathbb{R}	$[-1, 1]$?	
tan		$\mathbb{R} \setminus \{(k + \frac{1}{2}\pi) k \in \mathbb{Z}\}$	\mathbb{R}	?	
asin		$[-1, 1]$	$[-\pi/2, \pi/2]$	t	4
acos		$[-1, 1]$	$[0, \pi]$	t	4
atan		\mathbb{R}	$(-\pi/2, \pi/2)$?	4
atan2		$\mathbb{R}^2 \setminus \{(0, 0)\}$	$(-\pi, \pi]$?	4, 5
sinh		\mathbb{R}	\mathbb{R}	?	
cosh		\mathbb{R}	$[1, \infty)$?	
tanh		\mathbb{R}	$(-1, 1)$?	
asinh		\mathbb{R}	\mathbb{R}	?	
acosh		$[1, \infty)$	$[0, \infty)$?	
atanh		$(-1, 1)$	\mathbb{R}	?	
abs	$ x $	\mathbb{R}	$[0, \infty)$	t	
rSqrt	$1/\sqrt{x}$	$(0, \infty)$	$(0, \infty)$	t	
hypot	$\sqrt{x^2 + y^2}$	\mathbb{R}^2	$[0, \infty)$	t	
compoundm1	$(1+x)^n - 1$	$(-1, \infty)$	$\begin{cases} (-1, \infty) & \text{if } n \neq 0 \\ \{0\} & \text{if } n = 0 \end{cases}$?	1

TABLE 2. Required elementary functions. The interval version is required to be an interval extension of the point function and to be of the specified accuracy.

Notes to Table 2:

1. The integer argument selects the proper function out of a parameterized family.
2. Defined in 5.4.3.
3. $b = e, 2$ or 10 , respectively
4. The ranges shown are the mathematical range of the point function. To ensure containment, the interval result may include values just outside the mathematical range.
5. $\text{atan2}(y, x)$ is the principal value of the argument (polar angle) of (x, y) in the plane.

⚠ I have included a column “accuracy mode” which assumes the Vienna scheme: t=tight, a=accurate, v=valid. I put ‘t’ against those that obviously must be ‘tight’. We need a motion to decide how to complete this.

[Note. The list includes all general-computational operations in 754§5.4 except **convertFromInt**, and many of the recommended functions in 754§9.2.]

Proving the correctness of interval computations relies on the Fundamental Theorem of Interval Arithmetic, which in turn relies on the relation between a point-function and its interval version. Therefore, for each point-function e , care is taken to define its domain D_e and its value at each point of D_e . This is mostly straightforward but needs care for functions with discontinuities, such as `pow()` and `atan2()`.

Each interval version shall be an interval extension, see 4.4.1, of the corresponding point function. Table 2 lists the name of the function, its mathematical definition where not obvious, and its domain and range.

AN
20100129

5.4.3. *General Power Function.* Usually the interval power function x^y is defined as extension of the real function $e^{y \cdot \ln(x)}$. Hence, x has to be positive.

But actually $x^{p/q}$ with $x < 0$ is definable for rational exponents, if q is odd. If p is odd the result is negative otherwise positive. Since the rationals are dense in the reals, we can find a rational number with odd denominator and odd or even numerator in each neighborhood of a negative real number. Hence, we can define an interval version of the power function that directly delivers the interval between the 2 values.

The general interval power function called `pow()` is definable as follows (Let \sqcup compute the interval hull of its 2 operands.)

$$(1) \quad \begin{aligned} &\text{pow} : \overline{\mathbb{R}} \times \overline{\mathbb{R}} \rightarrow \overline{\mathbb{R}}. \\ &\text{If } x > 0 : \text{pow}(x, y) = \exp(y \cdot \ln(x)) \\ &\text{If } x < 0 : \text{pow}(x, y) = -\exp(y \cdot \ln(|x|)) \sqcup \exp(y \cdot \ln(|x|)) \\ &\text{If } 0 \in x : \text{pow}(x, y) = \text{pow}([x, 0], y) \sqcup \text{pow}([0, 0], y) \sqcup \text{pow}((0, x], y) \end{aligned}$$

where $\text{pow}([0, 0], y) = [0, 0]$, if $\exists 0 < y \in \mathbf{y}$. \triangle and $\text{pow}([0, 0], y)$ is empty otherwise?

\triangle I have a problem with `compoundm1()`, as well as the 754-2008 function `compound()`. Namely you often want to convert monthly interest rate r to annual rate R , which is given by

$$R = \text{compoundm1}(r, 12)$$

but equally often want to convert from annual rate to monthly, given by

$$r = \text{compoundm1}(R, 1/12).$$

So isn't it pretty pointless if y must be an integer?

\triangle The position paper notes that interval `pow()` is not an extension of any point-function version of `pow()`. My view is that to preserve the FTIA, it must be so.

In fact I believe (1) is an extension of the function

$$\begin{aligned} &\text{pow} : \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}. \\ &\text{If } x > 0 : \text{pow}(x, y) = \exp(y \cdot \ln(x)) \\ &\text{If } x = 0, y > 0 : \text{pow}(x, y) = 0 \\ &\text{If } x < 0, y \text{ rational} = p/q \text{ in lowest terms :} \\ &\quad \text{if } p \text{ odd: } \text{pow}(x, y) = -\exp(y \cdot \ln(|x|)) \\ &\quad \text{if } p \text{ even: } \text{pow}(x, y) = \exp(y \cdot \ln(|x|)) \\ &\text{else } \text{pow}(x, y) \text{ undefined.} \end{aligned}$$

Two aspects need care about Jürgen's interval `pow()`. (i) It seems to have $\text{pow}([0, 0], [0, 0]) = \text{Empty}$, which is why I made $\text{pow}(0, 0) = \text{undefined}$ for the point version. (ii) For $x < 0$, it doesn't take any notice of whether y , if a singleton, is rational, so e.g. $\text{pow}([-3, -3], [2, 2])$ seems to be $[-(3^2), +(3^2)] = [-9, 9]$ (I would expect $[9, 9]$); while $\text{pow}([-3, -3], [0.5, 0.5])$ seems to be

$[-(3^{0.5}), +(3^{0.5})] = [-\sqrt{3}, \sqrt{3}]$ (I would expect Empty). I find both of these strange. Jürgen, did you mean this? Or have I mis-read the definition?

5.4.4. *Reverse-mode elementary functions.*

⚠ The list from a revised Motion 11 will go here.

5.4.5. *Comparison operations.*

⚠ No motion about these yet.

5.4.6. *Other operations.*

⚠ I expect intersection, midpoint, etc., will go here.

5.4.7. *Reduction operations.* Implementations shall provide an exact dot product operation.

⚠ Details TBW

5.5. Recommended operations (informative).

5.5.1. *Forward-mode elementary functions.*

Language standards should define interval versions of as many of the functions in Table 3 as is appropriate to the language.

⚠ Should we include a recommended accuracy mode in this table?

5.5.2. *Reverse-mode elementary functions.*

⚠ The list from a revised Motion 11 will go here.

5.5.3. *Other operations.*

⚠ The list from a revised Motion 11 will go here.

5.6. Sharpness measures.

⚠ E.g. Vienna's tight, accurate, valid. No motion about these yet!

name	definition	domain	range	notes
sign		\mathbb{R}	$\{-1, 0, 1\}$	
ceil, floor		\mathbb{R}	\mathbb{Z}	
nint, trunc		\mathbb{R}	\mathbb{Z}	
rootn	$x^{1/q}, q \in \mathbb{Z} \setminus \{0\}$	$\begin{cases} \mathbb{R} & \text{if } q > 0 \text{ odd} \\ [0, \infty) & \text{if } q > 0 \text{ even} \\ \mathbb{R} \setminus \{0\} & \text{if } q < 0 \text{ odd} \\ (0, \infty) & \text{if } q < 0 \text{ even} \end{cases}$	same as domain	1
powr	$x^{p/q}$	$\begin{cases} \mathbb{R} & \text{if } p \geq 0, q \text{ odd} \\ [0, \infty) & \text{if } p > 0, q \text{ even} \\ \mathbb{R} \setminus \{0\} & \text{if } p < 0, q \text{ odd} \\ (0, \infty) & \text{if } p < 0, q \text{ even} \end{cases}$	$\begin{cases} \mathbb{R} & \text{if } p > 0, pq \text{ odd} \\ [0, \infty) & \text{if } p > 0, pq \text{ even} \\ \{1\} & \text{if } p = 0 \\ \mathbb{R} \setminus \{0\} & \text{if } p < 0, pq \text{ odd} \\ (0, \infty) & \text{if } p < 0, pq \text{ even} \end{cases}$	1
$p, q \in \mathbb{Z}$, normalized by ($q > 0$, p/q in lowest terms, i.e. $\text{HCF}(p, q) = 1$.)				
sinPi	$\sin(\pi x)$	\mathbb{R}	$[-1, 1]$	2
cosPi	$\cos(\pi x)$	\mathbb{R}	$[-1, 1]$	2
atanPi	$\arctan x / \pi$	\mathbb{R}	$[-1/2, 1/2]$	2
atan2Pi	$\text{atan2}(y, x) / \pi$	$\mathbb{R} \times \mathbb{R}$	$[-1, 1]$	2
exp1x	$(e^x - 1)/x$	\mathbb{R}	\mathbb{R}	
exp2x	$(e^x - 1 - x)/x^2$	\mathbb{R}	$(0, \infty)$	
cos2	$(\cos x - 1)/x^2$	\mathbb{R}	$(-\infty, 0]$	
sin3	$3(\sin x - x)/x^3$	\mathbb{R}	$[-1/2, \infty)$	
cosh2	$(\cosh x - 1)/x^2$	\mathbb{R}	$(0, \infty)$	
sinh3	$3(\sinh x - x)/x^3$	\mathbb{R}	$[1/2, \infty)$	
gamma	$\Gamma(x)$	\mathbb{R}	\mathbb{R}	
lgamma	$\log \Gamma(x) $	\mathbb{R}	\mathbb{R}	
erf	error function	\mathbb{R}	$(-1, 1)$	
erfc	complementary error function	\mathbb{R}	$(0, 2)$	

TABLE 3. Recommended elementary functions

Notes to Table 3:

1. The integer arguments p, q select the proper function out of a parameterized family.
2. These functions avoid the behavior noted in Table 2, note 4.

⚠ Query: **ceil, floor, trunc** are names used in C standard. If **nint** is 'round to nearest integer' shouldn't it be called **round**, as in C standard?

⚠ This list is probably controversial: other functions may have a stronger claim for inclusion than some of the present members.

6. LEVEL 3 DESCRIPTION

6.1. Representation of intervals by lower/upper bounds.

An implementation may choose any means to represent a level 2 interval datum \mathbf{x} , provided that it shall be possible to retrieve the bounds of \mathbf{x} exactly. This is captured by the following definition.

A **concrete interval format (ci-format)** is a surjective mapping from a set C of instances of a data structure to an associated level 2 i-format.

[Note. Typical choices are

- **inf-sup representation.** The data structure is an ordered pair $(f1, f2)$ of floatingpoint datums. A nonempty \mathbb{F} -interval $\mathbf{x} = [\underline{x}, \bar{x}]$ is represented by (\underline{x}, \bar{x}) . The empty set may be represented by (NaN, NaN) .
- **neginf-sup representation.** As the previous, but $\mathbf{x} = [\underline{x}, \bar{x}]$ is represented by $(-\underline{x}, \bar{x})$.

The above two ci-formats use essentially the same data structure, and represent the same i-format, but the mappings are different.

Multi-precision interval packages may represent an interval $\mathbf{x} = [\underline{x}, \bar{x}]$ by a triple $(\hat{x}, \underline{\delta}, \bar{\delta})$ where \hat{x} is some point in \mathbf{x} , and $\underline{\delta}$ and $\bar{\delta}$ are very small numbers, and $\mathbf{x} = [\hat{x} + \underline{\delta}, \hat{x} + \bar{\delta}]$ exactly; or in other ways.]

6.2. Format conversion.

On 754 systems, level 2 interval format conversion (the hull operation) should be implemented in terms of the floating-point operations *formatOf-convertFormat* defined in 754§5.4.2, with the appropriate outward rounding.

6.3. Interchange formats.

⚠ We need a motion on this subclause, which was my invention.

The purpose of interchange formats is to allow the loss-free exchange of level 2 interval data between 754-conforming implementations. This is done by imposing a standard level 3 (and hence level 4) representation. Let \mathbb{F} be a 754 format and \mathbf{x} a (bare) \mathbb{F} -interval datum, so that its lower bound \underline{x} and upper bound \bar{x} are \mathbb{F} -numbers. An interchange format of \mathbf{x} is the concatenation of the bit strings of the \mathbb{F} -representations of \underline{x} and \bar{x} in that order, where:

- 0 shall be represented as +0.
- For decimal formats, any member of the number's cohort is permitted. The choice is implementation-defined.
- When \mathbf{x} is the empty set, \underline{x} and \bar{x} are taken as NaN. Whether qNaN or sNaN is used, and any payload, are implementation-defined.

[Note. The above rules imply an interval has a unique interchange representation if it is nonempty and in a binary format, but not generally otherwise. The reason for the rules is that the sign of a zero endpoint cannot convey any information relevant to intervals; but an implementation may potentially use cohort information, or a NaN payload.]

The interchange format for a decoration comprises the bit strings for each trit, concatenated in the order **valid**, **defined**, **continuous**, **bounded**. The bit string values for a trit are 11, 00, 01 for $-$, 0 , $+$ respectively (the twos-complement representations of $-1, 0, 1$ in two bits). Thus a decoration occupies one byte.

The interchange format for a decorated interval is the concatenation of those for its interval and decoration parts, in that order.

A 754-conforming implementation shall provide an interchange format for each supported 754 interval format. Interchange formats for non-754 interval formats, and on non-754 systems, are implementation-defined.

6.4. Support levels for interval elementary functions. The Fundamental Theorem of Interval Arithmetic (FTIA) relies on each point elementary function e in a real expression being replaced by an interval version \mathbf{e} . Mathematically, \mathbf{e} may be an arbitrary interval extension of e , and its arguments and result are not limited by any concrete interval format.

A level 2 interval version is implemented at level 3 in terms of concrete formats such as binary64. The standard defines three levels of mixed format support, below. For each one, \mathbf{e} delivers a result of a specified ci-format \mathbb{F} , from operands of a limited number of ci-formats.

Implementations shall give at least SRSF support to all supported elementary functions. 754-conforming implementations shall give at least SRMF support.

SRSF. Single-radix, single-format. In SRSF support, e has an interval version \mathbf{e} that takes \mathbb{F} -interval operand(s) and gives an \mathbb{F} -interval result. Thus explicit format conversion is needed for any operand of a different format from \mathbb{F} .

SRMF. Single-radix, mixed-format. In SRMF support, e has an interval version \mathbf{e} that takes operand(s) of any supported interval format of the same radix as \mathbb{F} , and gives an \mathbb{F} -interval result. Thus explicit format conversion is needed for any operand whose format has a different radix from that of \mathbb{F} .

MRMF. Mixed-radix, mixed-format. In MRMF support, e has an interval version \mathbf{e} that takes operand(s) of any supported interval format, and gives an \mathbb{F} -interval result. Thus no explicit format conversion is required for any operand.

SRMF includes SRSF (SRMF support provides SRSF in particular); and MRMF includes SRMF.

MRMF support, and mixed-format interval expressions of more than one operation, are considered to be language issues. Recommendations on them are in Appendix A.

[Note. For a 754 formatOf floating-point operation, for any combination of input and output formats of the same radix, the correctly rounded result is produced, eliminating the risk of “double rounding” error in mixed-format operations.

Most algorithms for the basic interval operations, in particular those in 6.5, can exploit the formatOf feature. That is, they can be written, in terms of point operations, so that arbitrary mixed formats of the same radix can be handled by essentially the same code, while remaining optimally tight at the level of a single interval operation.]

6.5. Operation tables for basic interval operations.

⚠ This was the trickiest part for me to reconstruct after losing all the Latex files of Version 01 when I was burgled. Prof Kulisch: will you or a colleague please check this carefully?

The tables in this subclause are an explicit realization of the general definition of interval operations given in 4.4.1. They are not normative, but are one possible basis for coding the interval versions of $+$, $-$, \times , \div . For fuller details see [1], [2].

Notation. In addition to the notation in 2.1 this subclause also defines, for a specified n-format \mathbb{F} :

- ∇, Δ : the roundings downwards and upwards,
- ∇ , etc. : the operations for elements of \mathbb{F} with rounding downwards,
- Δ , etc. : the operations for elements of \mathbb{F} with rounding upwards.
- \mathbf{s} : the same as $\text{hull}_{\mathbb{F}}(\mathbf{s})$, the \mathbb{F} -hull of a subset \mathbf{s} of \mathbb{R} .

For intervals $\mathbf{a}, \mathbf{b} \in \mathbb{IR}$ arithmetic operations are defined as set operations in \mathbb{R} by:

$$(2) \quad \mathbf{a} \circ \mathbf{b} := \{ a \circ b \mid a \in \mathbf{a} \wedge b \in \mathbf{b} \wedge a \circ b \text{ is defined} \},$$

for all $\mathbf{a}, \mathbf{b} \in \mathbb{IR}$ and $\circ \in \{+, -, *, /\}$. If $0 \notin \mathbf{b}$ in case of division, then for all $\mathbf{a}, \mathbf{b} \in \mathbb{IR}$ also $\mathbf{a} \circ \mathbf{b} \in \mathbb{IR}$.

Then binary arithmetic operations in \mathbb{IF} are uniquely defined by:

$$(3) \quad a \diamond b := \diamond(a \circ b),$$

for all $\mathbf{a}, \mathbf{b} \in \mathbb{IF}$ and all $\circ \in \{+, -, *, /\}$. For division we assume again that $0 \notin \mathbf{b}$.

For intervals $\mathbf{a} = [a_1, a_2], \mathbf{b} = [b_1, b_2] \in \mathbb{IF}$ these operations $\diamond, \circ \in \{+, -, *, /\}$, in \mathbb{IF} have the property

$$a \diamond b = \left[\min_{i,j=1,2} (a_i \nabla b_j), \max_{i,j=1,2} (a_i \triangle b_j) \right],$$

or with the monotone roundings ∇ and \triangle

$$a \diamond b = \left[\nabla \min_{i,j=1,2} (a_i \circ b_j), \triangle \max_{i,j=1,2} (a_i \circ b_j) \right].$$

The above operations, and the unary operation $-\mathbf{a}$, can be expressed by more explicit formulas as shown in the following tables. There the operator symbols for intervals are simply denoted by $+, -, *, \text{ and } /$.

Minus operator $-\mathbf{a} = [-a_2, -a_1]$.

Addition $[a_1, a_2] + [b_1, b_2] = [a_1 \nabla b_1, a_2 \triangle b_2]$.

Subtraction $[a_1, a_2] - [b_1, b_2] = [a_1 \nabla b_2, a_2 \triangle b_1]$.

Multiplication $[a_1, a_2] * [b_1, b_2]$	$[b_1, b_2]$ $b_2 \leq 0$	$[b_1, b_2]$ $b_1 < 0 < b_2$	$[b_1, b_2]$ $b_1 \geq 0$
$[a_1, a_2], a_2 \leq 0$	$[a_2 \nabla b_2, a_1 \triangle b_1]$	$[a_1 \nabla b_2, a_1 \triangle b_1]$	$[a_1 \nabla b_2, a_2 \triangle b_1]$
$a_1 < 0 < a_2$	$[a_2 \nabla b_1, a_1 \triangle b_1]$	$[\min(a_1 \nabla b_2, a_2 \nabla b_1), \max(a_1 \triangle b_1, a_2 \triangle b_2)]$	$[a_1 \nabla b_2, a_2 \triangle b_2]$
$[a_1, a_2], a_1 \geq 0$	$[a_2 \nabla b_1, a_1 \triangle b_2]$	$[a_2 \nabla b_1, a_2 \triangle b_2]$	$[a_1 \nabla b_1, a_2 \triangle b_2]$

Division, $0 \notin \mathbf{b}$ $[a_1, a_2] / [b_1, b_2]$	$[b_1, b_2]$ $b_2 < 0$	$[b_1, b_2]$ $b_1 > 0$
$[a_1, a_2], a_2 \leq 0$	$[a_2 \nabla b_1, a_1 \triangle b_2]$	$[a_1 \nabla b_1, a_2 \triangle b_2]$
$[a_1, a_2], a_1 < 0 < a_2$	$[a_2 \nabla b_2, a_1 \triangle b_2]$	$[a_1 \nabla b_1, a_2 \triangle b_1]$
$[a_1, a_2], a_1 \geq 0$	$[a_2 \nabla b_2, a_1 \triangle b_1]$	$[a_1 \nabla b_2, a_2 \triangle b_1]$

The general rule for computing the set \mathbf{a}/\mathbf{b} with $0 \in \mathbf{b}$ is to remove its zero from the interval \mathbf{b} and perform the division with the remaining set. Whenever the zero in \mathbf{b} coincides with a bound of the interval \mathbf{b} the result of the division can directly be obtained from the above table for division with $0 \notin \mathbf{b}$ by the limit process $b_1 \rightarrow 0$ or $b_2 \rightarrow 0$ respectively. The results are shown in the following table. Here, the parentheses stress that the bounds $-\infty$ and $+\infty$ are not elements of the interval.

Division, $0 \in \mathbf{b}$ $[a_1, a_2] / [b_1, b_2]$	$\mathbf{b} =$ $[0, 0]$	$[b_1, b_2]$ $b_1 < b_2 = 0$	$[b_1, b_2]$ $0 = b_1 < b_2$
$[a_1, a_2] = [0, 0]$	\emptyset	$[0, 0]$	$[0, 0]$
$a_1 < a_2 \leq 0$	\emptyset	$[a_2 \nabla b_1, +\infty)$	$(-\infty, a_2 \triangle b_2]$
$[a_1, a_2], a_1 < 0 < a_2$	\emptyset	$(-\infty, +\infty)$	$(-\infty, +\infty)$
$0 \leq a_1 < a_2$	\emptyset	$(-\infty, a_1 \triangle b_1]$	$[a_1 \nabla b_2, +\infty)$

When zero is an interior point of the denominator, the set $[b_1, b_2]$ devolves into the two distinct sets $[b_1, 0)$ and $(0, b_2]$ and division by $[b_1, b_2]$ actually means two divisions. The results of the two divisions are already shown in the table for division with $0 \in \mathbf{b}$.

In the user's program, however, the two divisions appear as a single operation, as division by an interval $\mathbf{b} = [b_1, b_2]$ with $b_1 < 0 < b_2$. So an arithmetic operation in the user's program delivers two distinct results.

One solution is for the computer to provide a flag for distinct intervals, that is raised if the divisor is an interval that contains zero as an interior point. The user may then apply a routine of their choice to deal with the situation as is appropriate for the application.

This routine could be: return the entire set of real numbers $(-\infty, +\infty)$ as result and continue the computation, or continue the computation with one of the sets and ignore the other one, or put one of the sets on a list and continue the computation with the other one, or modify the operands and recompute, or stop computing, or some other action.

An alternative would be to provide a second division which in case of division by an interval that contains zero as an interior point generally delivers the result $(-\infty, +\infty)$. Then the user can decide when to use which division in his program.

The above operation tables have assumed that \mathbf{a} and \mathbf{b} are nonempty and bounded. To extend them to general intervals, the first rule is that any operation with the empty set \emptyset returns the empty set.

Then, the above tables extend to nonempty, possibly unbounded, intervals of $\overline{\mathbb{R}}$ by using the standard formulae for arithmetic operations involving $\pm\infty$, which are implemented in 754, together with one rule that goes beyond 754 arithmetic:

$$0 * (-\infty) = (-\infty) * 0 = 0 * (+\infty) = (+\infty) * 0 = 0.$$

This rule is not a new mathematical law, merely a short cut to easily compute the bounds of the result of multiplication on unbounded intervals.

7. LEVEL 4 DESCRIPTION

TBW

APPENDIX A. LANGUAGE ISSUES

TBW

APPENDIX B. CONVENTIONS FOR EXAMPLES IN THE STANDARD

To explain how mathematical statements like $y = \sqrt{[-1, 4]} + 1$ are interpreted in terms of P1788 basic operations.

May become the section on Level 2 semantics. TBW

REFERENCES

- [1] Ulrich Kulisch: Complete Interval Arithmetic and its Implementation on the Computer, position paper and the Dagstuhl 2008 proceedings.
- [2] Ulrich Kulisch: Computer Arithmetic and Validity – Theory, Implementation, and Applications, de Gruyter, Berlin, New York, 2008.
- [3] Arnold Neumaier: Vienna Proposal for Interval Standardization, December 2008.
- [4] John Pryce: Text and Rationale for Motion 6: Multi-Format Support.

⚠ References to be extended, obviously. These are from Kulisch's operation table position paper.